TOWARD A DYNAMIC FEEDBACK THEORY OF OPEN ONLINE COLLABORATION COMMUNITIES

by

Vedat G. Diker

A Dissertation
Submitted to the University at Albany, State University of New York
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

School of Information Science and Policy
Information Science
2003

Toward a Dynamic Feedback Theory of

Open Online Collaboration Communities

by

Vedat G. Diker

ABSTRACT

This study posits a theory of open online collaboration communities in the form of a dynamic feedback framework and provides implications about the potential consequences of policy interventions for improving the performance of such communities. The study was carried out in three phases. During the first phase, several theoretical approaches were integrated to build a dynamic feedback model of a hypothetical open source software development community. The second phase involved the administration and analysis of a series of interviews with the members of an actual instructional material development community, in order to test the applicability of a generalized version of the open source software development model and its implications to that specific community. In the third phase, the implications of the initial model and the findings of the interviews were integrated to posit a theory of open online collaboration communities, in the form of a dynamic feedback framework. The study provided theoretical and practical implications about open online collaboration communities, and thus, contributed to several streams of literature, generated critical insights for managing open online collaboration communities, and laid a foundation for a variety of potential future research studies.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

Many individuals contributed to this study directly and indirectly. Most important of all, I am grateful to my advisors David F. Andersen, George P. Richardson, Deborah L. Andersen and R. Karl Rethemeyer for guiding me through the many stages of my dissertation and enabling me to earn my degree. David has been a very significant figure in my life since I started my doctoral study in Albany. He helped me with all sorts of problems, big and small, be it academic, personal or financial. George is the most skilful modeler and teacher I have ever met. I believe I had the privilege of being taught by excellent teachers throughout my life, and George was an excellent final note, who has become one of my role models as an educator. Deborah taught me a lot about empirical research and helped me build solid social research skills. She was an extremely patient reviewer of the numerous drafts of this dissertation, and taught me how to "see one, do one and teach one." Karl helped me with finding my way in the academic job market, as well as guiding me through the literature review and the model-based interview development stages of this study. His guidance has had tremendous impact on my career choices.

I am grateful to Roberta L. Spencer, my former boss and my friend, for helping me tremendously with all the aspects of my life since I met her more than six year's ago in Istanbul. She was the best boss ever, and will stay a very special friend. I also would like to thank Jennifer I. Rowe, Robin S. Langer and Joan M. Yanni of the System Dynamics Society for their friendship and support throughout the years.

Many fellow students and graduates of the Information Science and Public Administration doctoral programs have contributed to this dissertation. I am thankful to

Michael A. Deegan, Hassan S. Dibadj, Luis F. Luna-Reyes, Roderick H. MacDonald, Ignacio J. Martinez-Moyano, Mohammad T. Mojtahedzadeh, Nandhini Rangarajan, Eliot H. Rich, Hans J. Scholl, Silvia Ulli-Beer, Aldo A. Zagonel-Dos Santos and many others for their ideas, suggestions and personal support during my study. They made Albany a fun place to stay. A very special thanks goes to Luis F. Luna-Reyes for helping me with the technical details of submitting my dissertation, among many other things. Not many friends would do the things he did for me. Also, a special thanks to Hans J. Scholl for "initiating" me with the open source idea, which finally gave way to my dissertation topic.

I also would like to thank the members of the system dynamics K-12 community who kindly agreed to participate in the interviews and helped me in gathering crucial information about their community.

I am also grateful to my father Omer Diker and my mother Ozden Diker for raising me to be the person I am. I am particularly thankful to my father for flying thousands of miles to be with me and help me during the final stages of my study and my transition -- and I am thankful to my mother for agreeing to send him over for three months. I am also thankful to my sister Nur O. Eruzel for being one of the best friends in my life.

Last, but definitely not the least, I am forever thankful to my dear wife Zeynep Diker and my precious little son Omer F. Diker. They helped me through the frustrations and the occasional gloom that necessarily accompany working on a dissertation. They had to sacrifice a lot so that daddy could work. Zeynep, here is another step in our

journey together; let’s hope it takes us to the place we want to be. And Omer, I hope my accomplishment becomes a foundation for many more by you.

CHAPTER 1 -- INTRODUCTION AND RESEARCH PURPOSE

The foundation of this dissertation was laid in early 1999, when I started working on a term project with Jochen Scholl, a fellow graduate of the University at Albany's Interdisciplinary Doctoral Program in Information Science. Jochen and I were first year students in the doctoral program and we were looking for an interesting problem to model as our term project for an intermediate level system dynamics course taught by George Richardson. 1999 was the year of "DOJ vs. Microsoft", arguably the most critical monopoly case in the history of software industry. It did not take us too long to pick the cutthroat competition in PC operating systems market as our project topic. After some preliminary research, we concluded that we should build the model around the competition between Microsoft's Windows operating system and the competitor product that poses the greatest threat to Windows' market share: Linux. In our project report and the two conference papers that followed it we argued that Linux was the only imminent rival to Windows that could break Microsoft's vicious market cycle that could be summarized as "leverage applications with operating system -- leverage operating system with applications." We argued that Linux had the potential to break that cycle due to being an open source software project, which was not driven by market share or revenue (Diker and Scholl 1999, Diker and Scholl 2001).

I loved the experience of working on the operating systems market model. The dynamics in the operating systems market were quite interesting. However, I soon concluded that the dynamics of open source software development itself were even more interesting. In less than a year, I knew that my dissertation would involve open source software development phenomenon in one way or another. What fascinated me most

about open source software development was the idea that there were thousands of people ready and willing to write software for free -- an activity which brought some other people six-digit salaries. As I continued to read cases in open source software development, I began to identify certain dynamics, which I believed were responsible for making open source software projects succeed or fail. I concluded that I could make a contribution to the theory and practice of open source software development by identifying those dynamics and the ways to leverage them in order to increase the performance of open source software communities.

While my interest in open source software development was growing deeper, my advisor David Andersen suggested to me that I study as a potential dissertation topic a community of teachers and researchers working on developing instructional materials to introduce systems thinking and system dynamics concepts to K through 12 students. The community was making efforts to use the Internet for engaging a wider audience in its instructional material development and dissemination activities. Because I have interests in both educational issues and Internet applications, I liked David's idea quite a lot. When I started my preliminary study about the community, I thought that what they were doing was in many ways similar to what open source software developing communities were doing. Here was a group of people using the Internet to work collaboratively on developing and disseminating a freely available information product without direct financial compensation, and with the option of building on one another's work.

I started looking for a conceptual basis that would let me study open source software development and collaborative instructional development through a single lens. My search yielded two important findings. First, I found that there were many other

communities using the open source model to develop and disseminate a variety of information products. I heard and read about concepts like Internet-based collaborative authoring and collaborative music making. These were basically newly emerging concepts that had not become academic research topics, and in all honesty, not all of them looked promising and convincing at first sight. However, some of them, such as collaborative instructional development seemed to hold a potential for shaping the way we will build, disseminate and access content in the near future. In particular, open source software development and collaborative instructional material development approaches seem to hold a considerable potential for combining the voluntary contributions of thousands of people and putting the outcome to the use of all the people on the world. If this vision becomes a reality, it can make a tremendous difference for all mankind.

My second finding was the concept of online communities, which seemed fit to define both open source software communities and instructional development communities. In the general sense, an online community is a somewhat structured group of people sharing work, ideas, or other aspects of life in Internet-based environments, such as newsgroups, mailing lists, and message boards. An online community may consist of employees of a corporation, customers of a company, members of a society, or any group of people that shares a common interest in collaborating on the Internet around a certain aspect of their lives (Williams and Cothrel 2000). Online communities have recently attracted attention from both the academic and the corporate world (Preece 2000 pp.6-8). Online communities appeared soon after the Internet came into mainstream use and have spread together with the Internet.

As I studied online communities, I concluded that both open source software communities and instructional development communities could be defined as online communities, which are formed by loosely connected groups of people using the Internet as a medium for carrying out collaborative projects for developing and disseminating a variety of information products such as software, instructional materials, reports, presentations and multimedia files. This type of online community generally involves little or no barriers to entry and contribution. The information products produced by the members of the community are generally open for use and modification by anybody. I refer to such communities as "open online collaboration communities" throughout this dissertation.

Research Purpose

The open source model and open online collaboration communities may dramatically change the way we developed, disseminate and access digital content in the near future. However, the dynamic interactions between the determinants of success in open online collaboration communities such as barriers to entry, motivation, participation, collaboration and the quality of products, have not been fully explored and theorized. The stakeholders in such projects do not have the means to test policies to improve performance. Instead, they rely on a combination of personal experience, intuition and anecdotal guidelines derived from the experiences of other, similar projects.

As an example, the nature and level of motivation of contributors in an open online collaboration community appears to be an important driving factor behind the community's growth and overall success. Accordingly, the leaders of an open online collaboration community might be able to steer their community to success by managing

the motivations of the contributors in an educated manner. However, it is not all that obvious what would motivate or turn off contributors in an open online collaboration community setting. Theoretical approaches provide vague or contradicting implications about motivations factors in online communities, which makes it hard to develop hypotheses about this factor. For example, based on the literature, we can argue that as the collection or the product that is developed by an open online collaboration community matures, contributors would be more motivated to participate in development. We can also argue, again based on the literature, that as the collection or the product matures, contributors would be less motivated due to decreasing opportunities for making contributions. In fact, both of these arguments may hold for some open online collaboration communities. The relationship between the quality of the collection or the product, and the motivation level of the contributors is not clear either. We can argue that contributors would be more motivated to work on a product that they considered to be of high quality, since they would want to be among the developers of a reputable product. On the other hand, contributors may be motivated more to contribute when they observe that the quality of the product is low and their help is needed to make it better. Obviously, more theoretical and empirical research is needed to learn about open online collaboration communities, and how they can be made more successful.

One important reason for the existing void in the literature about open online collaboration communities is that such communities have not been studied as a distinct type of online communities. The general approach in the literature, as will be seen in the literature review section, is either to study online communities as a homogenous group, or to study just a limited group of communities that would fall into the definition of open

online collaboration communities, as in the case of open source software development communities.

As a consequence of all these considerations, this study has two main research purposes:

1) to develop and establish a definition of open online collaboration communities, supported by a dynamic feedback framework that is applicable to a range of open online collaboration communities,

2) to outline and analyze several policy options for improving the performance of open online collaboration communities.

Structure of the Study

This study analyzed and modeled a hypothetical case within the definition of open online collaboration communities, and tested the applicability of the model to an instructional material development community case, in order to posit a theory of open online collaboration communities in the form of a dynamic feedback framework. The study integrated several theoretical approaches to the study of online communities. A review of online communities and open source software development literatures provided implications for building an initial dynamic feedback simulation (system dynamics) model of a hypothetical open source software development community. Implications of several studies that had applied system dynamics to software project management were also used in developing the initial model.

The initial open source software development (OSSD) model was simulated under different external conditions and policy options in order to test whether it exhibited

plausible behavior for a wide range of parametric conditions, and to observe the potential consequences of different approaches to improving the system performance. The model and its implications were also tested for applicability to a wider range of open online collaboration communities through a series of interviews with the members of an actual instructional material development community.

The implications of the initial model and the findings of the interviews were integrated to build a dynamic feedback framework, which serves as a theoretical foundation for studying phenomena related to open online collaboration communities. The dynamic feedback framework has the potential for being further developed into a dynamic feedback simulation model, which would serve as a platform for testing the consequences of different external conditions and policy options in a wider range of open online collaboration communities. The framework can also be used as a theoretical basis for developing and articulating hypotheses for empirical studies on open online collaboration communities.

The study contributed to online communities, open source software development and system dynamics literatures. It also provided critical implications for practice, including the potential consequences of several policy options for improving the performance of open online collaboration communities in terms of product quality, product functionality, community growth and participant talent. The study also laid out a variety of topics for potential future research studies, including both theoretical and empirical ones.

This dissertation is organized as follows: This chapter introduces the study and sets forth the research purposes. Chapter 2 defines open online collaboration communities

as a special type of online communities, summarizes the findings of the literature review, and introduces a dynamic hypothesis based on these findings. Chapter 3 introduces the methods used in this study and discusses the research design. Chapter 4 introduces a system dynamics model of a hypothetical open source software development community. Chapter 5 summarizes the tests and analyses done on the open source software development model. Chapter 6 discusses the findings of a series interviews done with the members of an instructional material development community in order to test the applicability of the initial system dynamics model and its implications to other open online collaboration communities. Chapter 7 introduces a dynamic feedback framework for studying open online collaboration communities, which was based on the implications of the initial system dynamics model and the interviews, and discusses the contributions of this study, together with potential future research opportunities.

The next chapter introduces the concept of open online collaboration communities and summarizes the findings of the literature review. The dynamic hypothesis that led to the initial open source software development model is also introduced in the next chapter.

CHAPTER 2 -- PROBLEM BACKGROUND AND LITERATURE REVIEW

2.1. Online Communities

The emergence of online communities is a phenomenon that has attracted attention from both the academic and the corporate world over the last ten years. Online communities appeared soon after the Internet came into mainstream use, and have spread together with the Internet. There are many definitions of an "online community," and each of these definitions draws a conceptual boundary that includes certain online communities and excludes others (Preece 2000 pp.8-17). In the general sense, an online community is a somewhat structured group of people sharing work, ideas, or other aspects of life in Internet-based environments, such as newsgroups, mailing lists or message boards. An online community may consist of employees of a corporation, customers of a company, members of a society, or any group of people that shares a common interest in collaborating on the Internet around a certain aspect of their lives (Williams and Cothrel 2000).

There have been several attempts to classify online communities. Hagel and Armstrong (1997 pp.18-23) suggested a classification based on the needs of the community members:

(1) *Communities of interest*: These are online communities whose members are gathered around a topic of shared interest or expertise that they discuss, such as Usenet groups.

(2) *Communities of relationship*: These communities bring together people with similar experiences and personal agendas to build relationship and share their experiences about the relevant topic.

(3) *Communities of fantasy*: These are communities where people come together to play and entertain within a virtual world, such as multi-user dungeons.

(4) *Communities of transaction*: These are communities of people that come together to perform economic exchange and produce economic value, such as business-to-business market communities.

Hagel and Armstrong's need-based criteria approach is a very broad way of classifying online communities. Although it can be useful for classifying online communities for certain purposes, it is not the only possible classification approach. Other researchers suggested more detailed classification schema, using multiple criteria as the basis for classification.

Lazar and Preece (1998) suggested a list of four classification criteria for online communities. The authors argue that online communities can be classified based on:

(1) *Attributes*: This classification criterion is somewhat similar to Hagel and Armstrong's (1997) need-based classification. Some of the attributes Lazar and Preece (1998 pp.84-85) suggested are existence of a shared goal or interest among the members, intense interaction and emotional ties between members, existence of shared activities, and support between members. Lazar and Preece suggested two other important attributes, namely the population size of the community, and existence of social conventions, language and protocols. They quoted Gates, arguing that the value of an

online community for its members increases as the population size increases (Gates 1995). They also quoted Reid's (1996) argument that an online community should have social conventions so that the members can communicate as they intend.

(2) *Supporting Software*: This criterion is based on the premise that the software used to facilitate interaction between the members affects the community to the point of shaping it. Lazar and Preece mentioned listservs, newsgroup software, bulletin boards, Internet Relay Chat (IRC) and multi-user dungeon software (MUD) as examples for community supporting software; however, they did not give an explicit classification of communities based on supporting software (1998 pp.85).

(3) *Relationship to Physical Communities*: Online communities can be classified into three subsets based on this criterion. (i) Those based on physical communities, such as online communities that serve the people of specific towns or counties; (ii) those somewhat based on physical communities, such as the members of a professional society who meet infrequently in a physical manner at conventions, and conferences; and (iii) those not related to any physical community (1998 pp.85-86).

(4) *Boundedness*: This criterion is based on the proportion of social relationships exclusively between the community members, and social relationships with people from outside the community. According to this criterion, in a tightly bounded community most of the social relationships take place among the members of the community as opposed to a loosely bounded community, in which most of the social relationships take place between the members and the outsiders (1998 pp.86).

An alternative classification based on the purpose of the community and the types of transactions required to realize that purpose was suggested by Stanoevska-Slabeva and Schmid (2001):

(1) *Discussion communities*: These online communities are formed in order to facilitate information exchange on a specific topic (2001 pp.5). Discussion communities can be further divided into four sub-classes (2001 pp.5-6):

(a) *Person-to-person discussion communities* bring people together to build direct relationships with other members.

(b) *Topic-oriented discussion communities* are formed to let members discuss openly about a specific topic.

(c) *Communities of practice* emerge from within a specific organization in order to facilitate know-how exchange.

(d) *Indirect discussion communities* provide more indirect discussion among members, such as book or movie review sites, (e.g., amazon.com, imdb.com.)

(2) *Task-and-goal-oriented communities*: These are online communities which are formed in order to achieve a common goal of the members (2001 pp.5). They can be grouped into three within themselves (2001 pp.6-8):

(a) *Transaction communities* let members get together in order to carry out economic transactions, such as auction sites.

(b) *Design communities* are formed in order to carry out a specific design and production task collaboratively.

(c) *Online learning communities* are used for facilitating collaborative online learning.

(3) *Virtual worlds*: These communities provide a virtual environment for interaction between members, such as online gaming communities (2001 pp.5).

(4) *Hybrid communities*: These communities combine several functions that fall in different online community classes. An example would be an online auction site where members buy and sell baseball memorabilia among themselves, and also discuss the recent baseball matches on a bulletin board. This would be a hybrid transaction-discussion community (2001 pp.5).

2.2. Defining Open Online Collaboration Communities

2.2.1. A Working Definition of Open Online Collaboration Communities

We now can develop a working definition of open online collaboration communities that is appropriate for this study. The definition is developed in two stages. First, open online collaboration communities are positioned within the overall body of online communities according to the classifications discussed in the literature review. Then the characteristics of open online collaboration communities are outlined in contrast to other online communities and traditional collaboration communities.

2.2.2. Positioning Open Online Collaboration Communities

Open online collaboration communities fit in the definition of transaction communities, based on Hagel and Armstrong's classification. From Stanoevska-Slabeva and Schmid's classification's standpoint, they fall in the design communities sub-class within the task-and-goal-oriented class. In fact, Stanoevska-Slabeva and Schmid

mentioned open source software development communities as an example for design communities (2001). For the purposes of this study open online collaboration communities are defined as "online communities that are formed by loosely connected groups of people, who use the Internet as a medium for carrying out collaborative projects for producing and improving a wide range of information products."

Probably the most widely known example that fits into the definition of open online collaboration communities is the open source software movement. The open source software movement is a collaborative software development model, which involves online communities of computer programmers dispersed around the world. These voluntary programmers use the Internet to collaboratively develop software (O'Reilly 1999). Only a small fraction of these programmers gain direct tangible benefits in return for their contributions. Most of the participating programmers are motivated by indirect or intangible benefits, such as reputation among peers or a credential they can add to their resumes (Raymond 2001). Despite the lack of monetary incentives, the open source software movement has produced high quality free software that can compete with leading proprietary software. An example is the Linux operating system (Torvalds 1999).

2.2.3. Characteristics of Open Online Collaboration Communities

There are several characteristics that distinguish open online collaboration communities from other online communities, and traditional collaboration communities:

Internet-aided: The most obvious characteristic of these communities is that they are Internet-aided. The members of the community may use other media or face-to-face meetings to communicate and collaborate. However, the main medium of interaction is the Internet.

High number of participants: These communities involve a higher number of participants compared to those of their traditional, face-to-face counterparts. The number of participants may vary substantially between open online collaboration communities.

Spatially (geographically) dispersed participants: A certain portion of the participants may have face-to-face interactions, however the overall community is spatially dispersed.

High variation between expertise levels of participants: The expertise levels of participants within a community may differ substantially.

Non-compensated participants: Participants are almost never directly compensated. However, in many communities, the majority of the participants have paying jobs related to the topic of the community (Bezroukov 1999, Markus, Manville and Agres 2000, Raymond 2001).

Very low barriers to entry and contribution: Most of these communities accept contributions from anyone interested in participating in community activities. People can join the community and submit their contributions quickly and easily.

Digital end products: The end products produced by the members of the community are digital, and thus can be stored on digital media and can be dispersed via the Internet.

Self-contained end products: The end products are self-contained entities that can be used outside of the context of the community, such as a computer program or a report. This characteristic distinguishes open online collaboration communities from several other kinds of online communities, such as discussion groups, chat groups or online game

groups. The "products" of these communities, such as discussion threads, chat sessions, and game sessions are useful only within the context of the community.

Open and free end products: The end products are "open" in the sense that their sources are accessible; in certain cases to the point that they can be altered by other participants and outsiders. They are generally free to use, at least for specific uses, such as educational and non-profit applications. Project and product-specific licenses determine the conditions and limits for end use and alterations.

Non-final end products: The end products are almost never totally final, since they can be altered, improved, extended, and integrated with other products by other parties in the future.

2.3. Theoretical Approaches to the Study of Online Communities

The literature on online communities includes several theoretical perspectives. Most of the attention seems to be focused on the motivational elements that drive people to participate in online communities. Many authors have tried to explain the phenomena of voluntary participation in online community-related activities as opposed to conventional economic activity, where participation is compensated by tangible benefits. This section summarizes those different theoretical approaches.

2.3.1. Gift Economies

Several authors suggested studying online communities through the concept of "gift economies" (Barbrook 1998, Ghosh 1998, Kollock 1999, Bays and Mowbray 2001). Raymond (2001) argued that open source software development communities are gift economies. Gift economies are based on "gift exchange" as opposed to "commodity

exchange." (Gregory 1982, Bell 1991, Carrier 1991) Commodity exchange takes place as an instantaneous exchange of products or services of equivalent value (Bourdieu 1997). In modern economies, this generally occurs in the form of transferring products or services in return for money. The parties that are involved in a commodity exchange do not necessarily have a previous or future relationship other than the specific transaction that takes place. On the other hand, gift exchange takes place between parties who have an existing relationship, or are aiming to build an ongoing relationship (Bell 1991, Carrier 1991). Furthermore, a gift exchange is not instantaneous, in the sense that the gift is not necessarily reciprocated by the giving of a "counter-gift" right away (Bourdieu 1997). However, the giving of a gift generally implies an unstated expectation of a reciprocation at an indefinite time on the part of the giver (Carrier 1991).

Some authors argued that an inherent property of a gift is that it is tied to the giver in an inalienable way, while "commodity" products or services exist and have a fixed value for the buyer irrespective of who the seller was (Mauss 1990, Carrier 1991). As an example for inalienability, a watch presented by someone to his/her spouse as an anniversary gift becomes "the watch which is a gift from my spouse", instead of just "a watch", and thus would have a value beyond the value of an ordinary watch. However, others argued that alienation is not a fundamental difference between gifts and commodities (Bell 1991). Bell defined barter exchange as a form of gift exchange, and argues that alienation is a distinguishing factor between "ceremonial gifts" and "bartered gifts," rather than a distinguishing factor between gifts and commodities. From this perspective, the watch in the previous example would still be linked to the giver in an inalienable way, since it is a "ceremonial gift." Bays and Mowbray (2001) drew parallels

between online communities and the example of a cookie barter between women, where each woman bakes a different type of cookie and trades them with others so that each woman has a variety of cookies. In this example the maker of each cookie would not be an essential characteristic of the cookies, thus they would be "impersonal"; however, they are still gifts in the sense that they are not commodities that can be bought by anyone, but instead exchanged between individuals who have on-going relationships. The argument about the possibility of "impersonal gifts" is important for using gift exchange as a theoretical framework for online communities, since the "products" or "services" exchanged via online communities are generally of impersonal nature (Kollock 1999). Including "inalienability" as an essential aspect of any gift would restrict the applicability of the gift economies concept to online communities.

The impersonal characteristic is not the only intricacy encountered while applying the gift concept to online communities. Most online communities are platforms for the exchange of digital goods, e.g. textual information or information products such as software, digital sounds and pictures. Digital goods can be reproduced rapidly in infinite numbers without any loss in quality and with very low costs. In that sense, when a "digital gift" is given, it can be given to a group of people instead of a single individual, with no or a very small additional cost (Barbrook 1998, Ghosh 1998, Kollock 1999). This sets "digital gifts" apart from "physical gifts." Ghosh (1995) called this fact the "infinity of information."

A digital gift can be given to a predetermined group of people, e.g. members of a membership-based online community which is closed to outsiders, as well as an indefinite number of people, by placing the gift on a publicly open website. Considering

that a digital gift can be given to an indefinite number of people, most of whom are unknown to the giver, the issue of reciprocity poses yet another intricacy in defining online communities as gift economies. If the takers of the gift are unknown to the giver, they would be under no obligation to reciprocate the gift, and this would discourage the giver from giving the gift in the first place. Kollock (1999) suggested the concept of "generalized exchange" to overcome this problem. In a system of generalized exchange, a gift or a favor is not necessarily reciprocated by the beneficiary, but by someone else within the group that takes part in the generalized exchange. When people help a complete stranger by giving directions or telling the time, they do not expect to get a similar favor in return from *the exact same person* they help; however, they expect to get similar help from some other person, should they need it.

When "infinity of information" comes together with "generalized exchange," the giver is better off by giving away more copies of the "gift" rather than fewer, because the real cost associated with the digital product is the cost of producing the master copy, not copying it. Once the product is produced, giving away many copies of it would not add to the burden of the giver. On the other hand, generalized exchange would increase the likelihood of reciprocation, since people would give away more copies with the expectation of impersonal reciprocation from others (Ghosh 1998).

An important implication of the concept of gift economies applied to online communities is that a larger community would motivate contributors to a greater extent, since the probability of generalized reciprocation increases as the number of contributors in the community increases. This is due to the fact that digital products are consumed in a non-rival manner, which brings us to the concept of "public goods."

2.3.2. Public Goods

The concept of public goods is another theoretical framework suggested for explaining phenomena related to online communities (Kollock 1999, Millen 2000, Wasko and Teigland 2002). Several authors used the concept of public goods as a framework for studying open source software development communities (Hawkins 2001, Bessen 2002). Public goods (or collective goods, as they are sometimes called) have two aspects that distinguish them from private goods. First, public goods are "non-excludable"; that is, it would be too hard or too costly, if not impossible, to exclude the non-payers from benefiting from a public good. Second, the consumption of public goods is on "non-rival" basis; that is, the consumption of a public good by an individual does not hinder other individuals' consumption of the same good. Most public goods show these two characteristics to different extents, rather than in an absolute manner. "Pure public goods," on the other hand, are totally non-excludable and non-rival (Cowen 1993). Widely used examples of public goods are firework shows, lighthouses, public libraries, parks, and traffic lights.

The provision of public goods is sometimes problematic. Since it is infeasible to exclude non-payers from benefiting from public goods, it is also not feasible to charge for their use. This brings about the problem of lack of interest in producing and distributing public goods. Certain public goods, such as public education, national defense, and highways are provided by the government, and paid for through taxes. Another array of public goods is tied to private goods. These public goods are paid for through payments for the private goods they are tied to, such as public services in a shopping mall, which are paid for indirectly through private goods sold in the mall (Cowen 1993).

The basic social dilemma about public goods is that the rational thing for each individual is to “free ride”; that is, to benefit from public goods without participating in their production or without even paying for them. Nonetheless, someone must produce them or pay for them, just like private goods. Even if the members of a community know that they would benefit from the production of public goods, their rational choice would be not taking part in that production. This follows from the argument that the rational members of a group would not act in favor of their common group interests, but their own personal interests (Olson 1965). Take the example of a society with a high number of members, where dues are not compulsory, but voluntary. A rational member would choose not to pay dues, since that would not affect the overall revenue of the society substantially, but the member would be better off financially by not paying dues. What follows is that any rational member would choose not to pay dues for the same reason, and the overall revenue of the society would be adversely affected. This problem can be overcome when there is some form of coercion or incentive that would motivate members of the group to act in favor of the common group interests (Olson 1965). Another condition that would overcome this problem is the existence of altruistic motivation; however, the body of literature discussed above is mostly from the field of economics, and altruism is not generally treated as a viable motivation factor from the mainstream economics standpoint, unless it is defined with respect to the utility it would provide to the person acting upon altruism. The sociological perspective seems to be less rigid in terms of accepting altruism as a motivation factor.

In the same vein as Olson and others, Kollock (1999) pointed out two challenges for the provision of public goods. The first challenge is motivating individuals to

participate in the production of, or to pay for public goods. The second is the issue of coordinating motivated individuals in their efforts to produce public goods. Kollock outlined the possible motivation factors for participation in the production of public goods which are digital in nature. It is important for this research to examine how digital products fit the definition of public goods.

Digital products are non-rival in consumption, since they are easy and cheap to duplicate, and duplication does not reduce their quality. Especially, in the case of web-based diffusion through FTP and HTTP, the marginal cost of each download on the procurer's part is almost zero. However, digital products are not necessarily non-excludable. It is possible to restrict access to digital products, even though it is not always simple and feasible to prevent circumvention by means like illegal copying. In that sense, digital products are not pure public goods. Proprietary software or copyrighted musical recordings are examples of digital products that are not public goods. However, there is a wide variety of digital products which are public goods, such as free software, and web pages open to public access. In this sense, if a digital product is available to the public free of charge, it is a public good.

Kollock (1999) suggested four possible motivation factors for participating in the production of digital public goods:

1) Individuals may contribute to the production of digital products with the expectation that their efforts will be reciprocated in the form of contribution from other individuals in the group or community. This factor is similar to the idea of a generalized exchange within the group, as discussed above under the heading of "Gift Economies." Kollock argued that a system that identifies contributors and measures their contributions,

at least in a rough manner, would increase the effect of this factor, since individuals will feel obliged to contribute in order not to be shunned in the long run. Again, as discussed within the context of gift economies, the probability of reciprocation would increase as the audience grows larger, giving way to a higher level of motivation towards contributing. This motivation factor has a direct implication for the open source software development (OSSD) model, which was build in the first phase of this study: Participants would be attracted to contribute to communities that offer a high level of utility in terms of the products they are developing. Consequently, an open source software community becomes more attractive to participants as the level of functionality and quality of its product increases. An important component of the overall utility of a software product is the number of its users. Several authors have argued that a software product would become more attractive to users as its market share increases (Katz and Shapiro 1985, Gallaugher and Wang 1999). This is called the positive network externalities effect. The implication of this effect for the OSSD model is that a higher number of users would make the community's product more attractive for potential users.

2) Individuals may also be motivated by the expectation that their contributions will earn them recognition and reputation among the members of the group or the community. Reputation can be a motivating factor through two mechanisms: 1) ego satisfaction due to being respected by the community, 2) professional and financial opportunities that come with recognition. Programming skills proven through non-compensated work may open doors into a compensated position in the area of one's expertise. Kollock argued that the effect of this factor would be directly correlated with the visibility of contributions and the availability of some sort of a recognition

mechanism. It can be argued that the existence of opportunities for material compensation related to the voluntary work would increase the motivational effect of the reputation factor. An important condition for this motivation factor to have an effect is that programmers should be able to find adequate opportunities for contributing to a project, which would demonstrate their skills. A mature project may fail to offer enough opportunities for contribution. Raymond (2001) introduced the concept of "homesteading" an open source software project. He argued that participants would claim portions of a software project and build their reputations within and beyond the community based on the functionality and quality of the portions that they work on, or in other words, that they "own." This argument has a direct implication for the open source software development (OSSD) model: If an open source software product is in its maturity stage and most of the potential functionality is already added, the product would become less attractive for the contributors, since there would not be enough unachieved functionality to be homesteaded.

3) Another motivation factor may be the feeling of self-efficacy that comes with the perception that the individual has an effect on the community or the larger world by his/her contributions. Kollock argued that the effect of this factor would increase as the size of the community increases, since contributors will have the opportunity to affect the lives of a larger audience by their contributions. However, it can also be argued that the increasing community size would diminish the relative impact of the contributions of a given individual, since there would be more contributions from a larger contributor base. Distinguishing the contributors and the users who do not contribute as two separate audiences can make it easier to theorize about the effect of this factor. A larger user

audience given a fixed number of contributors would increase the effect of this factor, while a greater number of contributors given a fixed user audience would decrease the effect. A direct implication of this motivation factor for the OSSD model is that a larger user pool would make the community more attractive for contributing participants. Another implication of this factor is in parallel with the implications of the reputation factor discussed above. Based on this motivation factor, an open source software development community would become less attractive as its product reaches a very high level of maturity, and thus fails to offer ample opportunities for contribution.

4) Finally, Kollock argued that contributors might be motivated in a purely altruistic manner by the potential benefit to other members or the community as a whole. Here again, a larger audience may mean a higher effect on motivation, due to the increase in the cumulative benefit. It can also be argued that the existence of feedback channels, which would inform the contributor about the realization of potential benefits to others, would have a positive effect on this factor's contribution in the level of motivation. This motivation factor supported the implication that a larger user pool would increase the attractiveness of the community for contributors.

2.3.3. Social Networks

Another theoretical framework suggested for studying online communities is social network analysis (Garton, Haythornthwaite and Wellman 1997, Wellman 1997, Wellman and Gulia 1999, Jones 2000). Social network analysis is a methodology widely used for studying patterns of relationships among agents, which in many cases are people. However agents can also be other social entities such as families, companies, or states (Garton, Haythornthwaite and Wellman 1997).

Social network analysis defines a given group of people (or other agents) as a network, which is formed by the members of the group and the relationships between these members. The members of the group are represented as nodes, and the relationships as the links of the network. Social network analysis has been widely used to study the exchange of resources among the members of social groups (Wellman and Berkowitz 1988, Wasserman and Faust 1994, Scott 2000, Rethemeyer 2002). It is possible to approach information sharing from a social network analysis point of view by defining it as a resource that is shared among people (Garton, Haythornthwaite and Wellman 1997, Rethemeyer 2002).

The unit of analysis in social network analysis is a "relation." Relations have different characteristics. For instance, a relation can be directed or undirected (Garton, Haythornthwaite and Wellman 1997). Friendship is an example of undirected relations, since both agents are friends from each other's point of view. On the other hand, parenthood is a directed relation. Another characteristic that distinguishes relations is their strengths. Relations may be strong or weak. Different types of relations would have different operationalizations for defining their strengths (Garton, Haythornthwaite and Wellman 1997). For example, the strength of friendships can be operationalized in terms of the frequency and length of meetings among the friends, or the amount of self-sacrifice they claim they would make for their friend.

One or more relations connecting two agents form a "tie." A tie that involves more than one relation is a "multiplex tie." Ties also differ based on their strengths: "strong ties" and "weak ties." Strong ties are ties among agents that share many resources, and in a more frequent, intimate and dependent manner, while weak ties are

those between agents that share fewer resources, infrequently, and not in a dependent manner. While strong ties are more crucial for an agent's social existence and well being, weak ties nevertheless may also play a crucial role in an agent's social life, especially if they are many in number and used efficiently. The concept of "networking" between colleagues is an example of an effort to maintain and increase one's weak ties.

A substantial portion of the social network studies done on online communities focuses on the nature and usefulness of Internet based weak ties, and whether strong ties are possible in online relationships (Wellman and Gulia 1999, Preece 2000 pp.177-178). Another important question related to online communities, which several researchers have tried to answer, is whether online communities support or hinder physical communities (Wellman and Gulia 1999, Preece 2000 pp.182). Several authors have suggested that online relationships and online communities may hinder relationships and communities in the physical domain of everyday life (Fox 1995, Slouka 1995 pp.95-100). A strong argument made by such authors is that online relations distance people from non-crucial social interaction in the physical domain, and thus decrease the social capital within the society. Social capital is defined as "capital captured through social relations" (Lin 2001 pp.19). In that sense, social capital refers to the quantity and quality of social ties within a community or a society. According to Putnam (1995) social capital "refers to features of social organization such as networks, norms, and social trust that facilitate coordination and cooperation for mutual benefit." While some authors argue that online life reduces the amount of time people spend building and maintaining social ties in their physical life, some others suggest that online relationships and online communities may

foster trust and cooperation between those who engage in online socialization and thus help increase and improve social capital (Preece 2000 pp.22-24 and 182).

Social network analysis focuses on the relationships between individuals, and thus differs from most other social science approaches that focus on individuals (Garton, Haythornthwaite and Wellman 1997). This alternative way of looking at groups gives way to critical findings, which might not reveal themselves through other approaches. However, social network analysis provides only one part of the picture with respect to the development of open online collaboration communities. The implications provided by social network analysis do not lend themselves readily for translation into a dynamic feedback model. Thus, the implications that this theoretical approach provides about phenomena related to online communities were not as useful in conceptualizing the initial system dynamics model as those provided with the other approaches discussed in this literature review.

2.3.4. Social Informatics

Several authors approach the study of online communities from a perspective which is interchangeably called “social informatics” or “social impacts” (Turoff and Hiltz 1982, Hiltz 1986 pp.151, 165, 191, Preece 2000 pp.194-196). Social informatics research focuses on the social impacts of information systems (Preece 2000 pp.194-196). The basic argument of the social informatics approach is that the design and use of information systems have an impact on the social processes that govern the context in which those information systems operate. Furthermore, information systems, together with social processes, have an impact on social structures and relationships. Based on these premises, several authors argue that while designing an information system, the

effects on the social processes, structures and relationships should be taken into account, and the information system should be designed as a part of the social process it will be "embedded in" (Turoff 1997, Preece 2000 pp.194-196).

A certain array of research focusing on the organizational issues within the Human-Computer Interaction field have roots in the social informatics approach (Eason 1997, Grudin and Markus 1997, Smith and Conway 1997). The social informatics approach is also among the theoretical foundations upon which computer supported cooperative work, and computer mediated communication fields are based (Applegate, Ellis, Holsapple, Radermacher and Whinston 1991, Turoff 1991, Eason 1997, Grudin and Markus 1997, Olson and Olson 1997, Smith and Conway 1997).

Preece explained the implications of the social informatics perspective through examples of electronic journals (2000 pp.194-195). The first example, taken from Kling (1999), is an electronic journal whose submission process is designed to let authors and readers discuss online about submitted articles, before the articles are finalized and go into the peer review stage. The submission process of the electronic journal discussed as the counter example is designed more or less like a traditional peer reviewed journal, which operated through an editorial board, without the opportunity of wide discussion. Preece argued that the social process design and the related software (technological) design of the first journal would generate more community involvement (2000 pp.195).

This argument brings about the general implication that the design of the social processes and the software used for the operation of an online community may have considerable impact on participation. The first example set forth by Preece is arguably more "democratic," or has a "flatter" hierarchy structure compared to the second

example. Thus it can be argued as an implication that a more democratic or a hierarchically flatter socio-technical design may increase participation, by decreasing the barriers to contribution.

In reality, both journals in Preece's examples use a peer review process as the final stage, where the decision about whether a given article should be published, and in what final form is made. However, the discussion stage in the case of the first journal provides an opportunity to incorporate suggestions and other input from a wider body of participants, which definitely would yield a different "final submission," if not a better one. A "final submission" shaped by a wide scope of contribution may be expected to have a better chance of being accepted in the peer review process with a lower number of revision suggestions for two reasons: First, it would probably have a higher quality since it would incorporate suggestions and corrections from a wider audience. Also, since it would reflect the consensus of a much wider portion of the community in question, it may have an impact on the decision of the reviewers through the power of being a socially negotiated and accepted "reality." Clearly, this second effect, if present, is not a necessarily positive one, since it may impose socially accepted errors, or mistakes on the reviewers' part.

The above outlined implications can also be drawn from the open source software development literature. Raymond argued that the participation of a wider audience in an open source software development project, especially in the testing and debugging phases, has a positive effect on the overall quality of the software being developed. Raymond argued that "[g]iven a large enough beta-tester and co-developer base, almost

every problem will be characterized quickly and the fix obvious to someone" (2001 pp.30).

With respect to the relationship between barriers to contribution and participation, Raymond implied that as barriers to contribution decrease, participation would increase. Raymond argued that there is an inverse relationship between "the number of hoops" a user needs to go through in order to contribute to a project and the number of contributors. Raymond argued that the barriers to contribution may be "political" as well as "mechanical" (Raymond 2001 pp.109). The "mechanical" component is mostly related to the software, and partially to the technical dimension of the social processes that govern the community, while the "political" component is related to the policy dimension of the social processes, or in other words, the set of rules and policies with which contributions are handled. Here, Raymond compared Linux and various BSD projects from an organizational point of view. According to Raymond, the mechanical and political components of barriers to contribution may explain why an "amorphous" open source software development community such as the Linux community attracted far more contributors than tightly organized and controlled BSD communities (Raymond 2001 pp.109).

Fogel and Barr set forth arguments along the same lines (2001 pp.10-11). They argued that the convenience provided by an efficient system that makes contribution easy is not a mere luxury, but a necessity for projects that run on volunteer efforts. The level of convenience for making contributions may be the ultimate determinant of the number of contributors, and the amount of their contributions to a project (Fogel and Bar 2001 pp.11).

Fogel and Bar set forth another argument that is important with respect to the concept of open online collaboration communities in general. They positioned the problems posed by the physical and temporal separation between open source software developers within the context of Computer Supported Collaborative [Cooperative] Work[1] and argued that problem assessments and suggested solutions to these problems should apply to other open source-style content development (Fogel and Bar 2001 pp.10). This argument suggests that the implications of a social informatics approach that hold for open source software development projects should also hold for other open online collaboration projects. Clearly, Fogel and Bar viewed open source software development and other digital content development efforts as examples of the same phenomenon, which we define as open online collaboration in this study.

2.4. System Dynamics Approaches to Software Project Management

There is a substantial body of research that focuses on applying a system dynamics modeling approach to software development-related problems (Abdel-Hamid and Madnick 1991, Barlas and Bayraktutar 1992, Madachy 1994, Rodrigues and Williams 1997, Bell and Jenkins 1998, Barros, Werner and Travassos 2000, Williams 2001, Rai and Mahanty 2002). Most prominent of these to date is a line of studies carried out by Abdel-Hamid and other researchers who joined him during different stages of the overall research project (Abdel-Hamid and Madnick 1983, Abdel-Hamid and Morecroft 1983, Abdel-Hamid 1984, Abdel-Hamid 1989, Abdel-Hamid and Madnick 1989, Abdel-Hamid and Madnick 1991). The model Abdel-Hamid and Madnick (1991) discussed

[1] The established name for that field is Computer Supported Cooperative Work. However, Fogel and Bar preferred to use the term Computer Supported Collaborative Work in their book.

throughout their book Software Project Dynamics is a good summary of the overall research that was carried out over several years.

Abdel-Hamid's model was based on software engineering literature, and 27 interviews held in 5 software development organizations to supplement the literature wherever needed. The model and its managerial implications were tested through a series of case studies.

Abdel-Hamid divided the software project model into four sub-models, or "sectors" as they are called in system dynamics literature: human resource management, software production, project control, and project planning (Abdel-Hamid and Madnick 1991 pp.13). The human resource management sector addresses the aspects related to the hiring and turnover of the workforce, as well as the change in the experience level of the workforce. A critical issue that this sector addresses is the rate at which an inexperienced workforce is "assimilated," or becomes experienced through training (Abdel-Hamid and Madnick 1991 pp.63-68). The software production sector focuses on manpower allocation, quality assurance and rework, and system testing, as well as the actual software development itself (Abdel-Hamid and Madnick 1991 pp.69). This is the sector that provided most of the implication for the open source software development model built for the first phase of this dissertation study. Parameters such as productivity, error generation rate, error detection rate, error fixing rate are based on assumptions derived from this sector of Abdel-Hamid's model. The project control sector represents managerial functions related to measurement, evaluation and communication in an effort to improve project performance (Abdel-Hamid and Madnick 1991 pp.117). The project

planning sector is where decisions about key determinants such as scheduled completion date and workforce level are made (Abdel-Hamid and Madnick 1991 pp.129).

Abdel-Hamid used the software project model to test various arguments that have dominated the field of software project management, as well as alternative policy options that were hypothesized to improve project performance. One such example is Brooks' Law. According to Brooks, adding more people to the group working on a late project would make it finish even later. In other words, the net impact of assigning more people to a late project is negative (Brooks 1995). Abdel-Hamid argued that the behavior of his software project model indicates that this does not hold at least for a certain range of projects. He argued, based on findings from his simulations, that although assigning more people to a late project always causes it to become more costly, it does not necessarily push the completion date even later. He argued that Brooks' Law would hold for cases where the new workforce acquisition is made extremely close to the projected completion date (Abdel-Hamid 1989).

Abdel-Hamid's work remains the most comprehensive look at software project management from a system dynamics perspective, and has been heavily cited throughout software project management literature. The model and the overall study provide insights into all aspects of software development phenomena in terms of policy implications. For the purposes of this study however, the most useful implications were not the policy implications, but rather the method of incorporating the mechanics and parameter of software development into the model. This can be attributed to the fact that Abdel-Hamid's study focuses on "proprietary" software projects, while the software development model built within the scope of study looks at a generic open source

software project.[2] Some key differences between proprietary and open source software projects force the open source software development model to differ substantially from a model of proprietary software projects such as Abdel-Hamid's. Arguably the most important difference is that proprietary software projects are run by a paid workforce, while open source software projects are run by volunteers. It is possible to add new people to the workforce in a proprietary software project at any given time, as long as the budget provides the financial means. Open source software projects are not as flexible in terms of recruiting a new qualified workforce, since they are run through motivation factors other than direct financial compensation. Another important difference is that proprietary software projects need to follow a more or less preset schedule with a declared completion date. Open source software projects are more flexible in terms of schedule and completion dates, as long as they do not fall too far behind their competition in terms of delivering the product in a timely fashion.

Another notable application of system dynamics to software development issues is a line of research by Madachy (Madachy 1994, Madachy 1996, Madachy 2000, Madachy 2002, Madachy and Boehm 2003). Madachy's study and his "inspection-based" process model differ from Abdel-Hamid's in certain aspects. Madachy focused on inspection and rework related activities. In order to simplify the model he excluded productivity determinants such as schedule pressure and manpower mix. For example, instead of using two pools for workforce -- one experienced and another inexperienced, Madachy used only one aggregate workforce pool. On the other hand Madachy's model is much more detailed with respect to inspection and rework related activities than Abdel-

[2] Abdel-Hamid's work predates the mass diffusion of the Internet, and consequently the conception and application of open source software development as we know it today.

Hamid's model. For example, unlike Abdel-Hamid's model, in Madachy's model quality assurance activities are not postponed or accelerated when schedule pressure sets in (Madachy 1996).

Madachy advanced Abdel-Hamid's work in certain aspects by applying a contemporary look at the issue. There is about a decade between the span of Abdel-Hamid's and Madachy's studies, and a decade is a considerably long time when it comes to evolving practices like software project management. Having said that, Madachy's work did not provide further implications for the model built with this study beyond those provided by Abdel-Hamid's study for the same reasons discussed with respect to Abdel-Hamid's work. Just like Abdel-Hamid's, Madachy's model essentially represents proprietary software project management.[3]

2.5. System Dynamics Approaches to Instructional Material Development

Application of system dynamics to the domain of instructional material development has been piecemeal at best, and in the small number of cases where the methodology is applied to related issues, instructional material development activities per se is not the main focus of the study. One example of using system dynamics for studying instructional material development was carried out by the "Grimstad Group." Grimstad Group is an international group of researchers who have studied the application of contemporary technology to instructional design. The objective of the Grimstad Group's

[3] There have been other studies focusing on applying system dynamics to software development phenomena (Barlas and Bayraktutar 1992; Rodrigues and Williams 1997; Barros, Werner and Travassos 2000; Donzelli and Iazeolla 2001; Kahen, Lehman, Ramil and Wernick 2001; Martin and Raffo 2001; Pfahl, Klemm and Ruhe 2001; Ruiz, Ramos and Toro 2001; Stallinger and Gruenbacher 2001; Williams 2001; Rai and Mahanty 2002 .) However, these are not discussed in this literature review in depth, since they add little to Abdel-Hamid's and Madachy's studies. These studies did not provide any additional implications for the purposes of building the open source software development model.

study was “to extend and validate system dynamics technologies for use in managing the complexities and risks of large-scale courseware development projects” (Spector 1995). While the main theme of the study was to introduce system dynamics and systems thinking tools into instructional design, the researchers also worked on a system dynamics model of the process of instructional planning and production. Though initial steps of model development were reported (Spector 1995), the literature does not indicate that the model was eventually completed. As far as the initial report goes, the researchers aimed to build a model that would be used to test policies to improve courseware development projects (Spector 1995).

A system dynamics model of the growth of the community of teachers and researchers applying system dynamics concepts to K through 12 education is still in the development phase. The model was initiated by a group of teachers and researchers working within the said community, through a process facilitated by Dr. James Lyneis. At the time this dissertation was written the model was still in development stage, and thus had not been published.

Another study, which is rather tangential to the topic of this dissertation, focused on the growth of the field of system dynamics (Andersen, Radzicki, Spencer and Trees 1997). This model has not been published in detail. However, one very brief conference paper about it does exist (Andersen, Radzicki, Spencer and Trees 1997). The main focus of the model is the process through which people are attracted to work in the field of system dynamics. The model suggests that as more system dynamics based projects are completed and published more people will become aware of system dynamics, and a certain portion of those will chose to join the field and carry out more system dynamics-

based projects. Word-of-mouth through newcomers will also increase the number of people aware of system dynamics. The conception of new system dynamics projects is not only contingent upon the existence of many people working on system dynamics (namely the supply side), but also the quality of the existing projects, since the existing quality would determine the level of demand for further system dynamics projects. Growing too fast might bring about a problem of decreasing quality, since most of the people working in the field would be newcomers. One way to overcome this, according to the model, is to provide mentoring for newcomers by experienced system dynamicsists.

The "growth of the field" model was not developed further by the original authors, however the System Dynamics Society recently started an initiative to update and extend the model with the participation of the executive director of the society and several volunteer system dynamicists. As the updated version of the model is still in development phase, no publications about it have been made so far.

2.6. Problem Statement and Dynamic Hypothesis

This section integrates the implications of the literature review in order to introduce the problem statement and develop a dynamic hypothesis that will be the basis for the open source software development model.

The level of success open online collaboration communities achieve varies substantially. While some communities reach a wide audience and achieve considerable success, others fail to reach critical threshold in terms of number of contributors, end users, and product functionality (Bezroukov 1999, Preece 2000 pp.25-27, Raymond 2001, Sandred 2001 pp.81-92). Figure 2.1 and Figure 2.2 portray generic behaviors of

successful and unsuccessful open online collaboration communities with respect to product functionality, number of contributors, and users.

As seen in Figure 2.1 this research posits that the quantity of products developed by a successful open online collaboration community keeps growing until it reaches a point where it attract a sustainable audience of contributors and end users. After the threshold is passed, a project may keep growing exponentially or linearly, or it may reach a more or less fixed size. Generally, convergent products, such as software, tend to reach a fixed size after a certain period, while the size of divergent products keep growing.

Figure 2.1 shows that the number of contributors and number of end users of successful communities either continue to grow, exhibit a logarithmic growth and reach an equilibrium, or overshoot and then decline to a sustainable equilibrium.

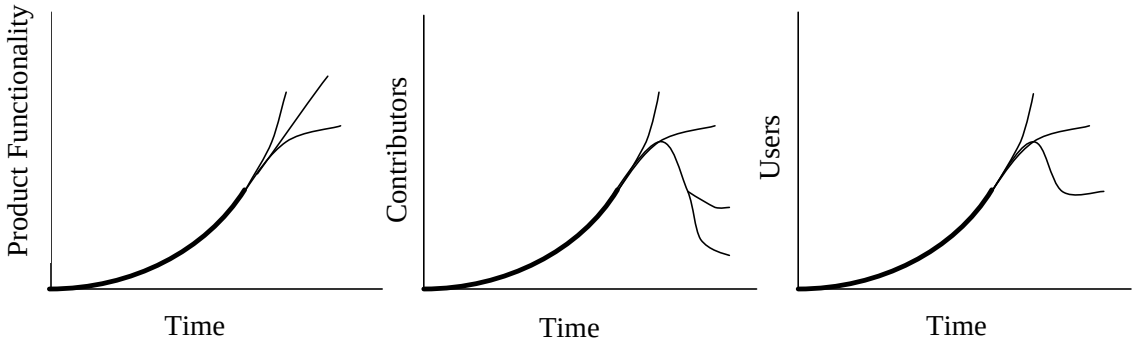


Figure 2.1. Generic Behavior of Successful Open Online Collaboration Communities

On the other hand, Figure 2.2 shows that unsuccessful communities can never reach the level of product functionality or number of contributors needed to reach a wide audience and sustain the community. Product functionality grows too slowly and never reaches a level where it could attract more active contributors and end users.

Consequently, the contributor and end user audiences either vanish, or stay at extremely low figures, turning the community into a “cult,” which cannot grow.

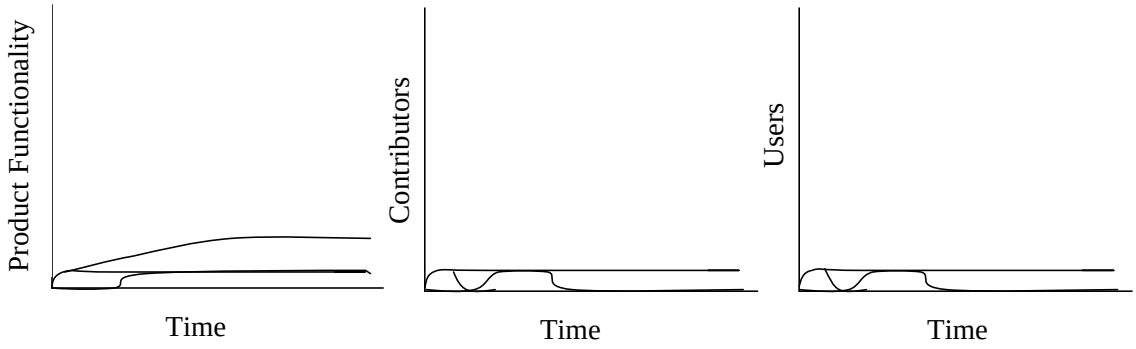


Figure 2.2. Generic Behavior of Unsuccessful Open Online Collaboration Communities

This study hypothesizes that success indicators of open online collaboration communities with respect to product functionality, product quality, number of contributors, and number of end users are determined by a complex system of interactions between determinant factors such as participation, production, barriers to entry and contribution, motivation, level of collaboration, and technology. Consequently, this study addresses the problem of identifying the underlying dynamic feedback structure among these elements and analyzing a set of policy option to improve the overall performances of open online collaboration communities. The dynamic hypothesis discussed below was the first step in identifying that dynamic feedback structure. The dynamic hypothesis was used as a candidate to replicate and explain the phenomena observed in open online collaboration communities. The open source software development community model introduced in Chapter 4 was based on this dynamic hypothesis.

The two reinforcing feedback loops shown in Figure 2.3 are the drivers behind the growth of an open online collaboration community. Here, developers participate in production, and add functionality and quality to the product. Product functionality and product quality positively affect perceived success in achieving functionality and quality respectively, which in turn affect attractiveness of the product positively. Finally, attractiveness of the product has a positive effect on the number of developers, since it attracts more developers into the community. This loop reflects the implication that an open source software community becomes more attractive to participants as the level of functionality and quality of its product increases, as discussed in the literature review within the context of gift economies and public goods concepts.

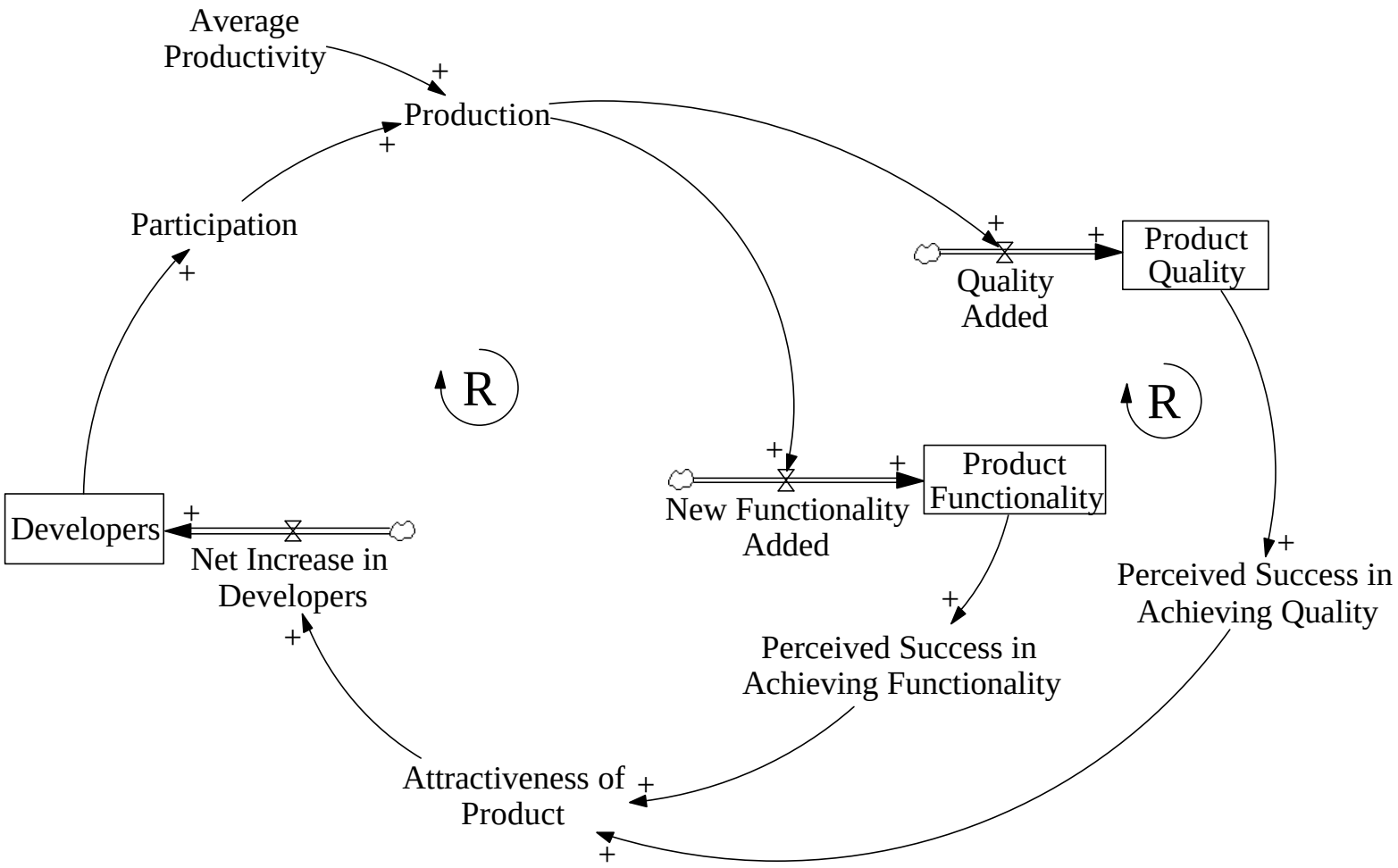


Figure 2.3. Two Reinforcing Feedback Loops Driving the Growth of an Open Online Collaboration Community.

There are two additional positive loops that reinforce the effect of the main driving loops, as shown in Figure 2.4. Attractiveness of the product has a positive effect on the motivation of developers to participate, which in turn positively affects the number of hours each developer spends on the project in a given time period; or in other words, average participation. Average participation has a positive effect on total participation, since a higher level of average participation would mean a higher level of total participation given the same number of developers. It is important to understand that while these four reinforcing loops have the potential of driving the growth of the community, they also have the undesirable potential of shrinking the community in a self-reinforcing manner, if the related variables show a decreasing behavior.

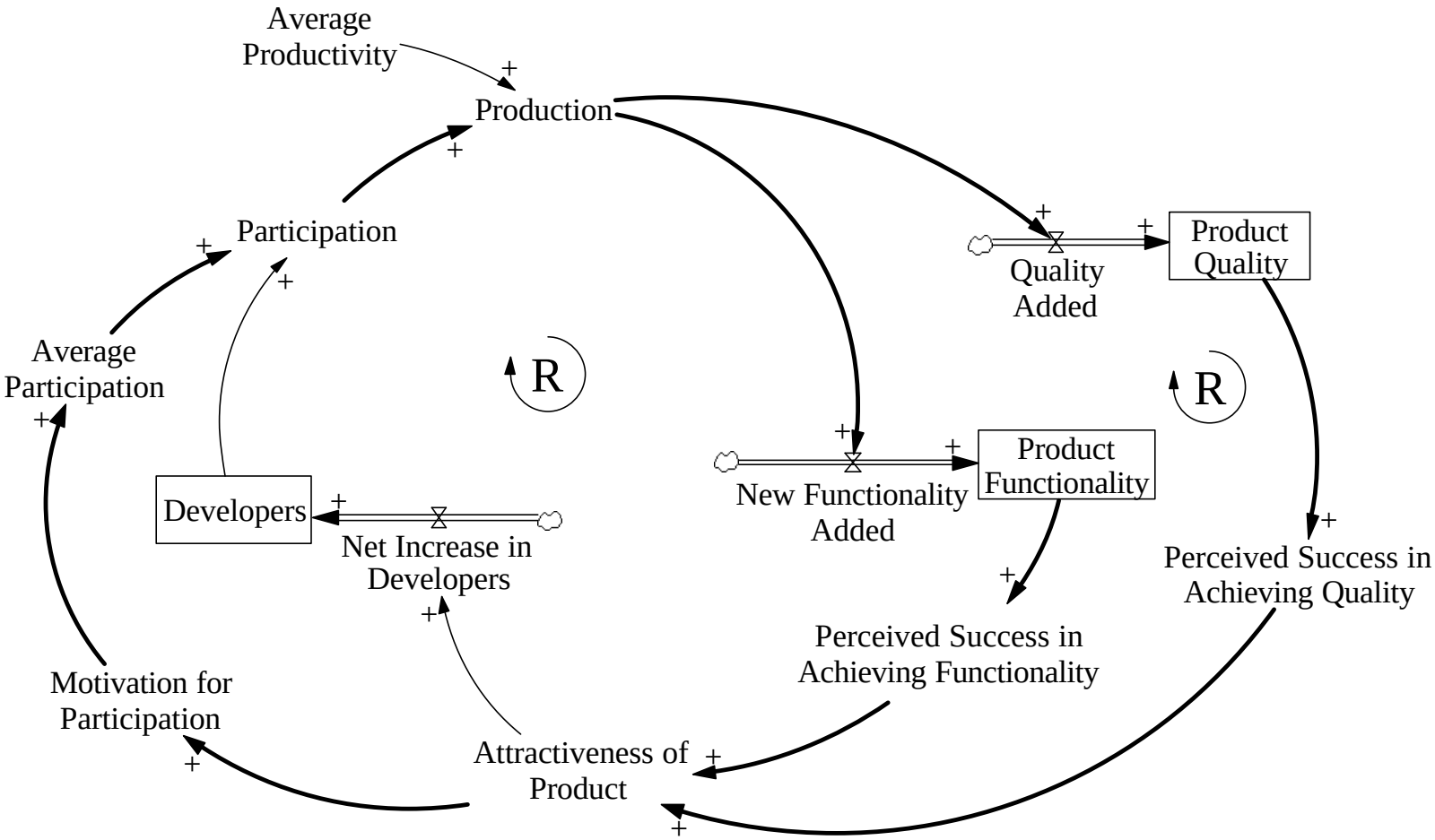


Figure 2.4. Two Reinforcing Feedback Loops, which work through Motivation for Participation.

A hypothesis of this research is that while the above discussed reinforcing loops drive the community toward growth, two important balancing loops restrict that growth, as portrayed in Figure 2.5. Production adds to cumulative production, which represents the accumulation of production efforts over time. As the cumulative production increases, the developers expect more from the product in terms of both functionality and quality. Thus, cumulative production has a positive effect on expected functionality and quality, which in turn have a negative effect on perceived success in achieving functionality and quality respectively. The two paths running from perceived success in achieving functionality and quality to production complete the two balancing loops. These two loops have the potential of restricting, and even reversing the reinforcing effects of the four positive loops discussed above.

Figure 2.5 also shows a reinforcing loop that works through the size of the end user audience. The attractiveness of the product has a positive effect on the number of end users, which in turn has a positive effect on the attractiveness of the product. This loop is based on the implications of the positive network externalities concept, as discussed in the literature review.

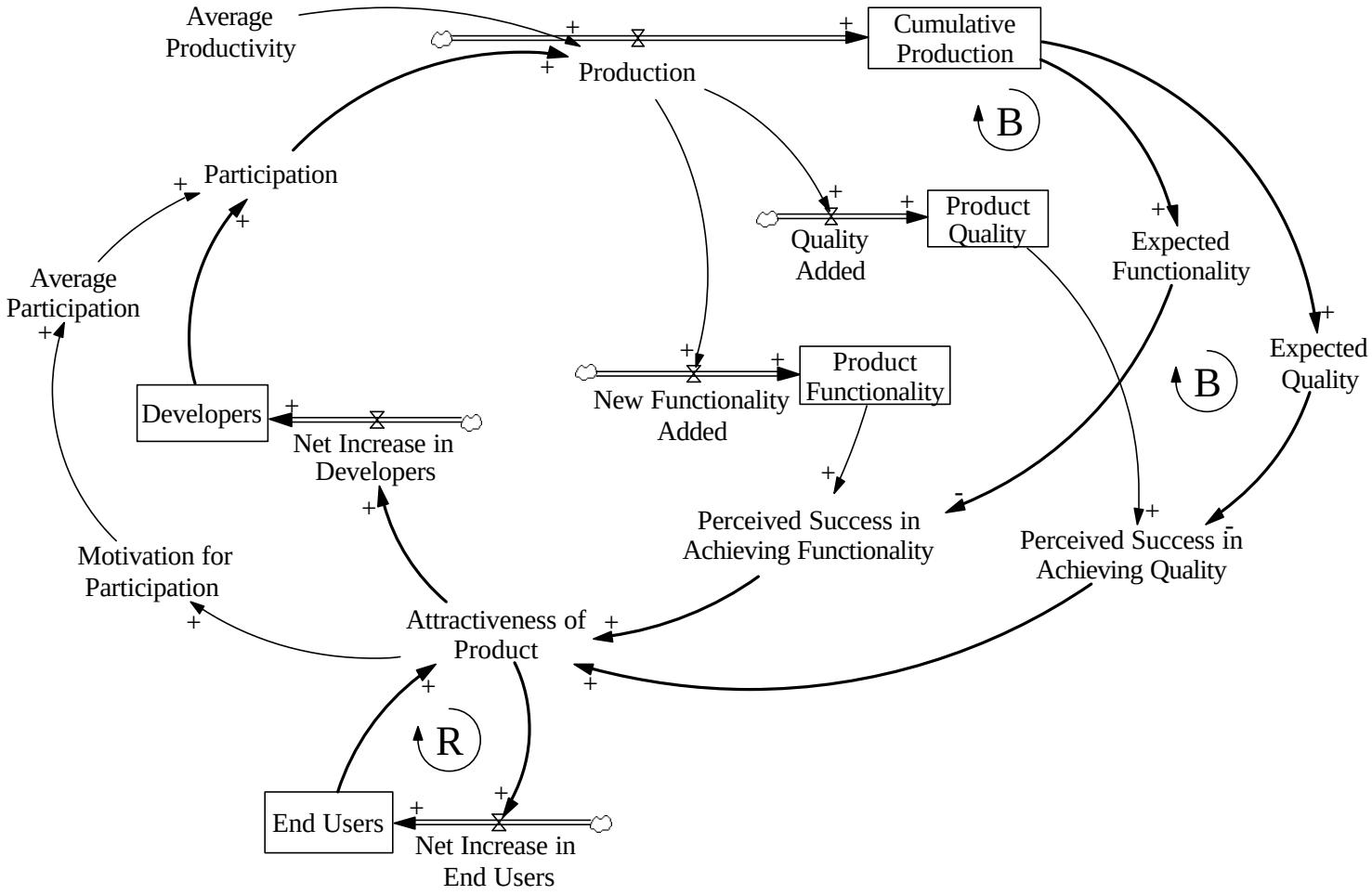


Figure 2.5. Two Balancing Feedback Loops which Restrict the Growth of the Community and the Reinforcing Loop which Works through End Users.

Many open online collaboration communities have mechanisms for checking and approving the proposed contributions from developers, in order to maintain a desirable quality level for the products (Browne 1998, O'Reilly 1999, Dempsey, Weiss, Jones and Greenberg 2002). Figure 2.6 shows the hypothesized feedback structure under the condition of inclusion of such a mechanism. Here production adds to the backlog of contributed items to be checked, which implies a need for quality checking activity. The need for quality checking would cause pressure on the system after a certain point and decrease the quality of the quality checking activities, thus, having a negative effect on it. Quality checking activities have a positive effect on product quality, which in turn has a positive effect on the perceived success in achieving quality. Perceived success in achieving quality affects quality standard for contributions positively. Quality standard for contribution affects barriers to contribution positively, as well. Barriers to entry have a positive effect on rejections, and negative effect on acceptances. Both rejections and acceptances subtract from the backlog.

The structure in Figure 2.6 is based on three additional feedback loops, two of which are balancing, and one reinforcing. An increase in the attractiveness of the product will bring more developers, and consequently increase participation, and thus production. More production generates more need for quality checking, which decreases the quality of quality checking and consequently the quality of the product. Decreased product quality means decreased perceived success in achieving quality, and therefore a decrease in the attractiveness of the product.

The other balancing loop in this structure implies that acceptance decreases the probability of future acceptances. Each acceptance subtracts from the backlog of items to

be checked, and therefore decreases the need for quality checking. This increases the quality of quality checking, since it removes pressure from the system, and consequently increases product quality and the perception of quality achievement. Increased perception of quality achievement causes an increase in quality standards for contributions, and thus increases the barriers to contribution, which decreases the likelihood of acceptances. Through the same mechanism, rejections increase the probability of further rejections, since they increase the quality of the product, and consequently the barriers to contribution by removing pressure from the system. That is the reinforcing loop in this structure.

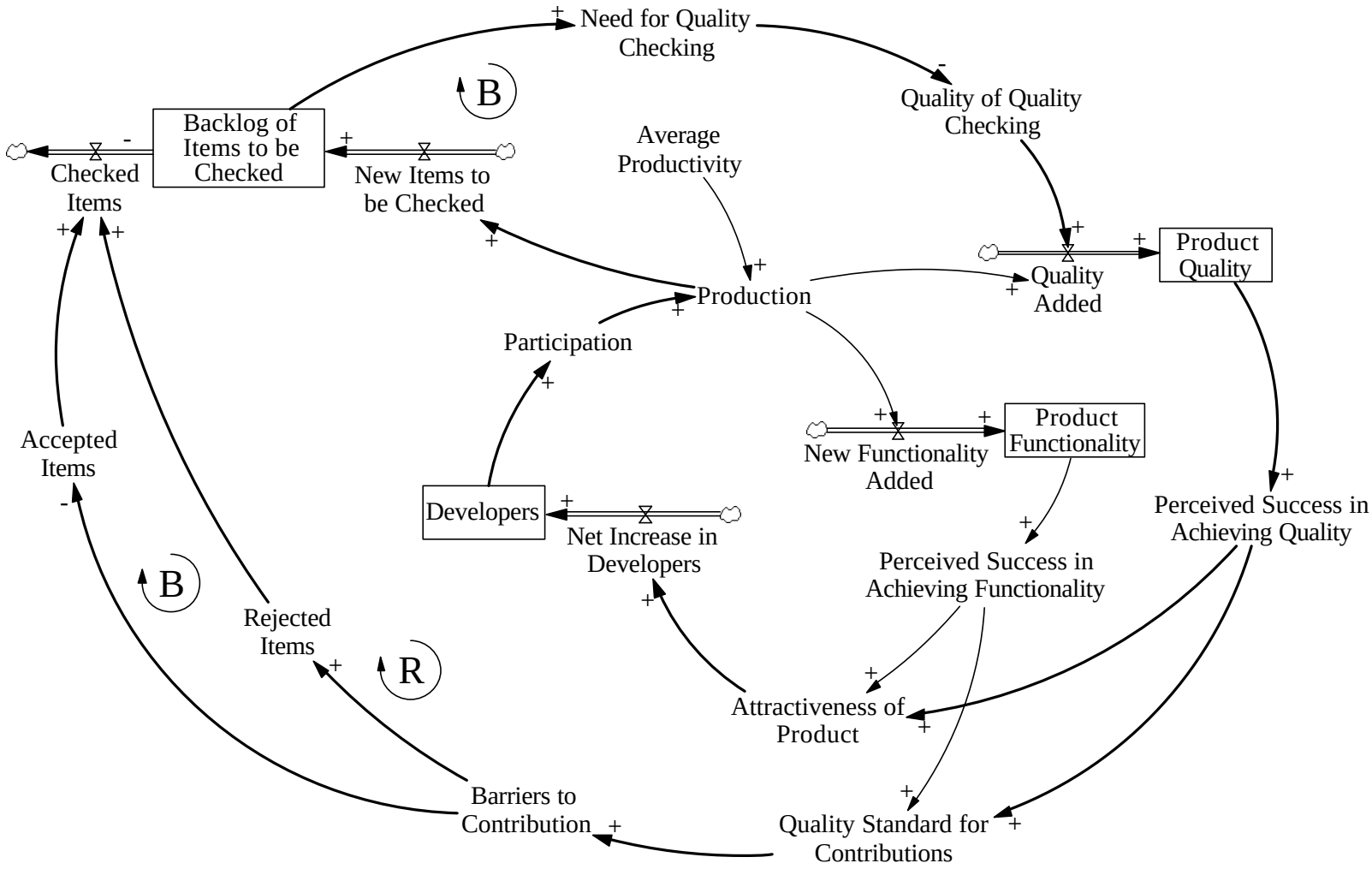


Figure 2.6. Feedback Loops Related to Product Quality Checking Mechanism.

The two feedback loops shown in Figure 2.7 have balancing effects on the product quality checking mechanism. Backlog of items to be checked generates need for quality checking, which in turn causes leading developers to spend more time on quality checking. More quality checking increases both acceptances and rejections, since more items are checked, and increased acceptance and rejection rates subtract relatively more from the backlog.

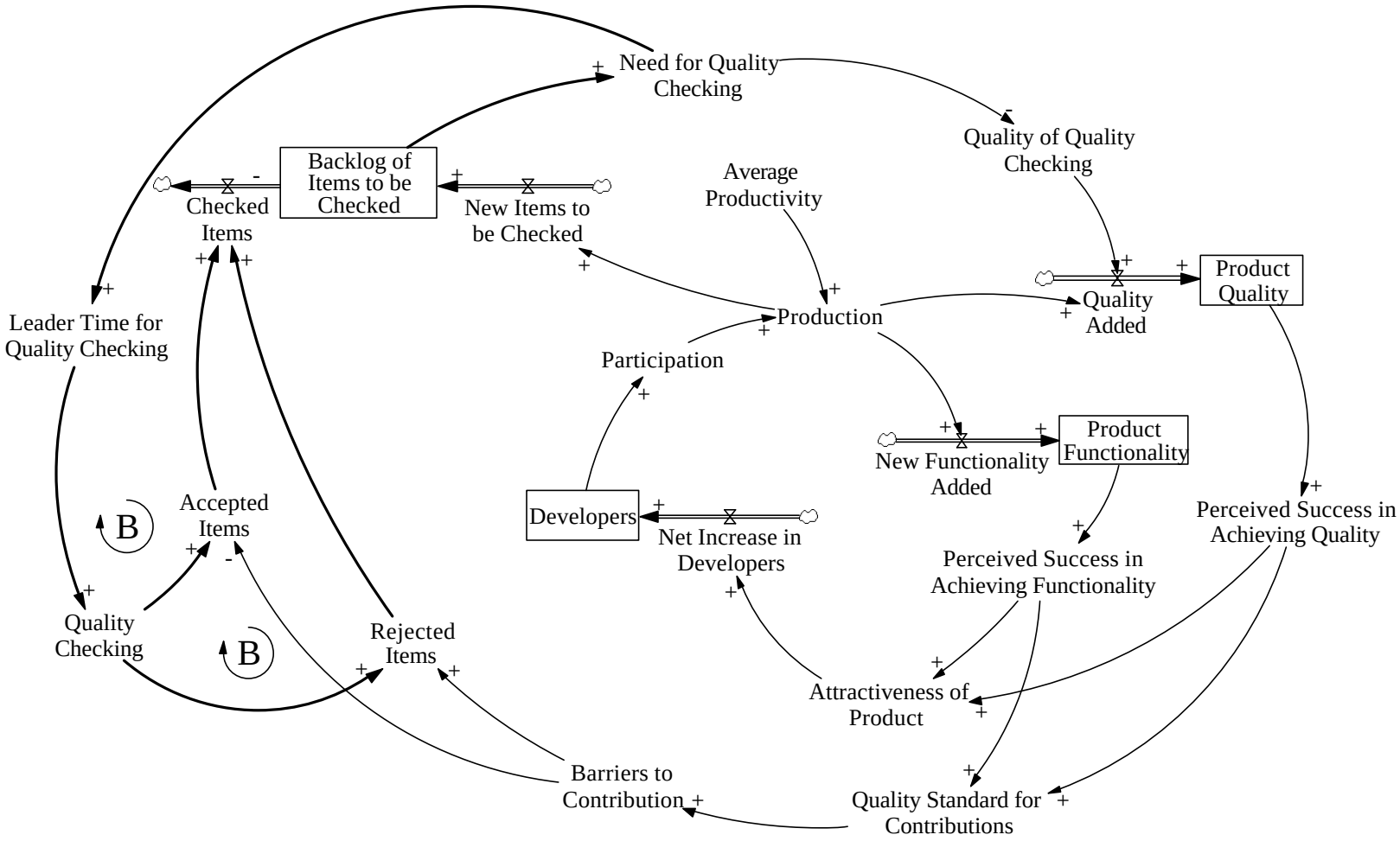


Figure 2.7. Two Feedback Loops Balancing the Product Quality Checking Mechanism.

In addition to its effect through the product quality checking mechanism, barriers to contribution have a balancing effect on the overall framework through motivation for participation. Figure 2.8 shows the two balancing loops that are driven by the negative effect of barriers to contribution on motivation for participation. As discussed in the literature review within the context of social informatics, motivation for participation would decrease as barriers to contribution increase. (This follows from the arguments made by Raymond (2001) and Fogel and Bar (2001). See section 2.3.4 for a detailed discussion.)

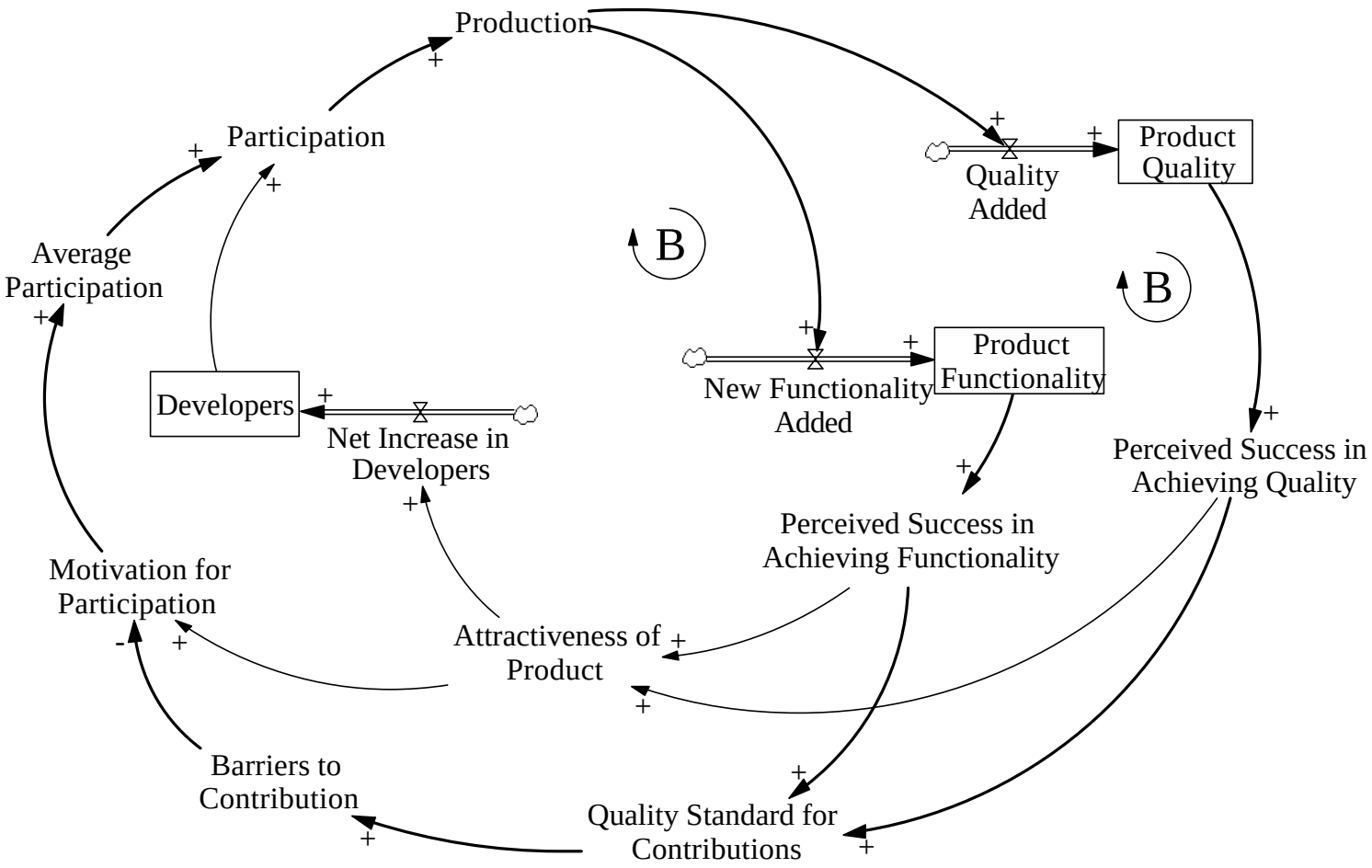


Figure 2.8. Two Balancing Feedback Loops that are Driven by the Negative Effect of Barriers to Contribution on Motivation for Participation

An extension of the arguments made by Raymond (2001) and Fogel and Bar (2001) within the context of barriers to contribution is the concept of barriers to entry, which represents the difficulty of getting accepted to the community as a new developer. Barriers to entry have a balancing effect on the overall structure through the two negative loops shown in Figure 2.9. An increase in the number of developers means more participation, and thus more production, which in turn increases the product functionality and quality. Product functionality and quality increase perceived success in achieving functionality and quality respectively, and those in turn increase the barriers to entry, which has a negative effect on the number of developers.

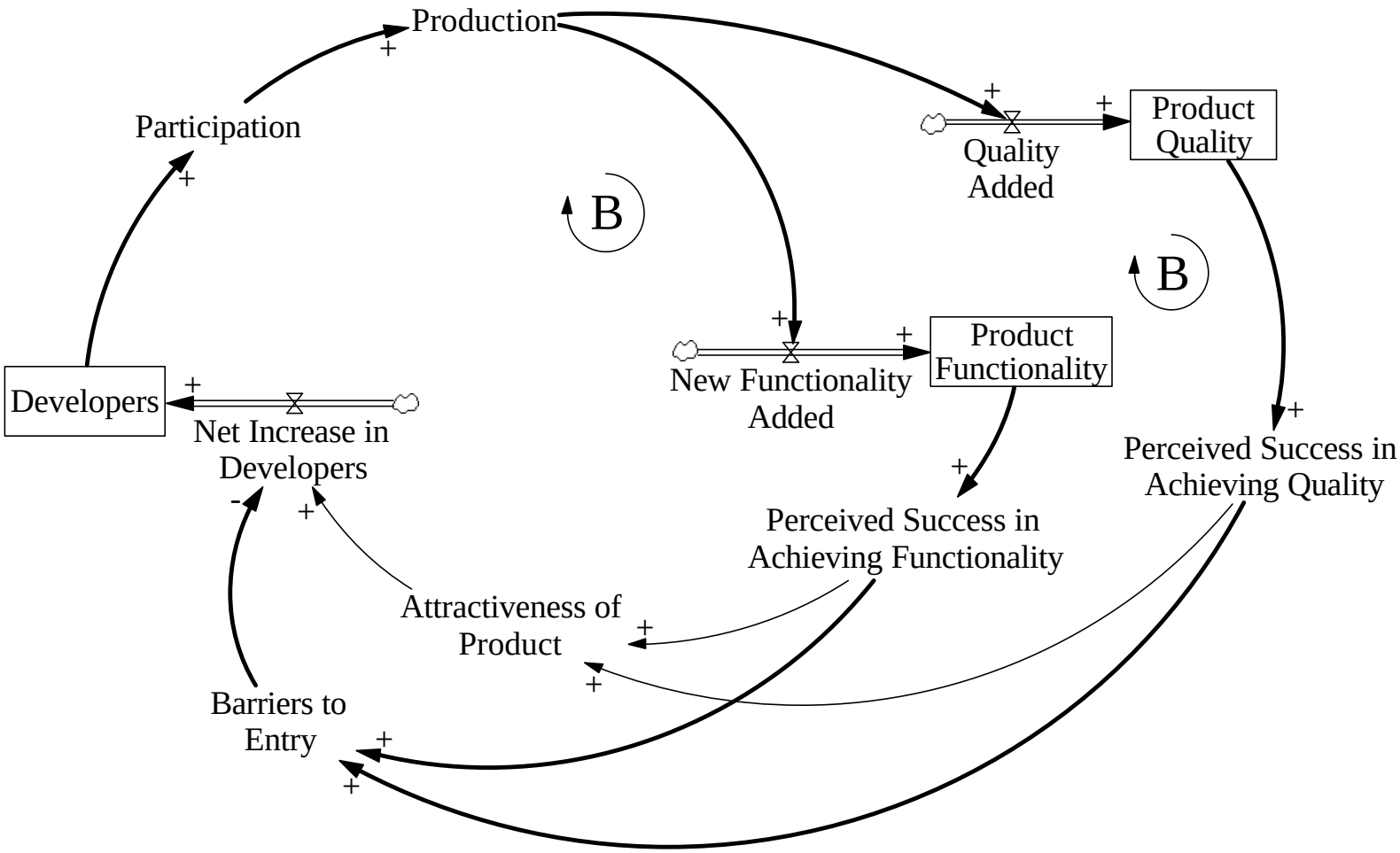


Figure 2.9. Two Balancing Feedback Loops which Work through Barriers to Entry.

Several authors argued that coaching of inexperienced contributors helps increase both the productivity of the inexperienced contributors and the quality of the work they do in the long run (Cox 1998, Fogel and Bar 2001). Accordingly, coaching probably has a positive effect on average developer skill level, and therefore on average productivity. However, in the short run coaching has a negative effect on productivity, since time dedicated to coaching decreases participation dedicated to production. Figure 2.10 shows the changes after adding coaching to the preliminary framework in bold.

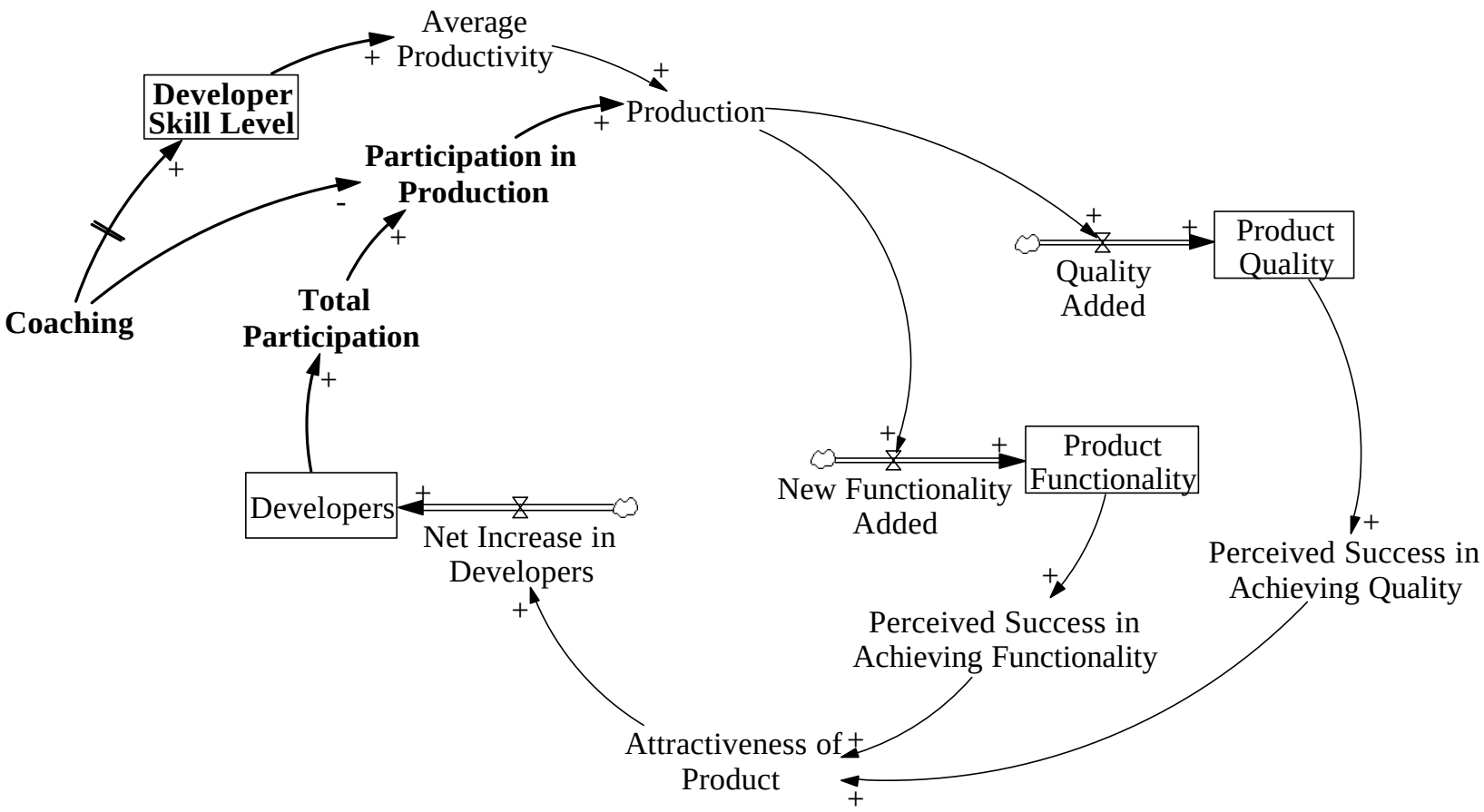


Figure 2.10. Changes in Structure after Adding Coaching to the Preliminary Framework.

According to several authors, technology is the most important driving force behind open online collaboration (Fogel and Bar 2001, Raymond 2001). Here "technology" means a combination of a communication channel, and a collaboration platform. The main and most important communication channel in the context of open online collaboration is the Internet. The Internet makes open online collaboration between project contributors, and dissemination to end-users a truly open and global undertaking. In order to involve and manage mass participation by a high number of contributors, a structured collaboration platform is needed in addition to the communication channel. Fogel and Bar (2001) argue that it is crucial to implement a system which makes collaboration and contribution simple and convenient in order to be able to attract and retain contributors. Raymond (2001) argues that the number of contributors, and consequently the success of the project, is inversely correlated with the difficulty of making contributions. Thus, technology has a positive influence on participation, coaching, and the size of both developer and end user audiences. Figure 2.11 shows the changes after adding technology to the preliminary framework.

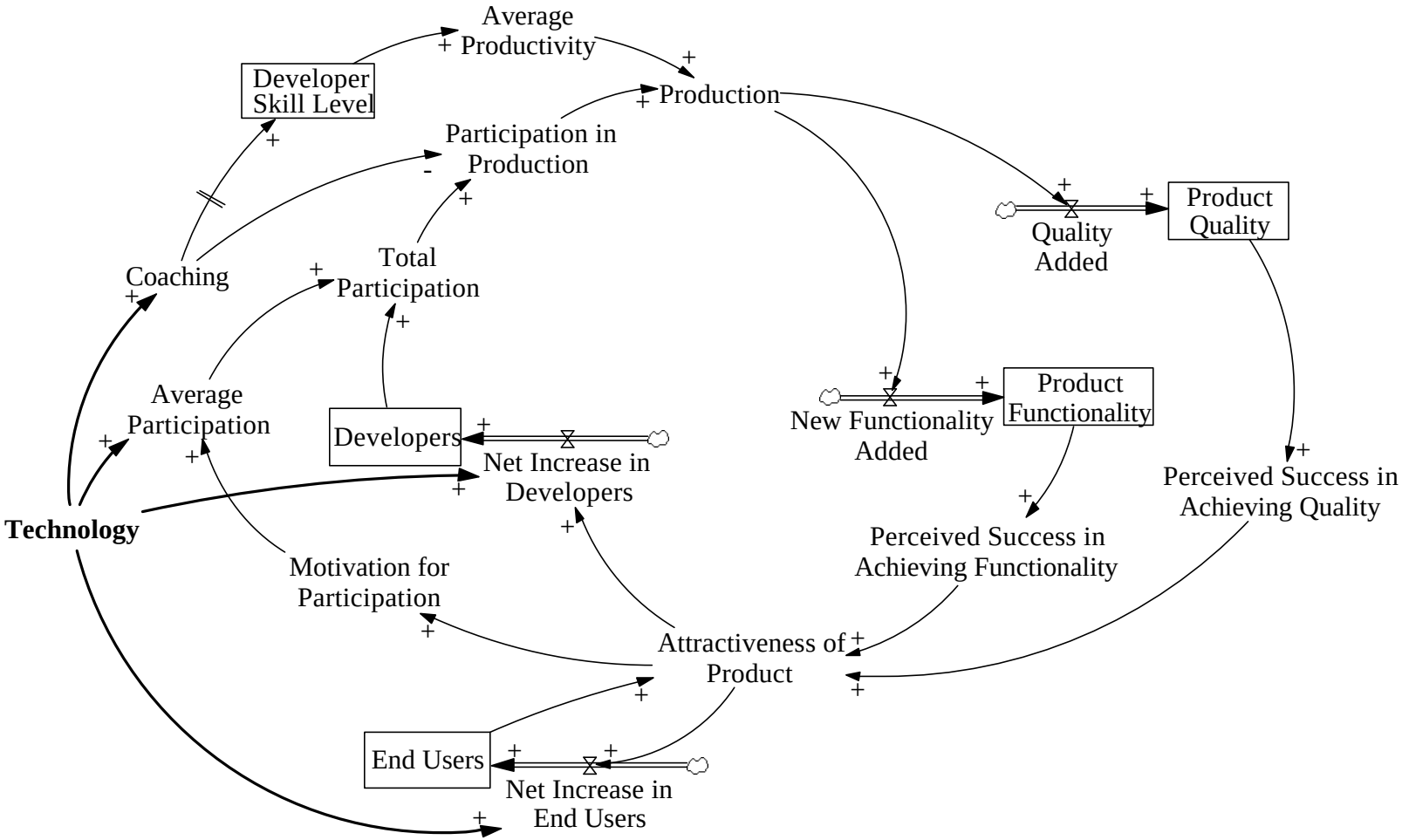


Figure 2.11. Changes in Structure after Adding Technology to the Preliminary Framework.

The dynamic hypothesis discussed in this chapter acted as the basis for the initial system dynamics model of a hypothetical open source software development community. Some other feedback loops and variables, which were not conceptualized as part of the dynamic hypothesis, were also added to the structure of the open source software development (OSSD) model as needed. In the end, the OSSD model successfully replicated the reference behavior patterns of both successful and unsuccessful open online collaboration communities. The detailed structure of the OSSD model, and its behavior under a variety of conditions are discussed in Chapter 4 and Chapter 5, respectively. However, before those, Chapter 3 introduces the methods and the research design used in this study.

CHAPTER 3 -- METHODOLOGY

3.1. Overview

The ultimate goal of this study was to posit a theory of open online collaboration communities in the form of a dynamic feedback framework. A multi-method approach combining qualitative social research methods and system dynamics modeling method was used to achieve this goal. The study began with the building of a system dynamics model of a hypothetical open source software development community. The model was based on three streams of literature: literature on theoretical approaches to the study of online communities, literature on open source software development, and literature on application of system dynamics method to software project management. The second stage of the study involved a series of interviews with the members of a specific community that focused on building instructional materials for introducing system dynamics concepts to K through 12 students. The interviews were used as an instrument to test the applicability of the dynamics that govern the open source software development model to the instructional material development community. The final stage of the study involved the outlining of a theoretical framework that can be applied to studying a range of open online collaboration communities.

Brewer and Hunter (1989 pp.17) define multi-method research strategy as "attack[ing] a research problem with an arsenal of methods that have non-overlapping weaknesses in addition to their complementary strengths." In this study, system dynamics modeling and structured interviews complement each other. The initial system dynamics model acted as an overarching hypothesis for representing open online collaboration

communities. The interviews tested the applicability of the model to an actual community that fits the definition of an open online collaboration community.

3.2. System Dynamics

System dynamics is a methodology for building causal feedback models of complex, large-scale, non-linear, dynamic socio-economic and natural systems. A group of researchers led by Jay W. Forrester introduced the methodology in the early 1960s. Forrester (1961) outlined the methodology and the underlying philosophy behind it in his book Industrial Dynamics. The two main assumptions of the system dynamics methodology are:

(1) direct causal relationships between variables that form the model, and

(2) interdependence of causal factors through feedback loops.

Feedback refers to a two-way causal relationship between variables, where variable X influences variable Y, and after a delay, and perhaps through a series of other variables, Y influences X. This mutual causal influence structure is called a feedback loop. The most basic feedback loop structure consists of two variables. Multiplying the polarities of the causal relationships that form a feedback loop gives the polarity of the overall feedback loop. Positive feedback loops are also called “reinforcing loops,” since there is a mutual reinforcing effect between the variables of a positive loop as it operates. Negative feedback loops are also called “balancing loops,” since the opposite polarities of the causal relationships that form a negative loop force it toward a balance.

Most system dynamics models include a number of both negative and positive feedback loops, which interact and operate simultaneously. Large-scale models include

large numbers of variables, and as a result of that a large number of feedback loops. The lengths of feedback loops vary from two variables to tens of variables within large-scale models; but generally, as the length of the feedback loop increases, its impact decreases.

A complete system dynamic model consists of a diagram that depicts the variables of the model and the causal relationships between them, and the underlying mathematical equations, which represent the algebraic relationships among the variables. Since a system dynamic model is built with the ultimate aim of carrying out dynamics analyses by using computer simulations, a model without a complete set of equations would be incomplete. As stated earlier in this text, system dynamics methodology is used to analyze dynamic systems, in which the variables change through time. Thus, difference equations are the main mathematical structures underlying system dynamics models.

A system dynamics model may be represented by causal loop diagrams, which show the causal relationships between variables without making any distinctions based on their mathematical characteristics. Another way of representing a system dynamics model is using structure diagrams, which depict both the causal relationships and the mathematical characteristics of the variables.

The variables are grouped into three, based on their mathematical characteristics:

(1) Stock (level) variables,

(2) Flow (rate) variables, and

(3) Converters (auxiliaries).

Stock variables represent values that accumulate or decay through time. The value of a stock variable, at a given time, depends on its initial value, and the sum of inflows and outflows over time until the given period.

Flow variables represent the changes in stock variables through time, and they are connected directly to stocks that they change. Stock-flow relationships correspond to differential equations whereby the flows represent the derivatives of stock variables.

Converters represent quantities that are determined at every time increment only by the variables that affect them and not by their previous values. In that sense, a converter simply represents the values of a variable at a given point in time, based on the value of the variables that influence it.

Several authors outlined different procedure to carry out a system dynamics modeling study. Although they are different articulations, most of these approaches map onto the same general procedural outline (Luna and Andersen 2002). Furthermore, each modeler brings personal nuances to system dynamics model building; however, there are general procedures or "best practices" that most modelers follow (Martinez-Moyano and Richardson 2002). System dynamics modeling can be done by a group of people as well as by individual modelers. The last decade witnessed the development of procedures for system dynamics modeling in group settings (Andersen and Richardson 1997, Andersen, Richardson and Vennix 1997).

The system dynamics modeling procedure begins with the problem identification and model conceptualization phase. This stage involves the representation of the key variables of the problem in terms of their behaviors over time. The overall collection of the behaviors of key variables over time is referred to as the "reference mode." The time

horizon over which the problem plays itself out is also defined in this phase. The problem identification and model conceptualization phase also involves the articulation of the system boundary to be modeled. System boundary is drawn by defining the variables that will be included in the model (Richardson and Pugh 1981, Sterman 2000).

Next comes the model formulation phase, where the structure diagram is built and the equations for variables are defined. In most cases, the modeler needs to go back and forth between the problem identification and model conceptualization phase, and the model formulation phase in an iterative fashion in order to revise the problem definition, and the system boundary (Richardson and Pugh 1981, Sterman 2000).

The following phase is model testing, which aims to determine the validity of the model. "All models are wrong" is an axiomatic statement that can be heard frequently in the context of system dynamics modeling. The statement means that any given model is a limited representation of a given portion of the real world, and is prone to be inaccurate. Nonetheless, some models are "more wrong" than the others. The modeler strives to make the model at hand "less wrong." In that sense, testing involves finding out how wrong the model is, and iteratively making it less wrong. It can be argued that no model can be totally "validated," and thus "confidence building" is a better phrase to call the testing stage of a system dynamics study.

Validity (or confidence building) tests can be grouped according to their purpose, and their focus. The purpose of a given test may be to assess:

(1) the suitability of the model to the modeling purpose,

(2) the consistency of the model with reality, or

(3) the usefulness and effectiveness of the model in terms of achieving its purpose.

The focus of the test can be either the structure or the behavior of the system (Richardson and Pugh 1981 pp.314). The modeler frequently goes back to the previous stages of the modeling process in order to refine and reformulate the model. The overall modeling procedure is carried out in an iterative fashion.

The final phase of the system dynamics modeling procedure is the policy analysis and model use phase. This is the stage where alternative policies that address the problem at hand are tested by making use of simulations. The policies that stand out as adequate solutions to the problem are communicated and implemented.

3.3. Structured Interviews

Interviews are an alternative data collection method within the general class of surveys (Babbie 1998 pp.264). Interviews provide an interactive, synchronous data collection process between the data collectors and the subjects. Structured interviews are a variety of the interview method, which involve asking the same set of predetermined questions to all subjects that take part in the research.

Kvale (1996 pp.88) defines seven stages for administering an interview study. The stages are:

Thematizing: This is the stage where the purpose and the topic of the interview are determined.

Designing: This stage involves planning how the interview will be carried out, analyzed and reported. The interview questions are determined and the interview protocol is developed in this stage.

Interviewing: This is the stage where the actual interviews are carried out.

Transcribing: This stage involves preparing the interview data for analysis, generally by typing the notes and the recordings of the interviews in a format that is suitable for analysis.

Analyzing: This is the stage where the interview data are analyzed with the chosen method.

Verifying: This is where the findings of the analyses are verified in terms of generalizability, reliability and validity. Generalizability refers to whether the findings of the study can be used to explain the research phenomena about a wider population, and a wider variety of cases than just those used in the research. Reliability refers to whether the results are consistent, while validity refers to whether the study investigates what is intended to be investigated.

Reporting: This is the stage where the findings are communicated, mostly in written form.

3.4. Research Design

This study was carried out in three phases:

(1) Modeling of a hypothetical open source software development community.

(2) Administration and analysis of interviews with the members of a specific instructional material development community in order to test the applicability of a

generalized version of the open source software development model as a representation of the general dynamics that govern open online collaboration communities.

(3) Positing a theory of open online collaboration communities in the form of a dynamic feedback framework, based on the open source software development (OSSD) model and the findings of the interviews.

3.4.1. Analysis and Modeling of Open Source Software Development

The case of open source software development was analyzed and modeled based on three streams of literature. The analysis of these literature streams roughly maps to the problem identification and model conceptualization stages of the system dynamics modeling process. The literature on the theoretical approaches to the study of online communities and the literature on the theory and practice of open source software development were used as bases for conceptualizing the portions of the model that pertain to the social and psychological aspects of the open source software development phenomenon. Parallels were drawn between the two literature bodies in order to conceptualize variables and the causal relationships between those variables. The literature on application of system dynamics method to software project management, together with the practitioner segment of the literature on open source software development was used in conceptualizing the technical and project management related aspects of the model.

The open source software development (OSSD) model was built through several iterations. Each iteration produced a self-contained, running dynamic feedback simulation model, which is referred to here as a "version" of the model. Each version involves more structure than the previous version, and can explain more about the system

of the hypothetical open source software development community when compared to previous versions. The structure of the model, and the versions are discussed in detail in Chapter 4.

The OSSD model has the potential to test policies that would improve overall system performance, including success factors such as product functionality, product quality, developer talent, and community size in terms of developers and end users. Policy implications of the model, along with the findings of a set of confidence building tests are discussed in Chapter 5.

3.4.2. Interviews with the Members of an Instructional Material Development Community

The second stage of the research involved the development, administration and analysis of a series of structured interviews with the members of a specific instructional material development community, in order to test the applicability of the OSSD model and its policy implications to other open online collaboration communities. The specific community in question is a group of teachers and researchers who develop and disseminate instructional materials for introducing system dynamics concept to K through 12 students.

3.4.2.1. Population

The system dynamics K through 12 instructional materials development community has gathered around four main organizations or groups. Two of these are non-profit organizations propagating systems thinking and system dynamics in K through 12 education. The other two are research and practice groups working on developing

instructional materials for introducing system dynamics concept to K through 12 students. The interviewees were affiliated with the two organizations and one of the research and practice groups. Namely, the Creative Learning Exchange, the Waters Foundation, and CC-STADUS. No subjects affiliated with the System Dynamics in Education Project could be recruited for the interviews.

The Creative Learning Exchange (CLE) is a non-profit organization that propagates systems thinking and system dynamics approaches in K through 12 education. The CLE has two main functions that are aimed at fulfilling its mission. The first is a biannual conference that brings together teachers, mentors, researchers and activists who work on applying systems thinking and system dynamics concept to K through 12 education. The other main function of the CLE is to act as a clearinghouse and outlet for K through 12 instructional materials that use systems thinking and system dynamics as teaching tools. The CLE has an active website (clexchange.org) for gathering and disseminating such materials. Submissions are open to all. The website includes materials submitted by the affiliates of other K through 12 education organizations focusing on systems thinking/system dynamics, such as the Waters Foundation, CC-STADUS and MIT System Dynamics in Education Project, as well as individual authors who are not affiliated with such organizations. Consequently, the CLE website is the main repository of instructional materials for introducing system dynamics concepts to K through 12 education. The materials go through a volunteer-based review process before being posted on the website.

The Waters Foundation is non-profit organization that maintains a network of educators who do research and develop instructional materials related to systems

thinking/system dynamics for application in K through 12 education. The Waters Foundation network consists of "sites," which actually are school districts. Currently, there are 12 sites in the network: Carlisle Public Schools (Carlisle, MA), Catalina Foothills School District (Tucson, AZ), Chittenden South Supervisory Union (Chittenden County, VT), College Community School District (Cedar Rapids, IA), Glynn County Schools (Brunswick, GA), Greater Tucson Area (Tucson, AZ), Harvard Public Schools (Harvard, MA), James Bennett High School (Salisbury, MD), LaSalle College Preparatory High School (Milwaukie, OR), Murdoch Middle School (Chelmsford, MA), Portland Public Schools (Portland, OR.), Salvadori Education Center (New York City, NY). Every site has one or more mentors who assist educators in developing systems thinking/system dynamics based instructional materials, and apply these concepts to their classes. The mentors also train administrators and other staff in several sites. Instructional material develop at the sites are disseminated through the Waters Foundation website.

CC-STADUS (Cross-Curricular Systems Thinking and Dynamics Using STELLA) was a project supported by a National Science Foundation (NSF) grant, which had the purpose of training high school teachers for applying systems thinking/system dynamics concepts in the classroom. CC-STADUS had a website for disseminating instructional materials that were built as part of the project; however, the website went off-line after the project was completed. Most of the CC-STADUS materials now reside on the CLE website.

MIT System Dynamics in Education Project (SDEP) is a project aimed at developing a collection of self-study materials that introduce system dynamics. The collection is called Road Maps, and is developed by a group of MIT students under the

guidance of Professor Jay W. Forrester. The Road Maps collection was disseminated through SDEP's own website, until it was moved to the CLE website.

The rationale for choosing this specific community for study was twofold. First, the community was highly accessible for the researcher due to personal links between the members of the dissertation advisement committee and the members of the community. This fact made the selection and recruitment of the interviewees considerably easier. Also, since the members of the community were knowledgeable about system dynamics method, assessment of their opinions about the applicability of the model to their community was substantially easier than it would be with subjects who were not knowledgeable about system dynamics. These subjects could comprehend system dynamics diagrams fast and accurately, as well as being able to articulate their views using system dynamics terminology, making use of graphs over time, feedback loops, and stock-and-flow diagrams.

3.4.2.2. Sample Method and Rationale

A purposive, snowball sample of 10 experts from the overall population of system dynamics K through 12 teachers and researchers were used for the interviews. Kvale (1996 pp.102) found that the number of interviews in current qualitative interview studies tend to be between 5 and 25, with an average of roughly 15. Kvale attributed this to two factors. One is the fact that the time and resources available for carrying out the interviews are limited. The second factor is the law of diminished returns, which suggests that each additional interview will add less to the findings, and the contribution of an additional interview will be negligible once a certain number is reached.

The snowball sampling process was initiated with a list of 21 individuals that were involved in the system dynamics K through 12 community. The initial list was compiled with the help of George Richardson, who was very knowledgeable about the said community and its members. Based on George Richardson's suggestion, two key individuals on the list, who have ample connections within the community, were also contacted to ask for additional names to be added to the list of potential subjects. The suggestions of those two individuals did not add any more names to the list, since all the individuals they suggested as potential subjects were already on the list. Furthermore, the interviewees were asked at the end of the interviews for additional names to be contacted as potential subjects. However, the answers to that question did not add any names to the list either, since all the suggested individuals were already on the list. In summary, the snowballing process started and ended with the same list of individuals as potential subjects.

An important limitation of the specific community studied was the low number of potential interview subjects for a research of this detail. Although the numbers of contributors and end users within the community were reasonably high, the number of individuals who could provide the level of information asked through the interviews was quite low. The list of 21 potential subjects was a very optimistic list in terms of accessibility and knowledge level about the detailed working of the community. The initial assessment of the list of potential subjects suggested that their level of familiarity with the detailed workings of the community was highly variable. Also, it became clear that not all of the potential subjects were accessible, and willing to participate. In the end, the group of interviewees included most of the key people from the main centers

described above, who have considerably long experience in the field, and a good understanding of how the community works. Five of the interviewees were mentors, three were educational researchers, and two were community leaders/activists. One of the mentors was retired, while all other interviewees were active. One of the mentors focused mostly on kindergarten through elementary education, while the others worked in middle and high school settings. One of the researchers had worked as a principal at one time. Four interviewees were male, and six were female. Nine interviewees worked in the United States -- four in the northeast, three in the northwest, one in the southeast, and one in the southwest -- while one interviewee worked outside of U.S, in a predominantly English-speaking country. Consequently, all interviewees were from English-speaking countries.

3.4.2.3. Data Collection

The potential subjects were initially contacted by e-mail (See Appendix A.1 -- Initial E-mail Request) to ask whether they would participate in the interviews. Follow-up e-mail messages (See Appendix A.2 -- Follow-up E-mail Messages) were sent to potential subjects according to whether they agreed to participate or not. Potential subjects who agreed to participate received a packet containing a cover letter (See Appendix A.3 -- Interview Packet Cover Letter), a consent form (See Appendix A.4 -- Particiaption in Research Consent Form), reference mode worksheets to be used during the uninformed portion (See Appendix A.5 -- Reference Mode Worksheets), model sketches to be used during the informed portion (See Appendix A.6 -- Model Sketches), and return envelopes for the consent form and the reference mode worksheets. The interviews were administered over the telephone, and the conversations were recorded on

audiotape, with the approval of the interviewees. One interview was administered face-to-face at the request of one of the subjects, and that interview, too, was recorded on audiotape. The phone interviews lasted an average of 119.3 minutes, with a maximum of 137 minutes and a minimum of 101 minutes. The standard deviation was 11.1 minutes. The face-to-face interview lasted 135 minutes.

The interview consisted of two parts. (See Appendix A.7 -- Interview Protocol.) The first, uninformed part was aimed at obtaining information about the specific community before exposing the subjects to the generalized OSSD model. The second, informed part involved exposing the subjects to diagrams from a generalized version of the OSSD model and obtaining their opinions about the applicability of the generalized model and its policy implications to their community.

The uninformed portion of the interview was developed based on a list of variables derived form the dynamic hypothesis. Each variable corresponded to one or more questions that aimed to measure it. A list of the variables used for the development of the uninformed part of the interview is given in Table 3.1. The first two question of the uninformed part were designed to ask how the interviewees got involved in the community and their roles within the community. The third question was about the interviewees' general observation about the efforts within the community to develop and disseminate instructional materials. The first three questions also served the purpose of "warming up" the interviewees and focusing their attention on the topic to be discussed. The following 12 questions, Questions 4 through 15 were designed to measures the variables derived from the dynamic hypothesis, as listed in Table 3.1. The uninformed portion of the interview protocol included four more questions aimed at assessing the

views of the interviewees about the policy problems within the community, and possible scenarios about the future of the community. These questions also involved assessing the interviewees' observations and expectations about the existing and future behaviors of key performance measures and determinants of success within the community. The questions in the uninformed portion were refined through several iterations based on discussion with my advisors Deborah Andersen and Karl Rethemeyer.

Table 3.1. List of Variables and Corresponding Measures for the Uninformed Portion of the Interview

Variable Name	**Definition**	**Interview Question for Measurement of Variable**
Motivation	The level of motivation developers feel to participate in the project.	Questions 4c-7b.
Coaching	The level of coaching among developers.	Questions 8a-8c-8e.
Participation	The amount of time spent by developers on the project.	Question 4b.
Barriers to Entry	Scrutiny level for accepting new developers.	Questions 5-6
Barriers to Contribution	Scrutiny level for approving proposed contributions.	Question 10.
Product Quality	The quality of products produced.	Questions 9-10-11.
Product Functionality	The functionality of products produced.	Question 14
Attractiveness of Product	The attractiveness of products for developers and end users.	Questions 7a-15.
Production	The amount of production effort per time period.	Question 12.
Technology	Availability of effective mass digital communication.	Questions 8b-8d.
End Users	The number of users of the products.	Question 13.
Developers	The number and skill levels of developers.	Question 4a.

The informed portion of the interview was developed based on the Iteration V version of the OSSD model. The focus of the informed part was on the main loops that reinforce and limit the growth and the overall success of the community, as well as a series of policy interventions. The major reinforcing and limiting loops in the OSSD model were represented in a series of simplified stock-and-flow diagrams in order to be shown to the interviews and ask whether they observed similar dynamics at work in their community. The sketches included only the variables that are crucial for understanding the basic structure of the model and revealed each loop gradually. Many converter type variables were hidden in the diagrams in order not to complicate communicating the model to the interviewees. Furthermore, certain outflows and loops were omitted from the diagrams in order to simplify communication and comprehension of the model. The variable names used in the sketches were different than those in the OSSD model in order to represent concepts that would fit the case of the instructional materials development community. For example, the variable name "developers" became "authors," "bugs" became "errors," and "product functionality" became " functionality of materials." Printed diagrams were sent to interviewees in sealed envelopes. Interviewees opened the sealed envelopes at the beginning of the second, informed portion of the interviews upon a prompt from the interviewer. A narrative was also developed to accompany the sketches. The narrative was read to the subjects while they studied the sketches.

Four possible policy intervention options were also discussed with the interviewees. These were:

- Filtering materials produced by inexperienced authors,

- Reviewing and editing existing materials in the collection,

- Selecting new inexperienced authors based on their talent level, and

- Coaching inexperienced authors.

These policy options were represented as "pure" interventions in the sense that they were represented singly and in a totally separate fashion. For example, the filtering option was represented as a pure, flat "accept or reject" policy without rework or review. On the other hand, the reviewing and editing option involved solely rework on existing material, without elimination of poor material. The rationale behind this approach was to expose the interviewees to simple policy options that are easier to communicate and comprehend. Another important reason for this approach was to elicit the observations and mental models of the interviewees in an indirect manner, with the least amount of interference by exposing them to existing model structure.

The four policy options were also represented in four series of sketches and supporting narratives. These sketches and narratives were developed to explain the four policy options with their potential positive and negative consequences to the interviewees and ask whether they observed any of those policies being implemented in their community. The informed portion concluded with four additional questions that asked whether the interviewees had anything to add to the discussion at the end of the interviews, additional potential subjects, and the interviewees' suggestion for additional questions for future interviews. The questions in the informed portion were refined through several iterations based on discussion with my advisors Karl Rethemeyer and George Richardson.

The interview protocol was piloted with a Ph.D. student from the University at Albany's Information Science doctoral program The Ph.D. student was knowledgeable

about system dynamics in general and the topic of this study in particular. The pilot interview was done face-to-face. Due to the small number of potential subjects no piloting was done with individuals from the subject pool.

3.4.2.4. Interview Data Analysis

The interview data were analyzed in a qualitative and exploratory fashion. This approach was mostly driven by the nature of the interview protocol. (See Appendix A.7 for the complete interview protocol, and Appendix A.5 and Appendix A.6 for the worksheets and the sketches used during the interview.) The interview protocol was designed in order to foster wider interaction between the interviewer and the interviewee. This provided deeper information about the interviewee's observations and mental models with limited interference from the interviewer. The interviewees were encouraged to talk freely about their experiences, and to explore and discover their own mental models. This approach provided thick, rich qualitative data, which was much more adequate for a qualitative analysis approach than a quantitative one. Another important reason for using a qualitative approach to the analysis of the interview data was the limited sample size, which did not allow for plausible quantitative analysis. Further details about the analysis stage are given in Chapter 6.

3.4.3. Development of a General Dynamic Feedback Framework for Open Online Collaboration Communities

The final phase of the study involved the development of a theory of open online collaboration communities in the form of a dynamic feedback framework. The findings of the interviews were used to refine the generalized OSSD model to reach a general

dynamic feedback framework that is applicable to a wider range of open online collaboration cases.

The main approach was to review the reinforcing and limiting loops and policy intervention options based on information gathered from the interviewees. If many interviewees argued strongly against a loop, that loop was removed or changed based on interviewees' suggestions. If a few interviewees argued against a loop, and not forcefully, the loop was marked suspicious, and revised. Changes might or might not be made on such loops. Loops that were confirmed or at least not challenged by interviewees were kept as they were, unless a causal link on them was challenged. If a specific causal link was challenged on a loop, only that link was revised.

The final dynamic feedback framework is a simplified causal loop/stock-and-flow diagram that represents the basic dynamic feedback structure of an open online collaboration community in terms of causal relationships and loops. The framework is further discussed in Chapter 7.

The first step toward developing the framework was the open source software development model. The next chapter discusses in detail the structure of the model and how it was built.

CHAPTER 4 -- OPEN SOURCE SOFTWARE DEVELOPMENT MODEL

4.1. Process of Building the OSSD Model

The open source software development (OSSD) model was based on the dynamic hypothesis introduced in Section 2.6, which, in turn, was based on the implication of the literature review summarized in Sections 2.3 through 2.5. Additional structures, which were conceptualized after the dynamic hypothesis development phase, were also integrated to the OSSD model. The OSSD model evolved through several iterations. Each iteration produced a self-contained system dynamics model, and each iteration added more explanatory power to the overall model. Structurally, each version was built by adding more structure to the version that preceded it. In that sense, each version contains the previous version, and some additional structure. Versions were finalized as self-contained units at critical stages of development; such as adding the notion of product quality or adding the concept of coaching and its effect on average developer talent. The following sections describe the five versions of the model in the order of development.

4.2. Iteration I: Functionality

This initial iteration is focused on the dynamics of building product functionality, and developer and user pools. Here, the functionality of the product is a construct that reflects the general usefulness of the product for the intended tasks. The functionality of a given software product can be defined in numerous ways. One way is to define it as all the tasks that can be done using the software. There can be different definitions for different kinds of software products, and even for the same kind of product, depending on the type of the users in question. For example, in the case of a spreadsheet program, the functionality can be defined as the editing and formatting features for one group of users,

while for another group of users it might be defined as the number of built-in functions. Another way is to define the functionality as the combination of these two definitions. For the purposes of this study product functionality is defined as the general level of usefulness of a software product for a wide array of users.

The Iteration I version of the OSSD model consists of two sectors: Developers Sector and Users Sector. These two sectors are explained below, followed by how they are related in order to form the overall model.

The Developers Sector of the model represents the casual relationships between the developers' production effort and the product functionality level. Developers produce code, adding functionality to the product, and in turn the level of product functionality affects the developer population.

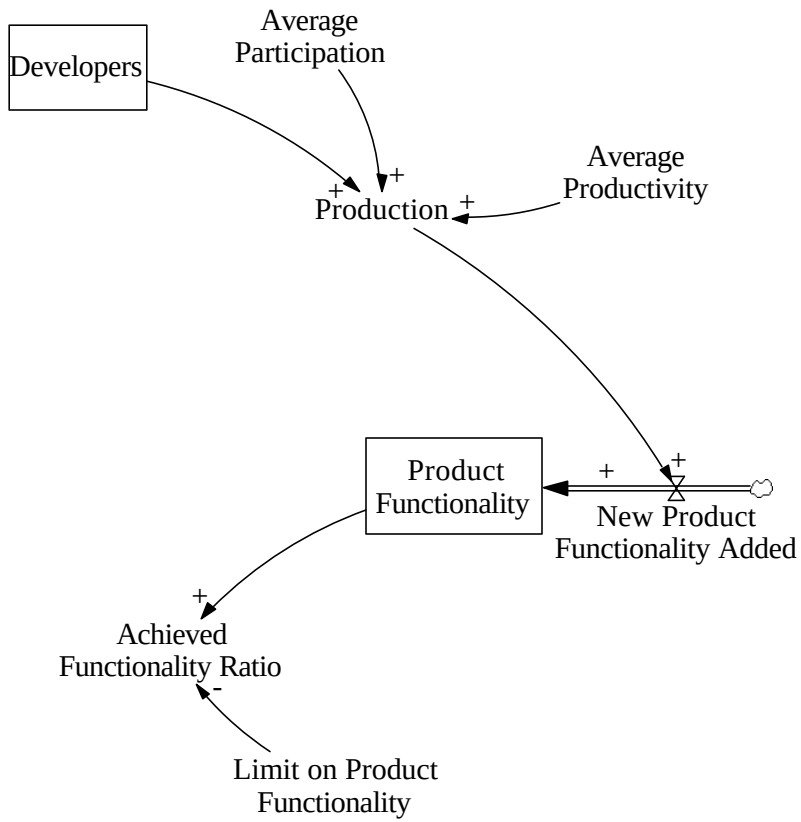


Figure 4.1. OSSD Model (Iteration I) Developers Sector

As demonstrated in Figure 4.1, an initial group of developers participate in code production and add functionality to the product. As new functionality is added to the product the overall product functionality increases, and so does Achieved Functionality Ratio, which is defined as the ratio between the actual Product Functionality and Limit on Product Functionality. Limit on Product Functionality is the maximum possible level of functionality that can be expected from a software product comparable to the product in question. Limit on Product Functionality is not a fixed ceiling since technology changes over time, and the level of functionality for a given kind of software product increases over the years (See Figure 4.2).

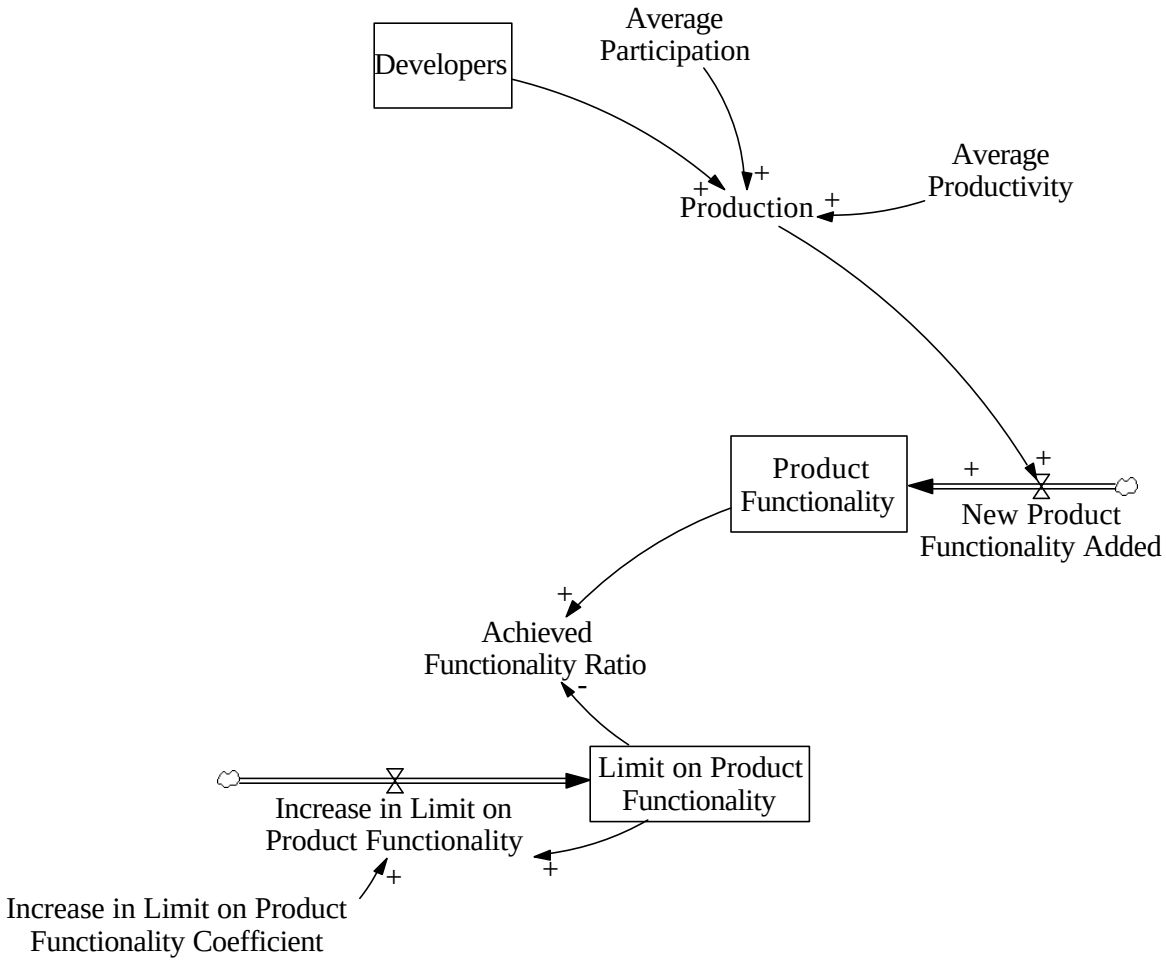


Figure 4.2. OSSD Model (Iteration I) Developers Sector

There are two potential mechanisms that may slow the process of adding functionality to the product (See Figure 4.3). First is the potential decline in average productivity of the developers as the developer population increases. As the number of developers increases and approaches the Productive Developer Population Limit, the average productivity of the developers would decline, due to the diminishing returns on marginal addition of contributors. Average Production is defined as the average number of lines of code written per hour by a developer. As such, this first mechanism limits the basic code writing productivity. The second potential limiting mechanism works through the achieved functionality ratio. As the product functionality approaches the limit on product functionality, it becomes more difficult to add marginal functionality to the product. Thus each unit of code adds less functionality to the product.

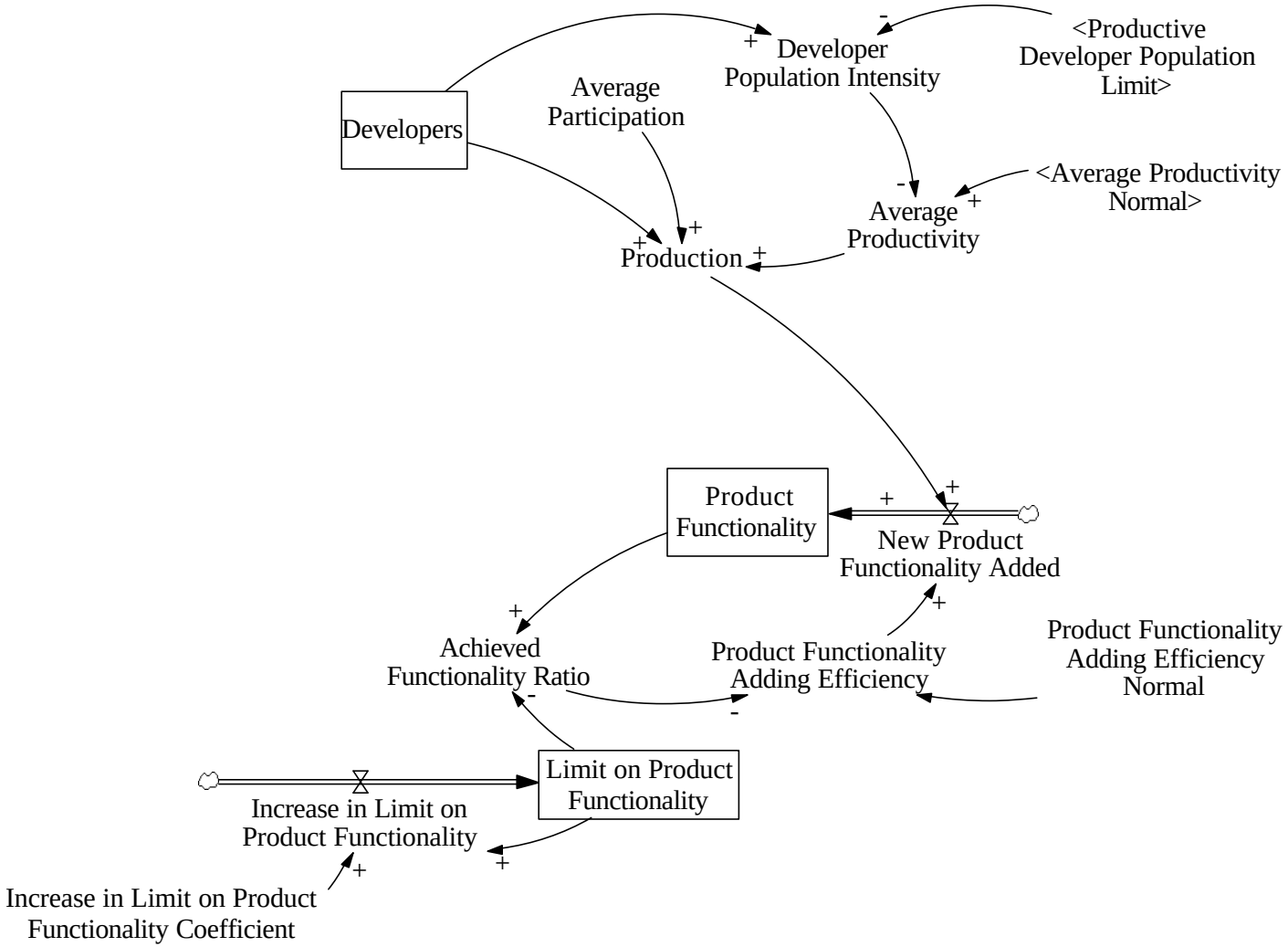


Figure 4.3. OSSD Model (Iteration I) Developers Sector

As Figure 4.4 shows, the number of developers increases as new developers join the community. New developers come from the pool of potential developers as a normal fraction of that pool at any given time period. This fraction is an ideal number, which is limited by the relative attractiveness of the product for developers (See Figure 4.5).

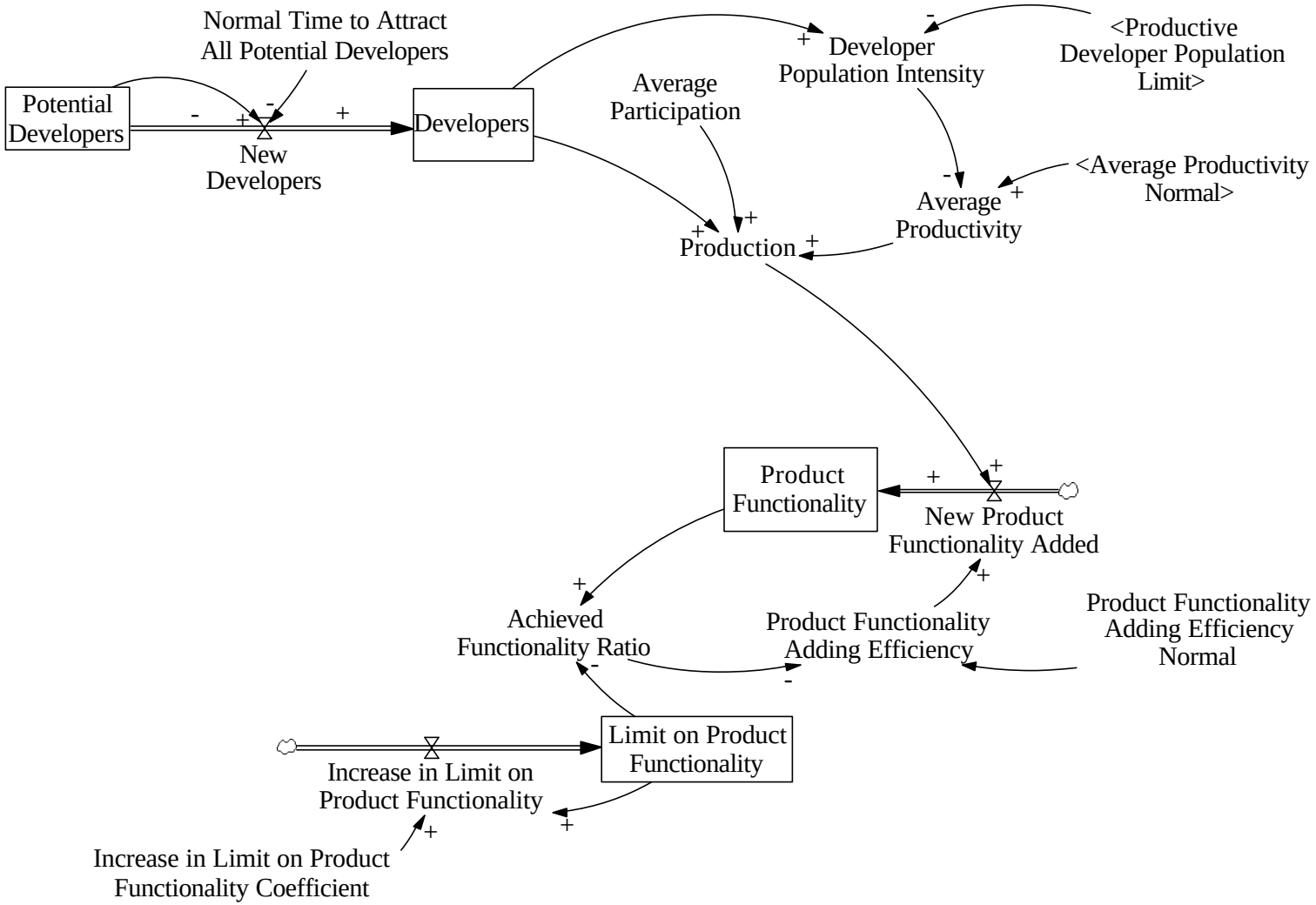


Figure 4.4. OSSD Model (Iteration I) Developers Sector

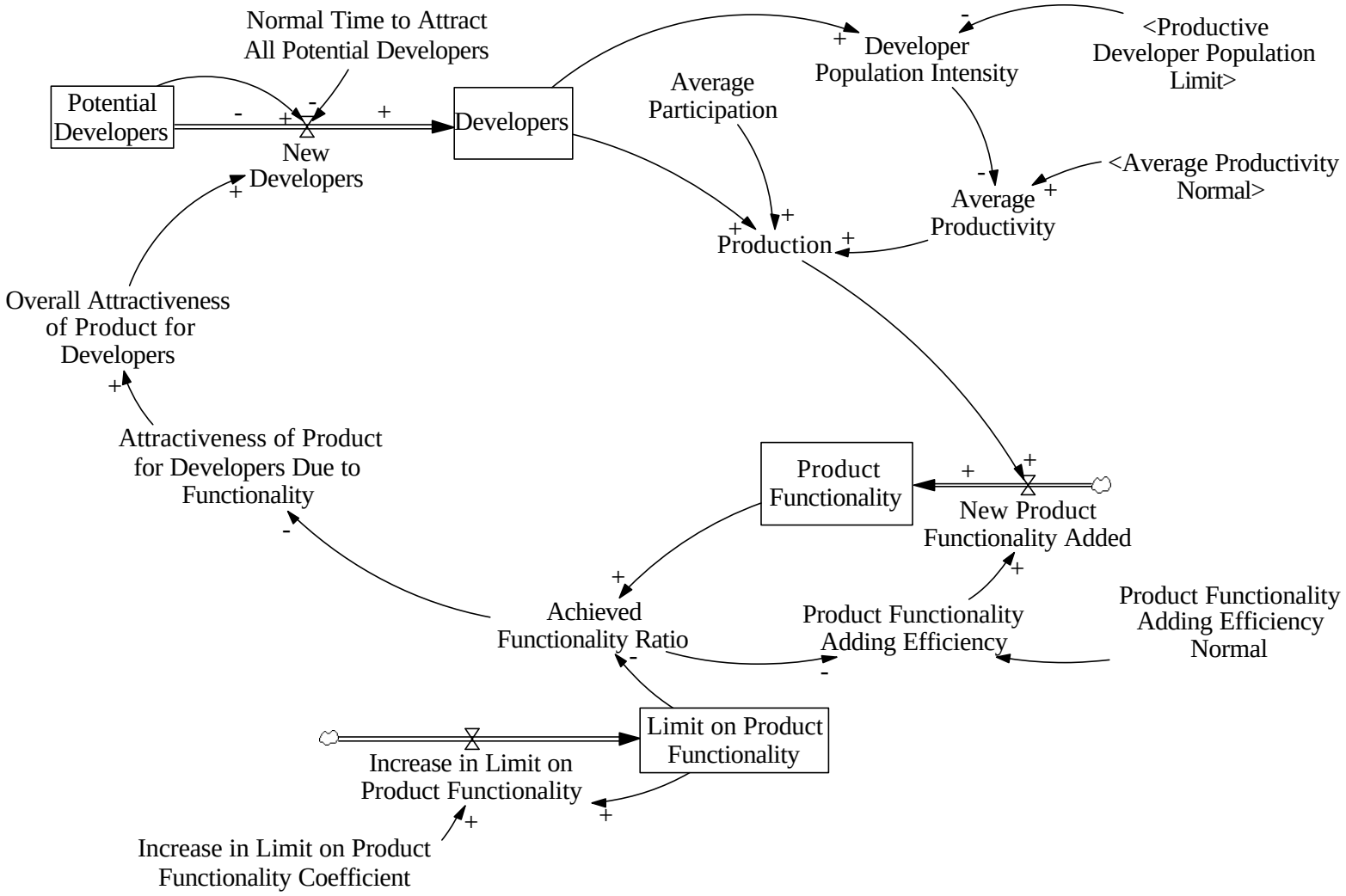


Figure 4.5. OSSD Model (Iteration I) Developers Sector

An important component of the attractiveness of a product for developers is the amount of unachieved functionality. This follows from Raymond's (2001) concept of "homesteading" as discussed in the literature review section. (See Section 2.3.3.) Raymond suggested that among other things, developers are attracted to participate in an open source software project if they can "homestead" and claim a certain segment of the project to themselves. If an open source software product is in its maturity stage and most of the potential functionality is already added, the product would become less attractive for the developers, because there would not be enough unachieved functionality to be homesteaded. Accordingly in the model, attractiveness of the project for developers decreases as the product functionality approaches the limit on product functionality. This is also in accord with the motivation factors discussed under the public goods section of the literature review. We can argue that developers would be attracted to projects that provide substantial opportunities for contributions whether they are motivated by reputation, self-efficacy or even altruism. If the opportunities for contribution are scarce, they would not be attracted.

Just as there are new developers that join the community, there are developers that leave the community (See Figure 4.6). Developers leave the community at a normal rate, which accelerates as the opportunities for contribution decrease (See Figure 4.7). Towards the end of the project, product functionality approaches the limit on product functionality. This means that most of the potential functionality is already added to the product, and most of the developers have completed their parts within the project. These developers would want to move on to other software projects or alternative activities, and that would accelerate the rate of developer departure substantially. At the end of the

project, only a small number of developers would stay for maintenance purposes to keep the product up-to-date as the general level of technology develops and the limit on product functionality increases slowly over time.

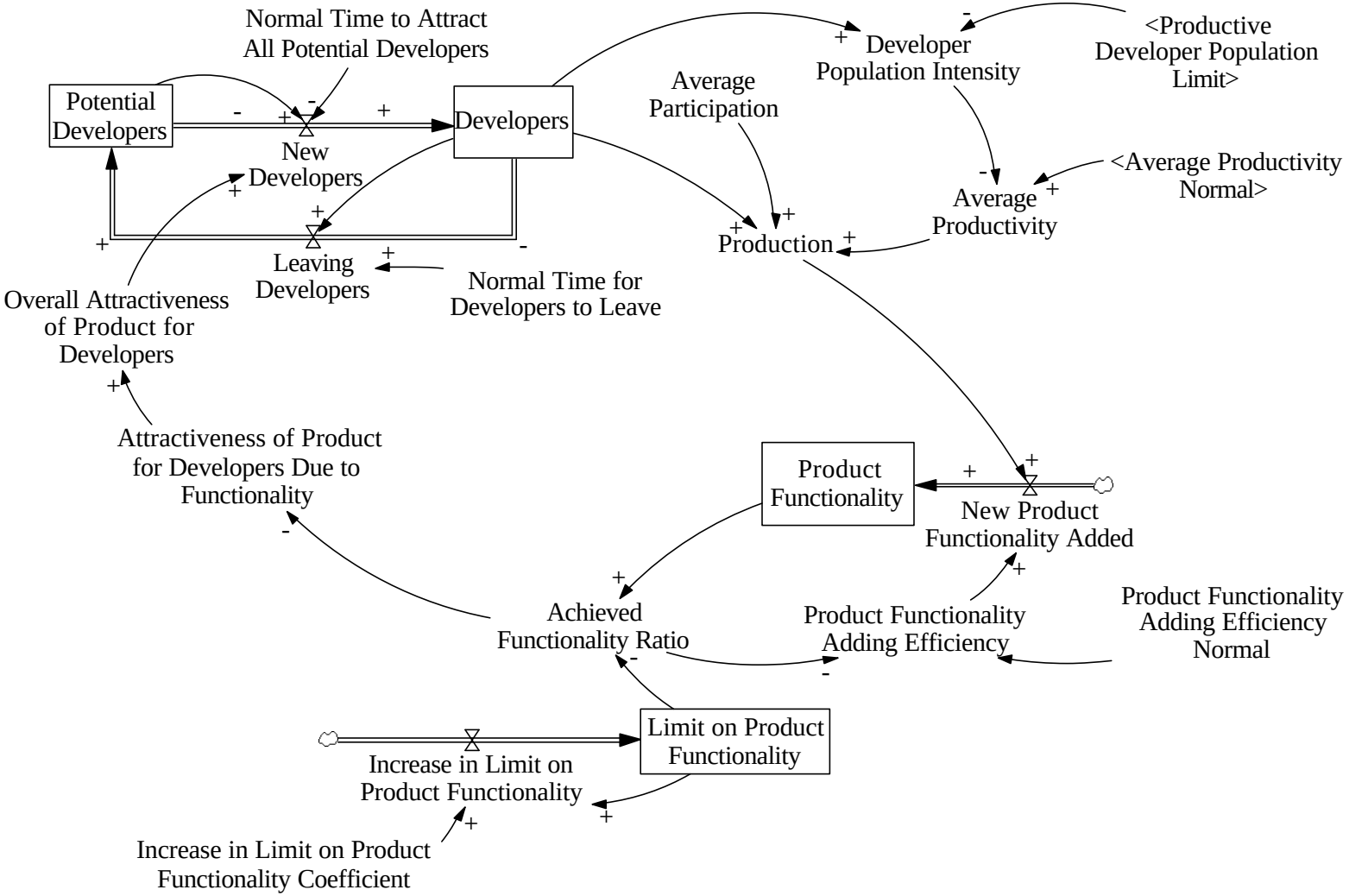


Figure 4.6. OSSD Model (Iteration I) Developers Sector

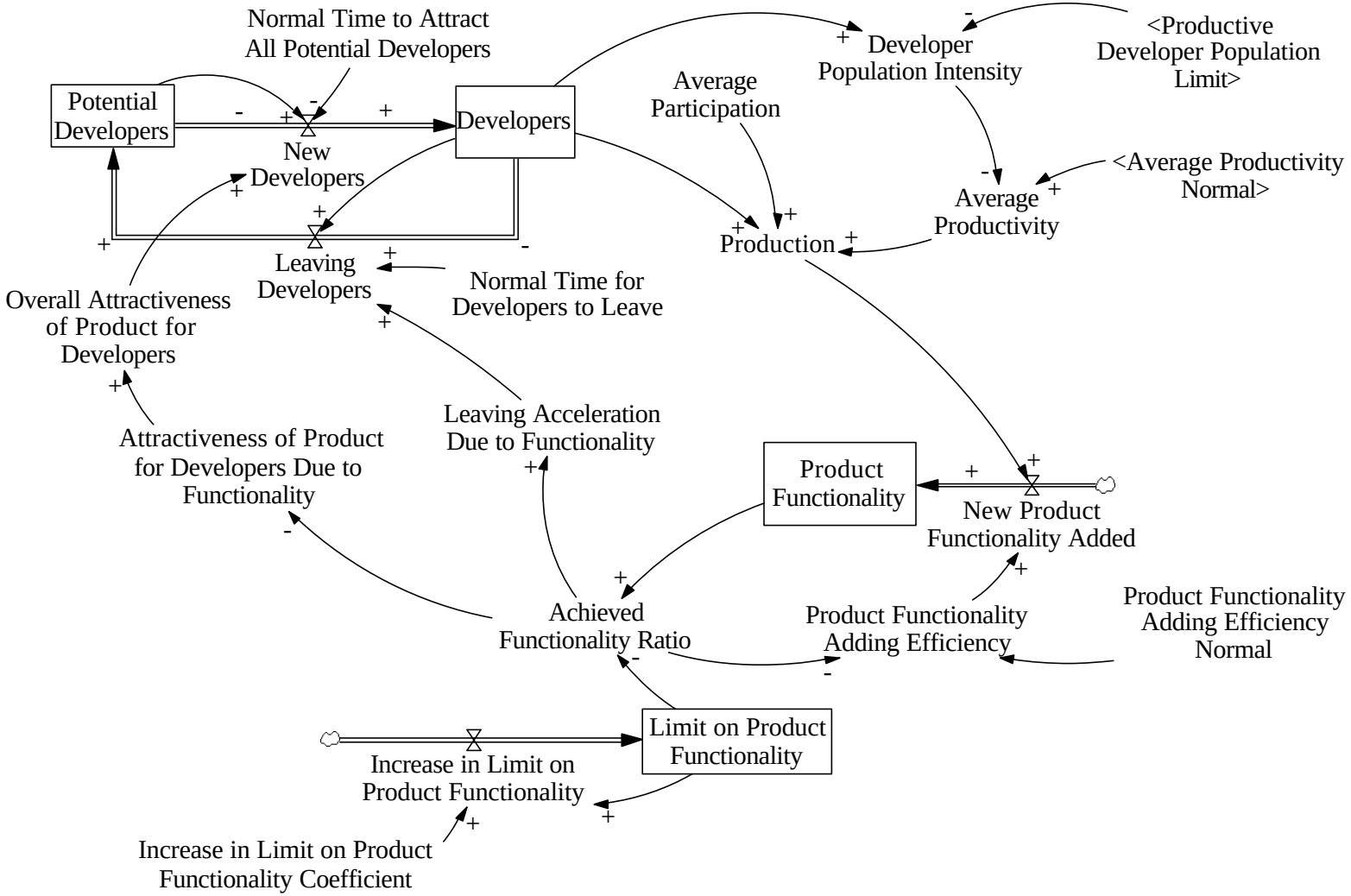


Figure 4.7. OSSD Model (Iteration I) Developers Sector

The Users Sector is the other main part of the Iteration I version of the model. This sector represents the causal relationships between the level of achieved product functionality and the growth of the product's user pool. The Users Sector also represents the effects of the number of users of the product on attracting potential users and developers.

New users are added to the product's user pool as potential users adopt the product. New users are attracted at a normal rate, which is a fraction of the potential user pool. This fraction is an ideal number, which is limited by the relative attractiveness of the product for users. The attractiveness of the product for users is influenced positively by the level of achieved product functionality (See Figure 4.8).

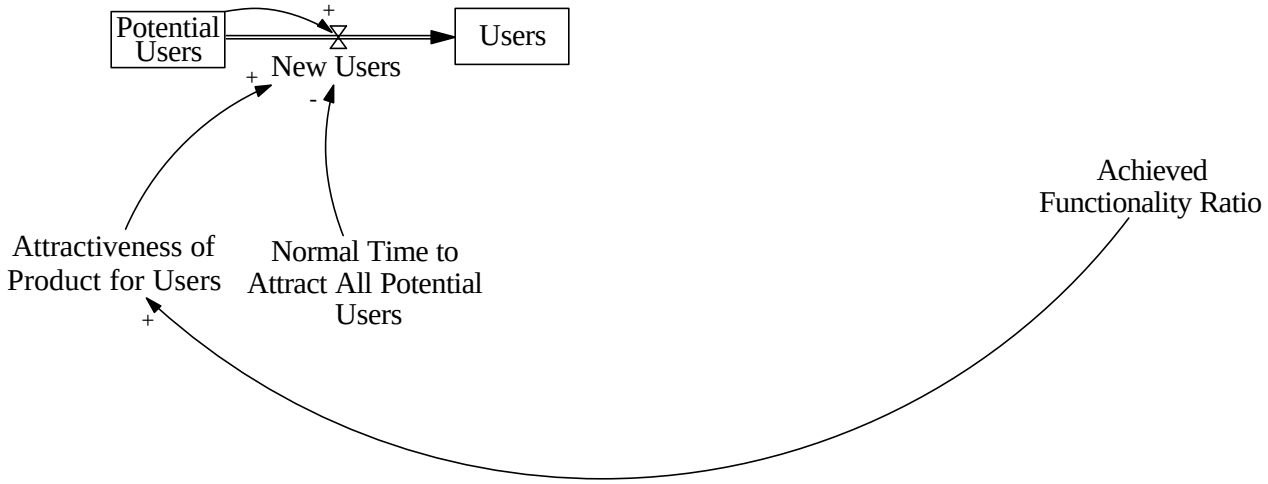


Figure 4.8. OSSD Model (Iteration I) Users Sector

The flow of new users into the product's user pool accelerates as the level of success in attracting users increases. The success in attracting users is based on the relative number of users of the product, with respect to the total user population (See Figure 4.9).

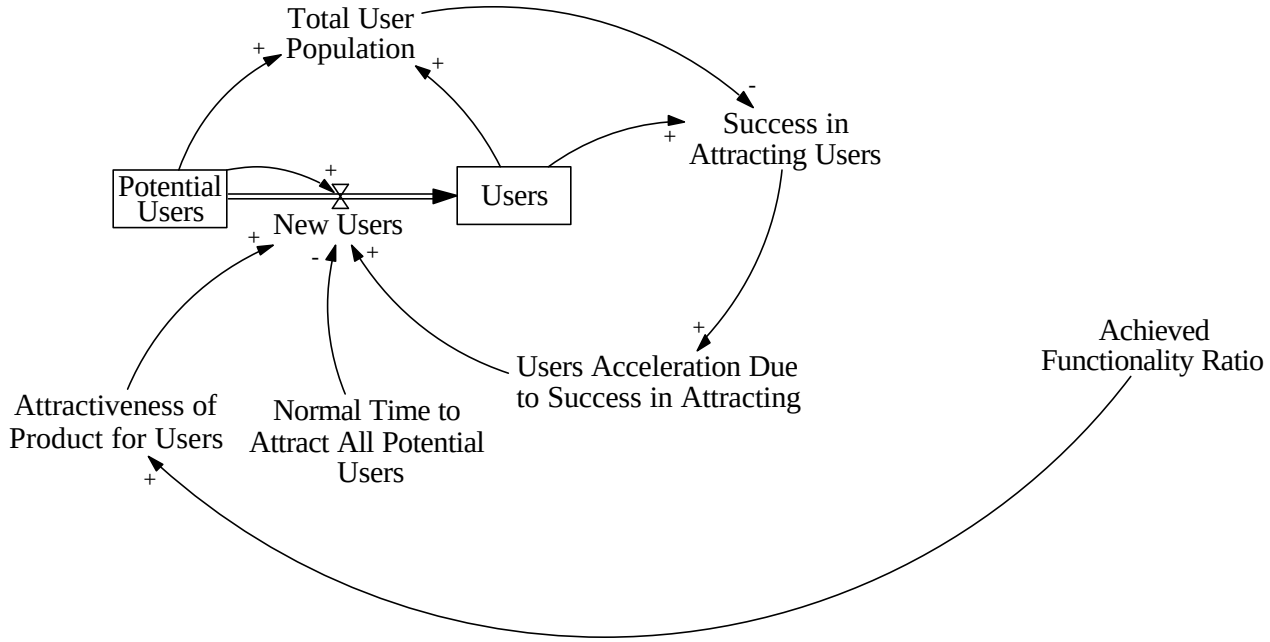


Figure 4.9. OSSD Model (Iteration I) Users Sector

Success in attracting users influences the attractiveness of the product for developers positively, as well. Attractiveness of Product for Developers Due to Users and Attractiveness of Product for Developers Due to Functionality together determine the Overall Attractiveness of Product for Developers (Figure 4.10), which in turn influences the number of new developers.

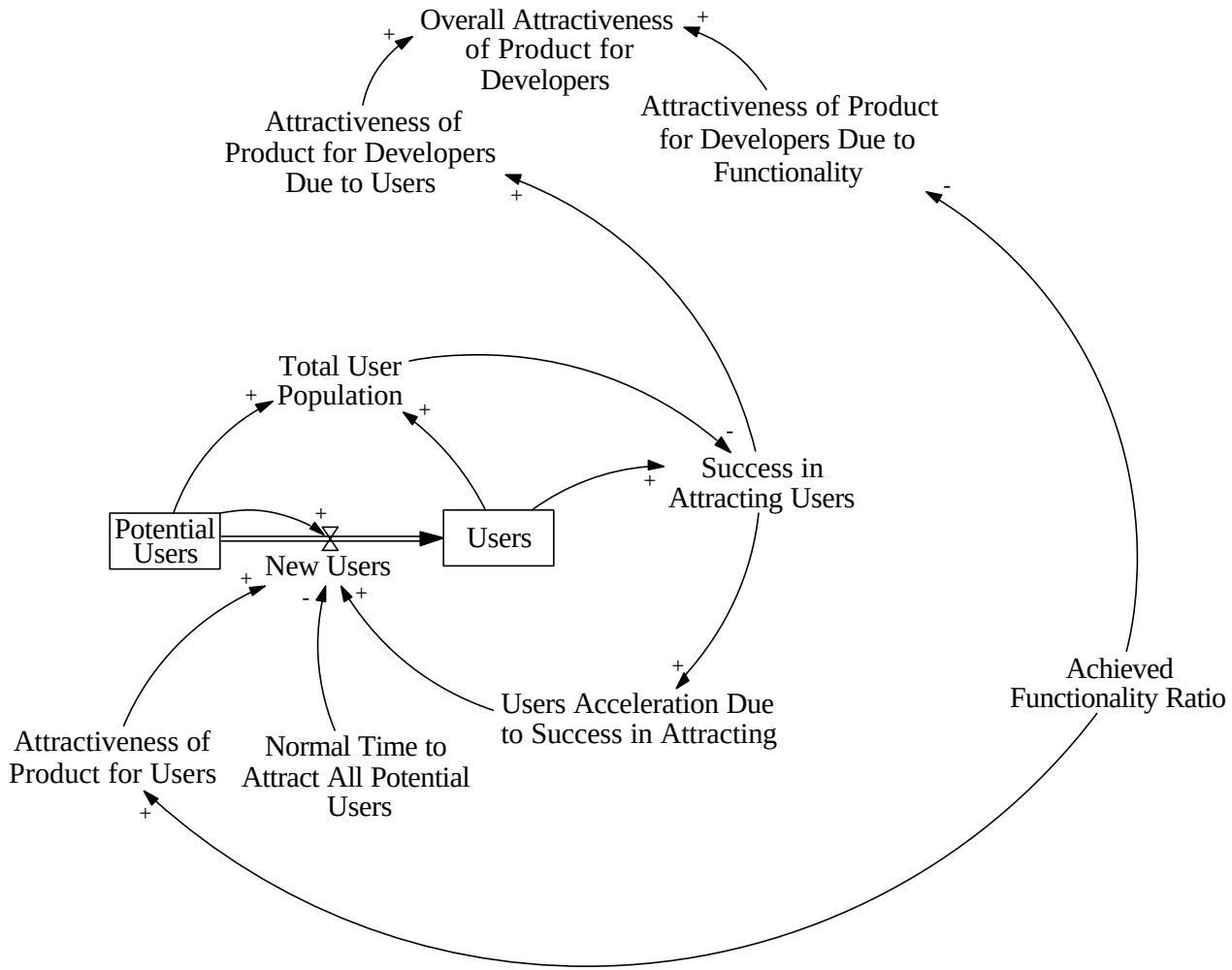


Figure 4.10. OSSD Model (Iteration I) Users Sector

The main feedback loops governing the Iteration I version of the model can be analyzed by putting the two sectors together, as shown in Figure 4.11. Besides several minor (two-variable) loops, the Iteration I version has five major loops that determine the overall model behavior. Three of these loops are balancing (negative) loops, while the other two are reinforcing (positive) loops.

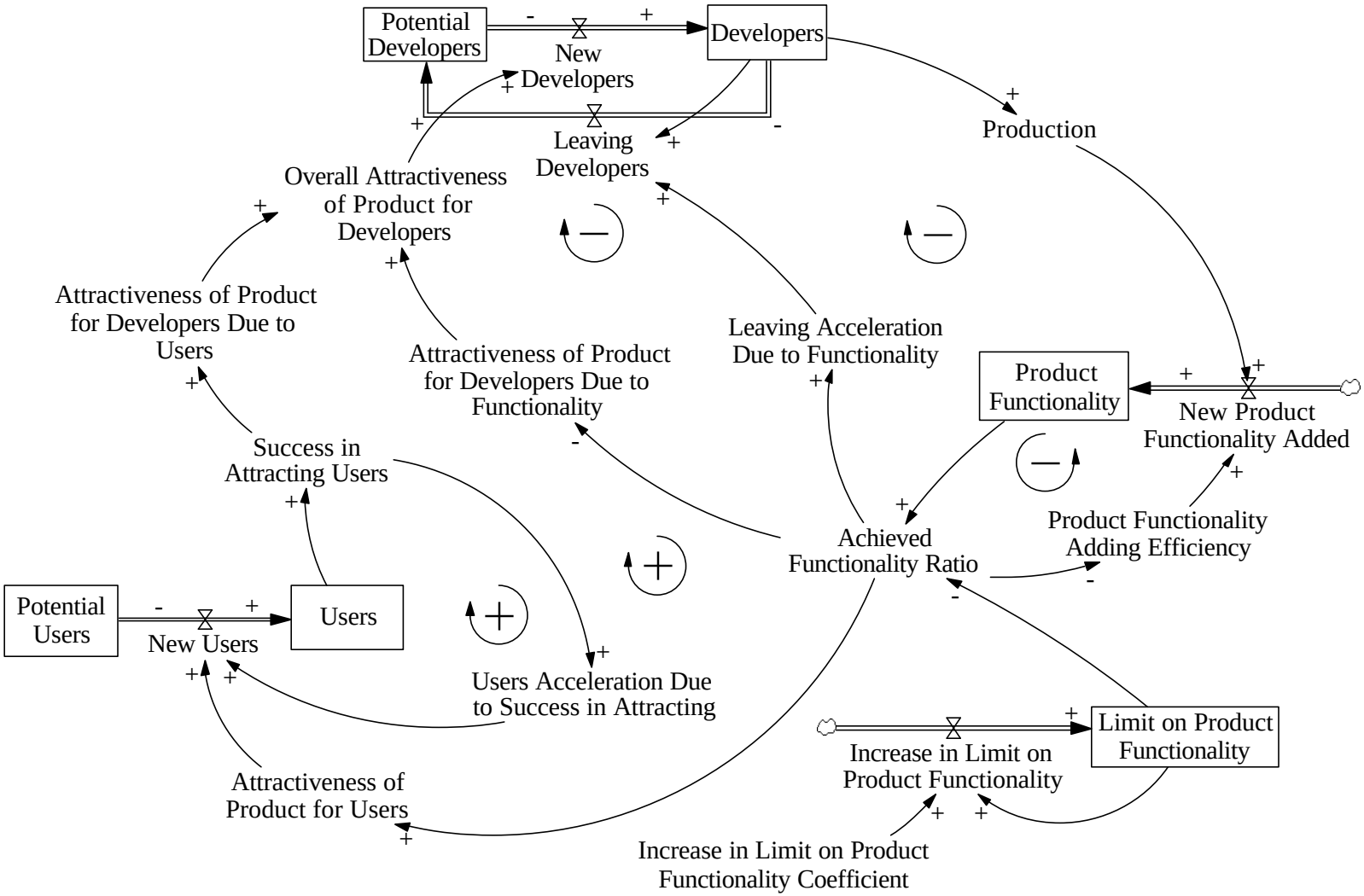


Figure 4.11. OSSD Model (Iteration I) Overview

The first balancing loop, as portrayed in Figure 4.12, limits the number of new developers that join the community, as product functionality increases and approaches the limit on product functionality. This is due to the decrease in the opportunities to contribute to the project, as discussed earlier in this section. As product functionality approaches the saturation point, potential developer see that there are not enough opportunities to claim a certain portion of the project. Thus, they refrain from joining the community, diverting their attention to alternative open source communities, where they can find more opportunities to "homestead" portions of the project. As the number of new developers decline, the developer pool first starts to grow at a slower rate, and after a point starts to decline. That tipping point is when the number of leaving developers becomes larger than the number of new developers.

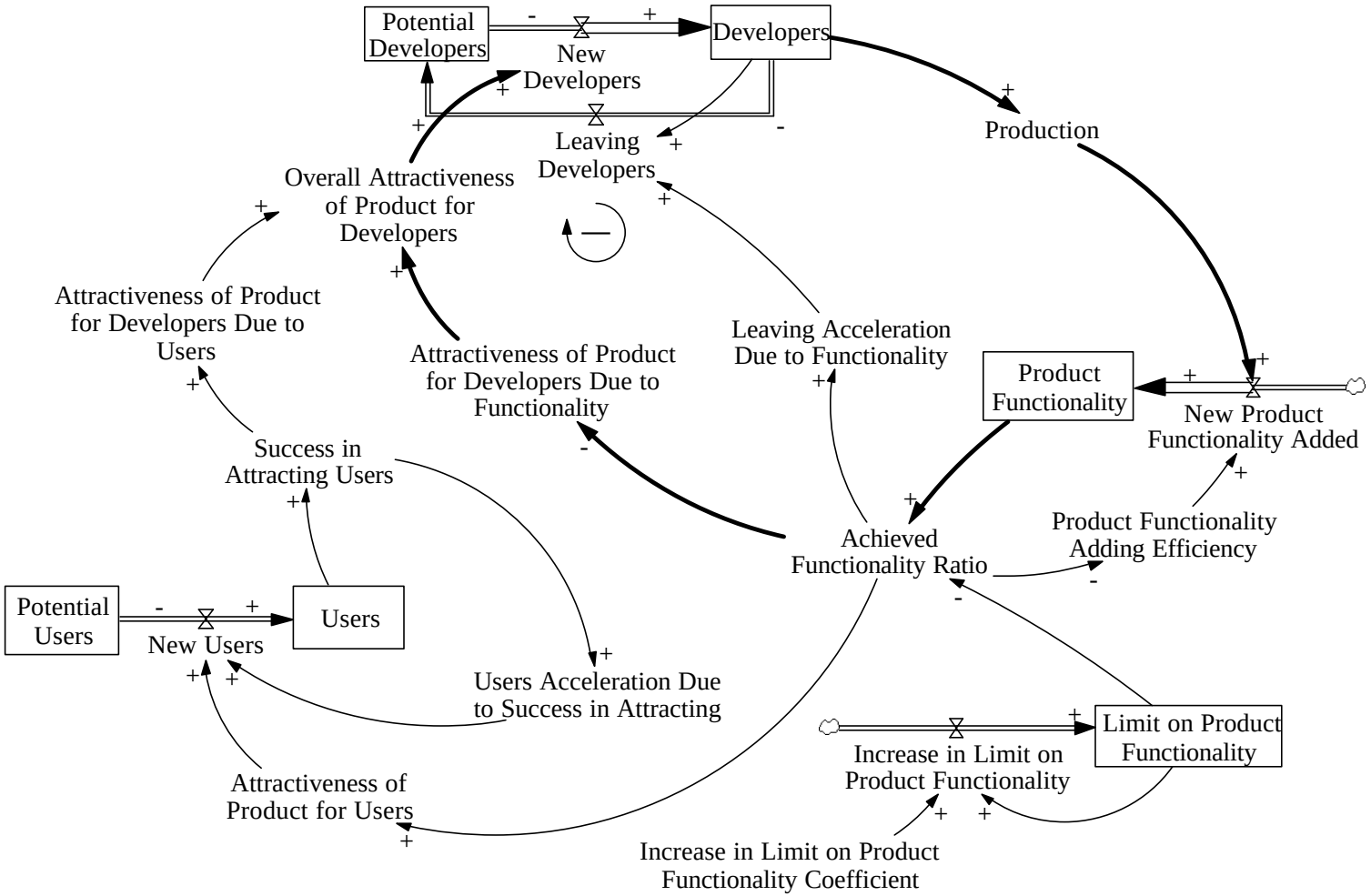


Figure 4.12. OSSD Model (Iteration I) Balancing Loop 1: “Fewer Opportunities for Contribution Attract Fewer New Developers.”

The second negative feedback loop limits the growth of the developer pool due to the increase in product functionality. (See Figure 4.13). However, it works through leaving developers, rather than new developers. When developers finish their portions of the project they tend to leave and move on to other projects, unless they stay within the community to maintain the product. Towards the end of the project, when product functionality approaches the limit on product functionality, many developers have done their share, so the number of leaving developers increases substantially. The increase in leaving developers, coupled with the decrease in the number of new developers, causes the developer pool first to grow more slowly and then to decline, as the tipping point discussed above is reached.

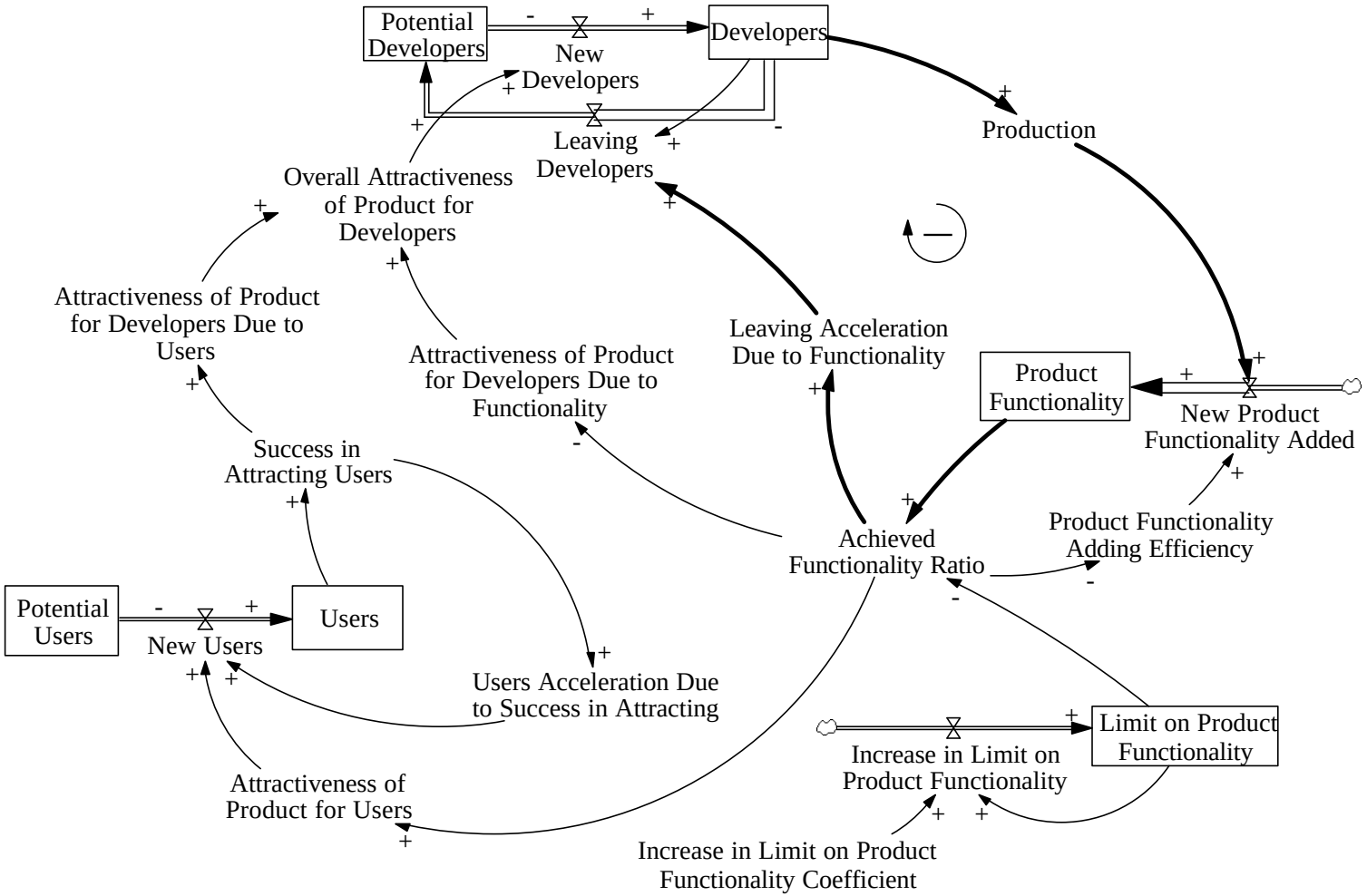


Figure 4.13. OSSD Model (Iteration I) Balancing Loop 2: "Fewer Opportunities for Contribution Retain Fewer Existing Developers."

The third balancing loop is the one that limits the new product functionality added per line of code produced. As the product functionality approaches the limit on product functionality, it becomes harder to add a marginal unit of functionality to the product. Accordingly, the same number of lines of code yields less functionality, as the achieved functionality ratio increases (See Figure 4.14).

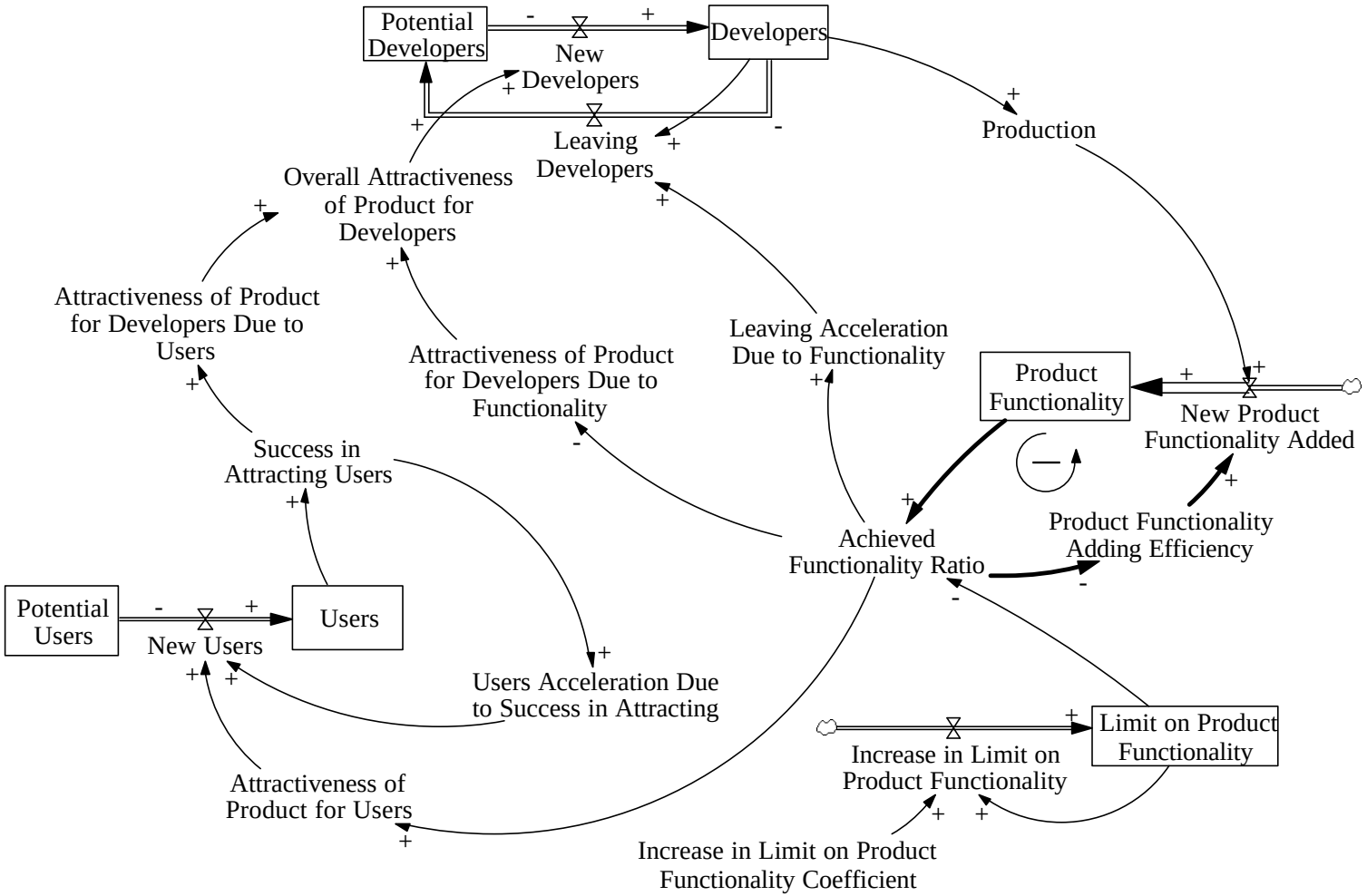


Figure 4.14. OSSD Model (Iteration I) Balancing Loop 3: “More Functionality Makes It Harder to Add Further Functionality.”

The first major reinforcing loop is local to the users sector (See Figure 4.15). This loop works according to the positive network externalities principle. As new users join the community by starting to use the product, the number of users increases. A higher number of users is perceived as a higher success in attracting users, and the higher success accelerates the rate of new users joining the community.

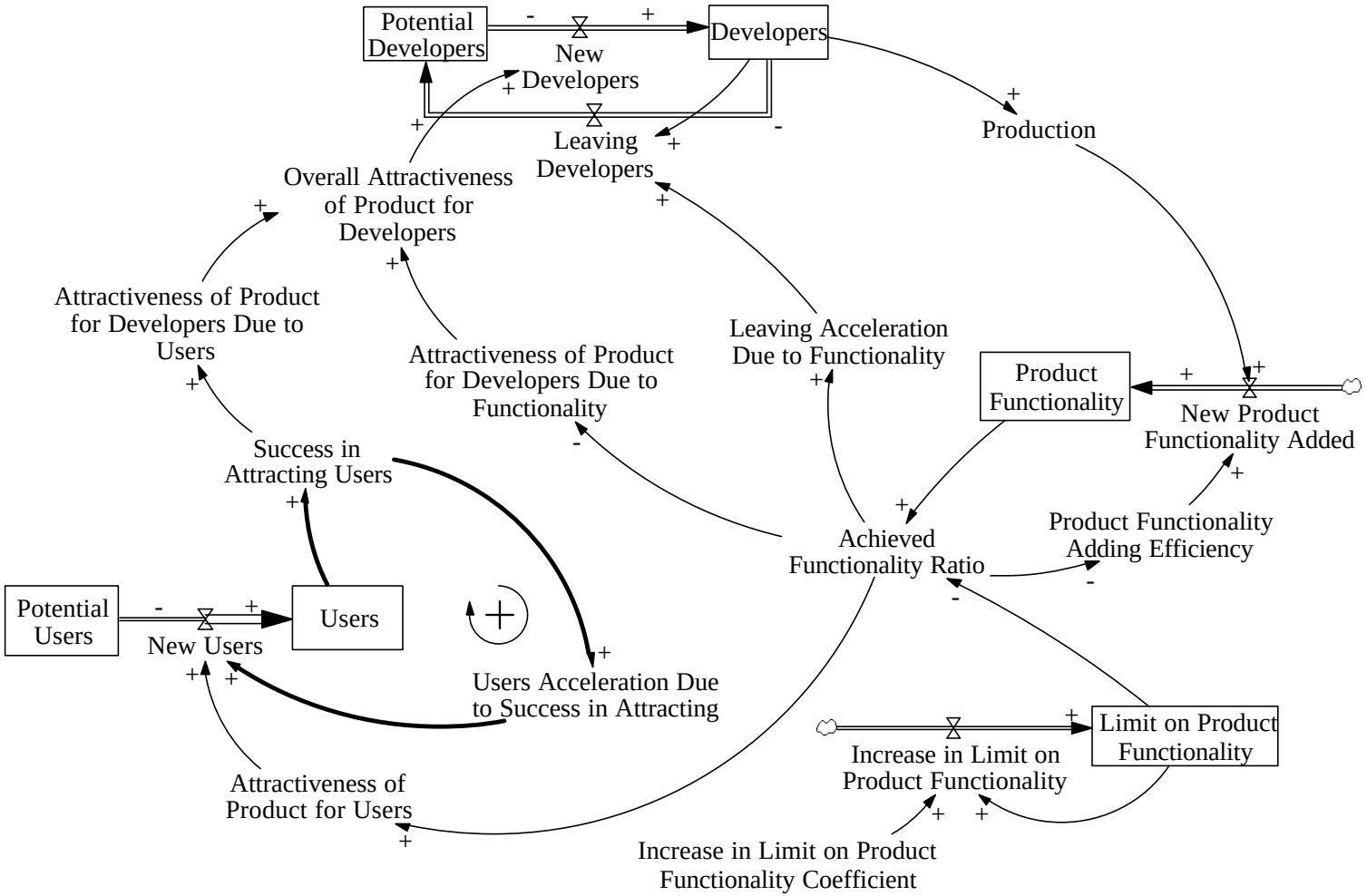


Figure 4.15. OSSD Model (Iteration I) Reinforcing Loop 1: “Positive Network Externalities Effect Attracts More Users.”

The second reinforcing loop is a model-wide one within the boundary of the Iteration I version. (See Figure 4.16.) This loop ultimately explains how a given community succeeds or fails in terms of overall growth. As developers participate in production and build product functionality, the achieved functionality ratio increases. A high functionality achievement attracts a higher number of new users, thus increasing the user pool rapidly. This is perceived as a success in attracting users. A considerable success in attracting users attracts more new developers, who in turn generate more production which helps build functionality faster. On the other hand, if existing developers fail to build functionality comparable with the increase in the limit on product functionality, the product fails to attract the critical level of users. That in turn decreases the attractiveness of the product for developers, decreasing the number of new developers, which would further decelerate the progress of the project. This loop is not as dominant in the Iteration I version as it is in the subsequent versions of the model. The reason for that is the exclusion of the time pressure factor in the Iteration I version. Time pressure is added to the model in the Iteration II version, which increases the effect of this reinforcing loop on the model behavior.

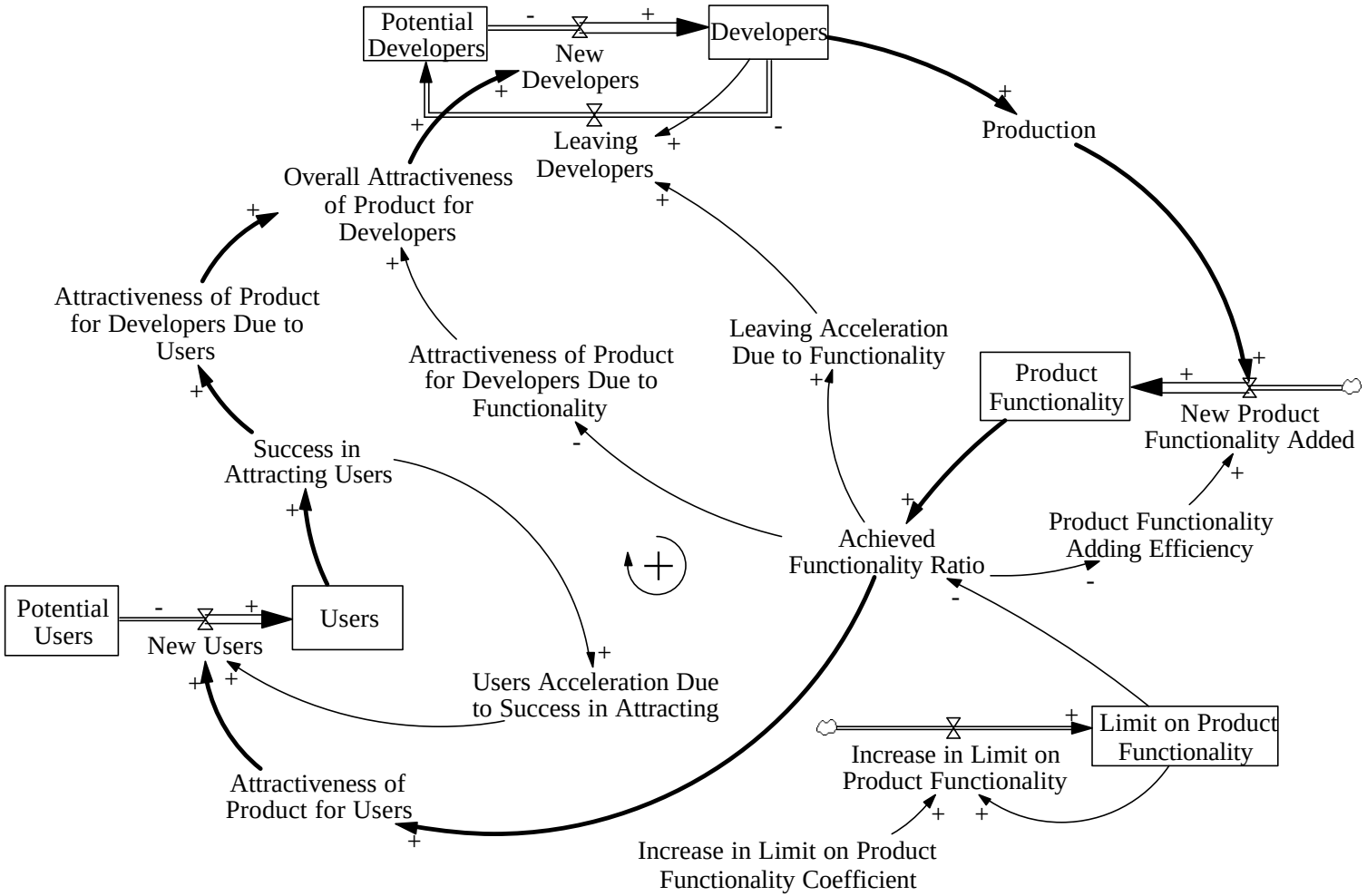


Figure 4.16. OSSD Model (Iteration I) Reinforcing Loop 2: “More Functionality Attracts More New Users, and That Attracts More New Developers.”

The base run of the Iteration I version involves a project with an initial product functionality limit of 400 Units of Functionality (UF). Figure 4.17 shows the behavior of product functionality for this base run. Product functionality increases almost in a linear fashion, seeking to reach the functionality limit after about month 55. After that point the rate of increase in product functionality drops since most of the potential functionality has been added to the product. Functionality limit, too, increases, as the general level of technology grows. However the increase in functionality limit is slower than that in product functionality.

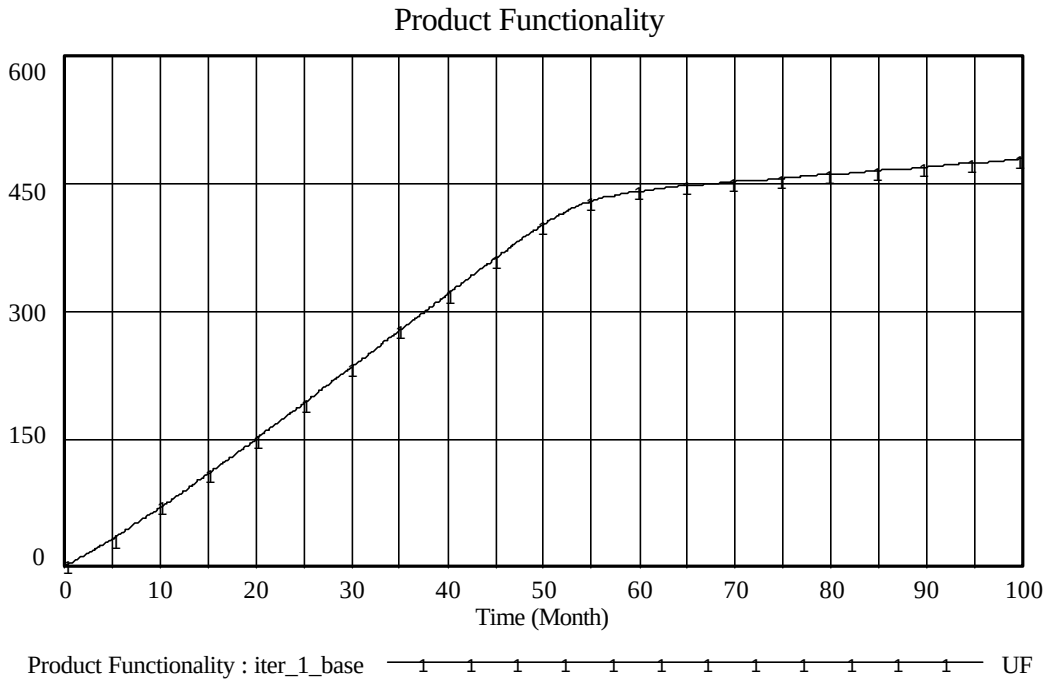


Figure 4.17. OSSD Model (Iteration I) Base Run - Product Functionality

Figure 4.18 displays the behavior of achieved functionality ratio, which in fact is the ratio between actual product functionality achieved and functionality limit. Here, achieved product functionality increases in a linear fashion until it reaches an equilibrium value a little below 1. After that point, achieved product functionality does not increase

any further due to the lag between the increase in the general level of technology and the actual maintenance improvements in the product in question.

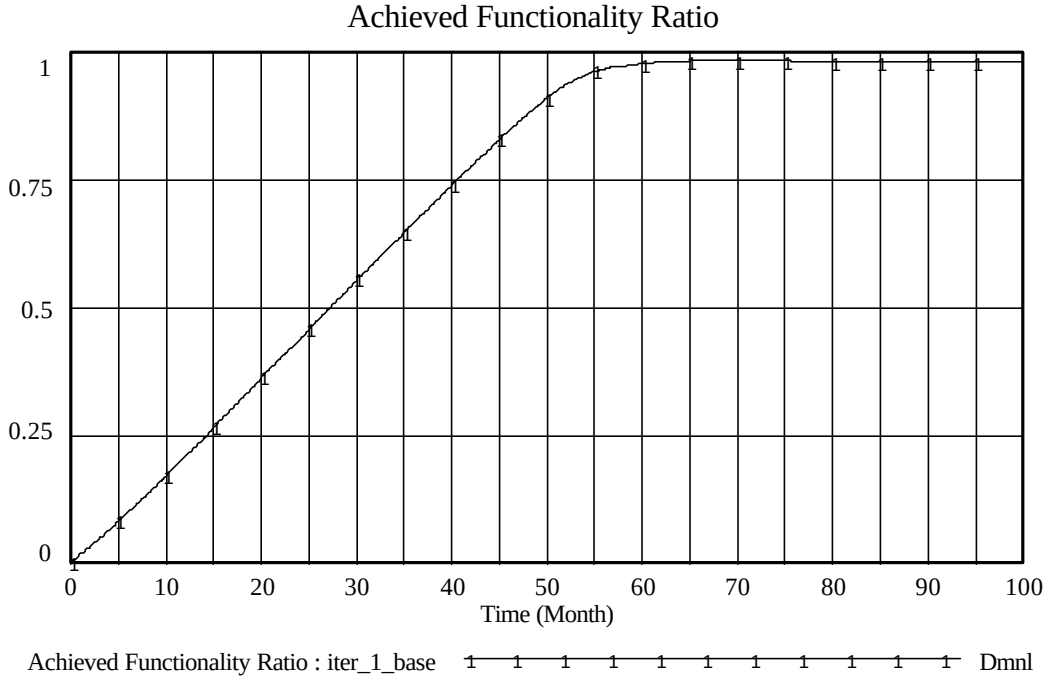


Figure 4.18. OSSD Model (Iteration I) Base Run - Achieved Functionality Ratio

The behavior of the number of developers in the base run for the Iteration I version is shown in Figure 4.19. The number of developers increases as the project unfolds because the overall attractiveness of the product keeps the rate of new developers above the rate of leaving developers. At around month 43, the rate of leaving developers surpasses the number new developers. This is caused by (a) decreases in attractiveness due to decreasing opportunities for making contributions and (b) the acceleration of developer departures due to the fact that many developers have completed their contribution to the product at that stage of the project. After that point the number of developers continues to decline until an equilibrium just below 10 is reached. These are

the developers that stay in the community for maintenance and updating purposes, in an effort to keep the product current with respect to the general level of technology.

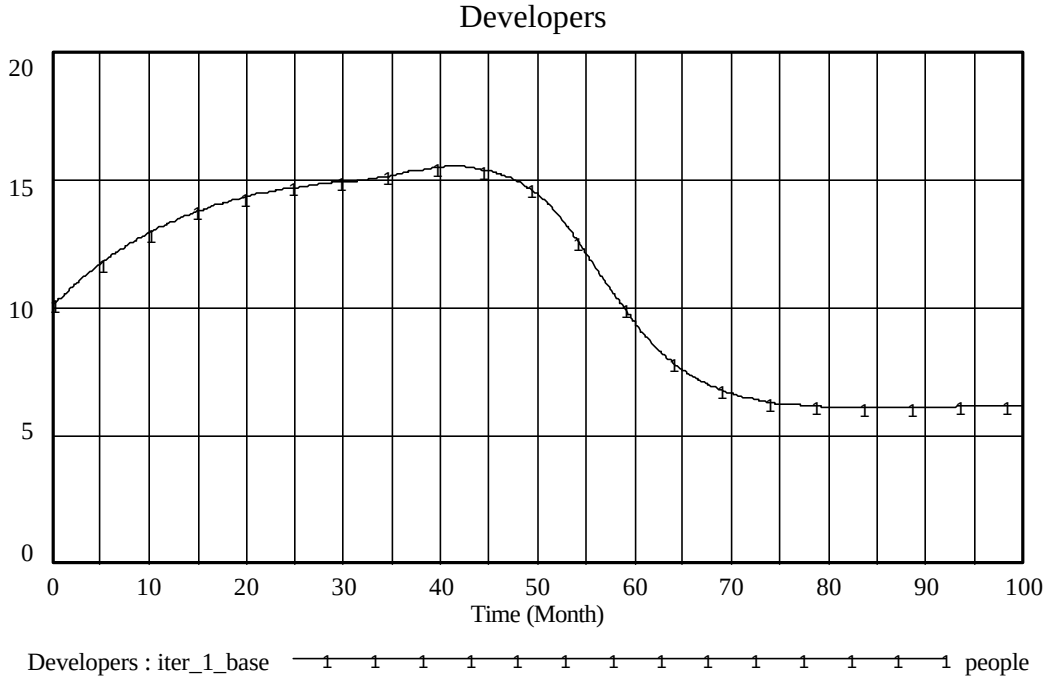


Figure 4.19. OSSD Model (Iteration I) Base Run - Number of Developers

The number of users of the product exhibits an S-shaped growth pattern as shown in Figure 4.20. The growth of the number of users is driven by the achieved functionality ratio through the attractiveness of the product for users, and the success in attracting users through positive network externalities. As the achieved functionality ratio and success in attracting users increase, the rate of new users increases faster and the number of users exhibit an exponential growth pattern until around month 33. After that point, the increase changes shape and becomes sub-linear because the pool of potential users becomes too small. Finally, the number of users converges to the absolute number of potential users at 20,000 people. This, of course, is based on the assumption that there are

a fixed number of potential users that would be interested in a given product and that that number would not increase over time.

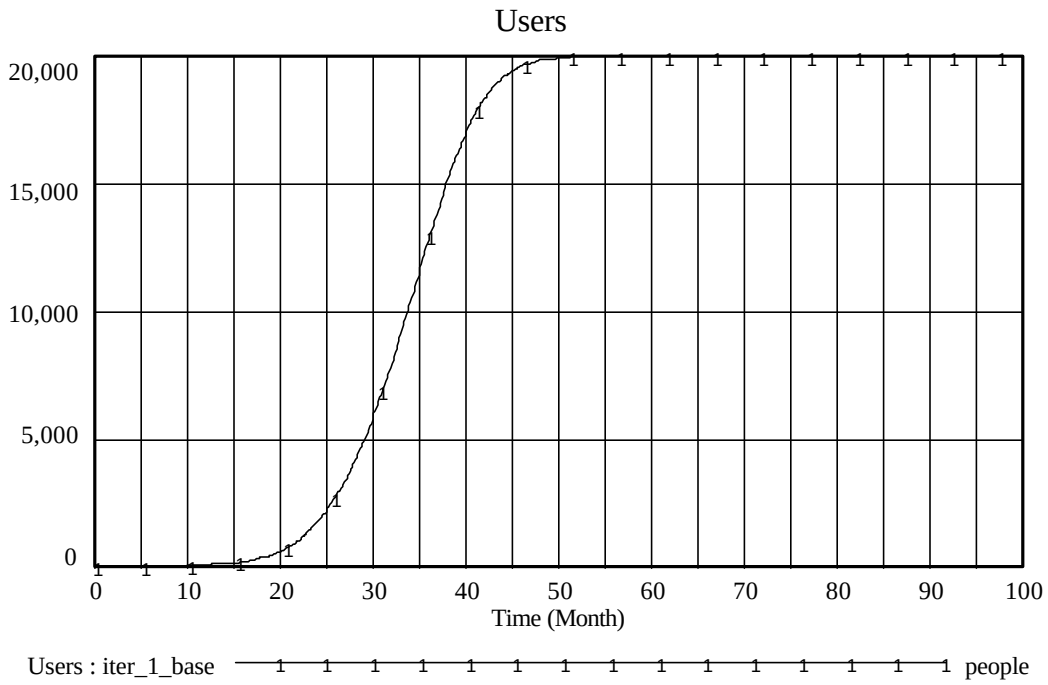


Figure 4.20. OSSD Model (Iteration I) Base Run - Number of Users

Another simulation is run with the initial limit on product functionality set to 4000 UF, and the potential user population set to 200,000 people. As Figure 4.21 shows, product functionality exhibits a behavior that is very close to linear. This behavior covers roughly 80% of the development period of the product. As Figure 4.22 shows, achieved productivity ratio reaches a little higher than 0.8 by the end of month 100. The number of developers increases until month 80, since there is still a considerable amount of functionality to be added until that stage in the project. After month 80, the number of developers starts to decrease (See Figure 4.23). The number of users reaches the saturation point around month 95 (See Figure 4.24).

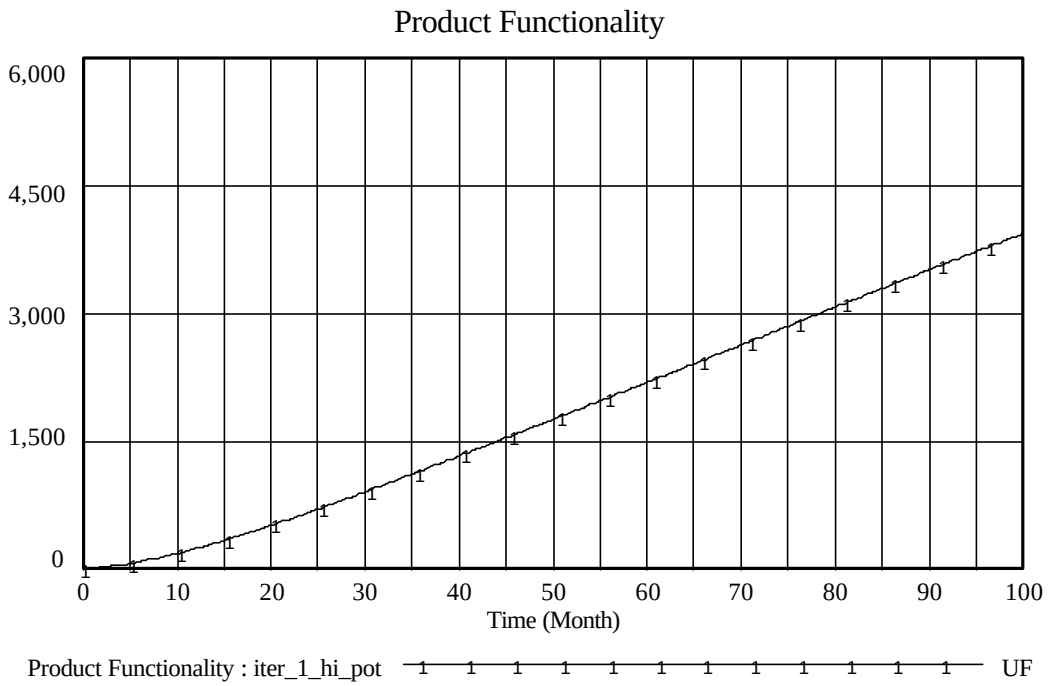


Figure 4.21. OSSD Model (Iteration I) High Functionality Potential Run - Product Functionality

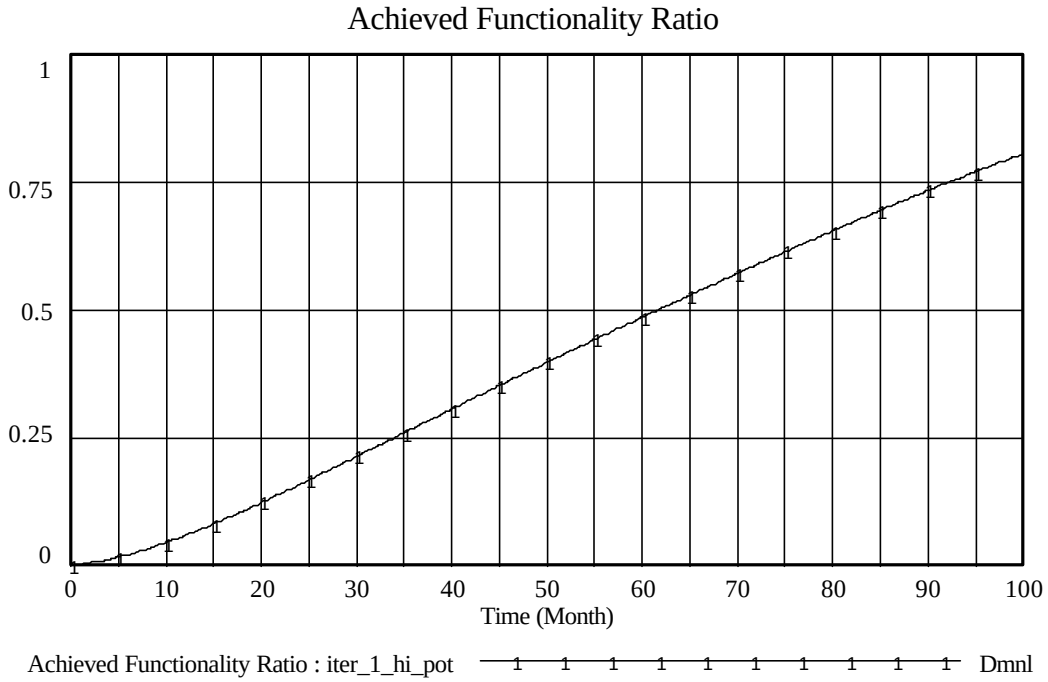


Figure 4.22. OSSD Model (Iteration I) High Functionality Potential Run - Achieved Functionality Ratio

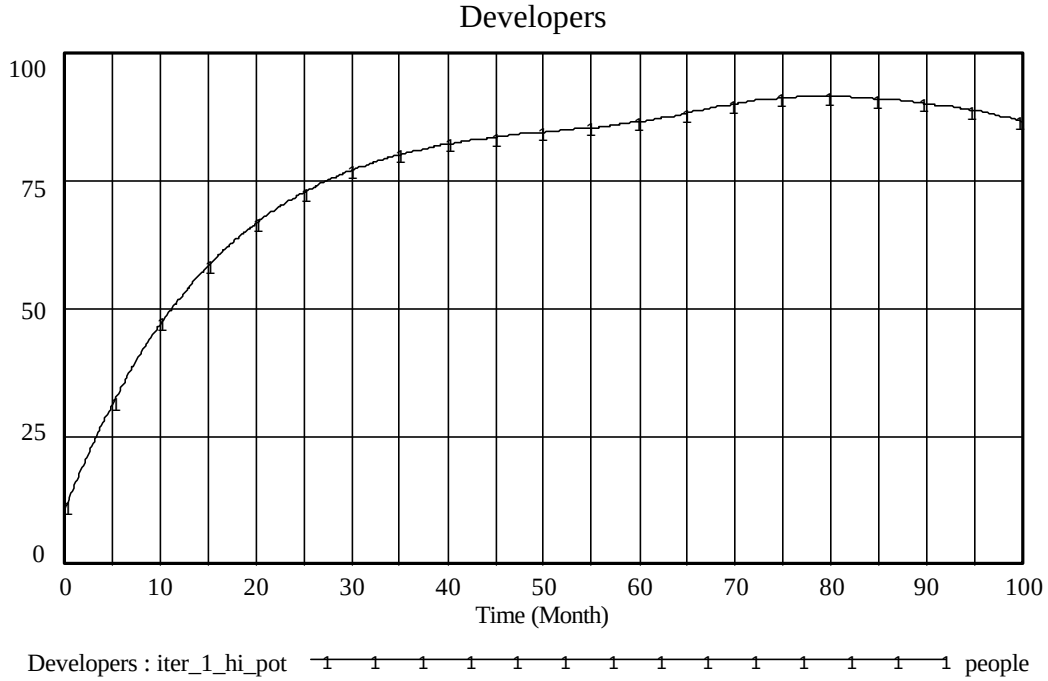


Figure 4.23. OSSD Model (Iteration I) High Functionality Potential Run - Developers

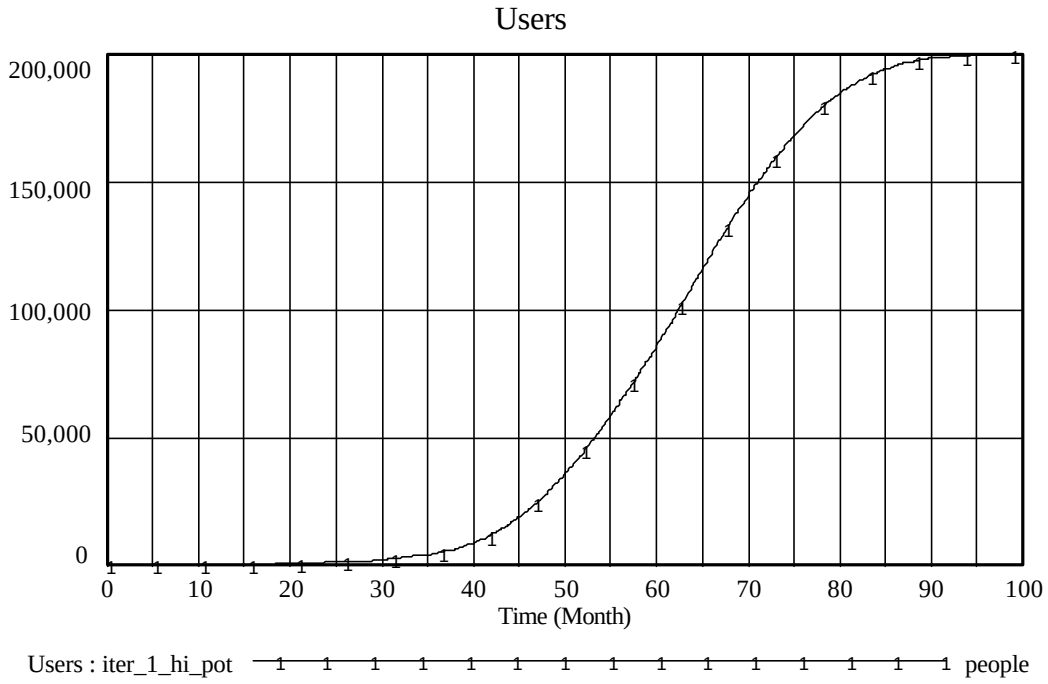


Figure 4.24. OSSD Model (Iteration I) High Functionality Potential Run - Users

A range of simulation runs with the Iteration I version under different initial conditions and parameter settings points out the importance of time pressure, which is addressed by the Iteration II version of the model. An example is a group of simulations run by setting average participation to lower values than the original value of 20 hours per person per month. Figure 4.25 through Figure 4.28 shows the behavior of the Iteration I version with Average Participation set arbitrarily to seven hours per person per month, as a lower participation level. While the growth of product functionality and the number of users slow down considerably, the community still succeeds in terms of retaining a critical mass of developers that continue to work on the product. Eventually, both product functionality and the number of users reach healthy levels. When average participation is decreased even further, the growth slows down even more; however, given enough time, product functionality and the number of users always reach healthy levels. This is a critical problem about the Iteration I version. The Iteration I version can replicate the behavior of successful communities, but not those of unsuccessful communities. Changing other parameters that have a decreasing effect on the overall production triggers the same problems. For example, decreasing average productivity, decreasing the normal (base) rate of new developers, or increasing the normal rate of leaving developers all directly or indirectly decrease overall production. As production decreases, the growth of the product and the community slow down, however the community never fails to reach a healthy level in terms of product functionality and the number of users, given enough time. This problem is addressed by the Iteration II version of the model, which includes the time pressure factor and replicates a wider range of situations more accurately.

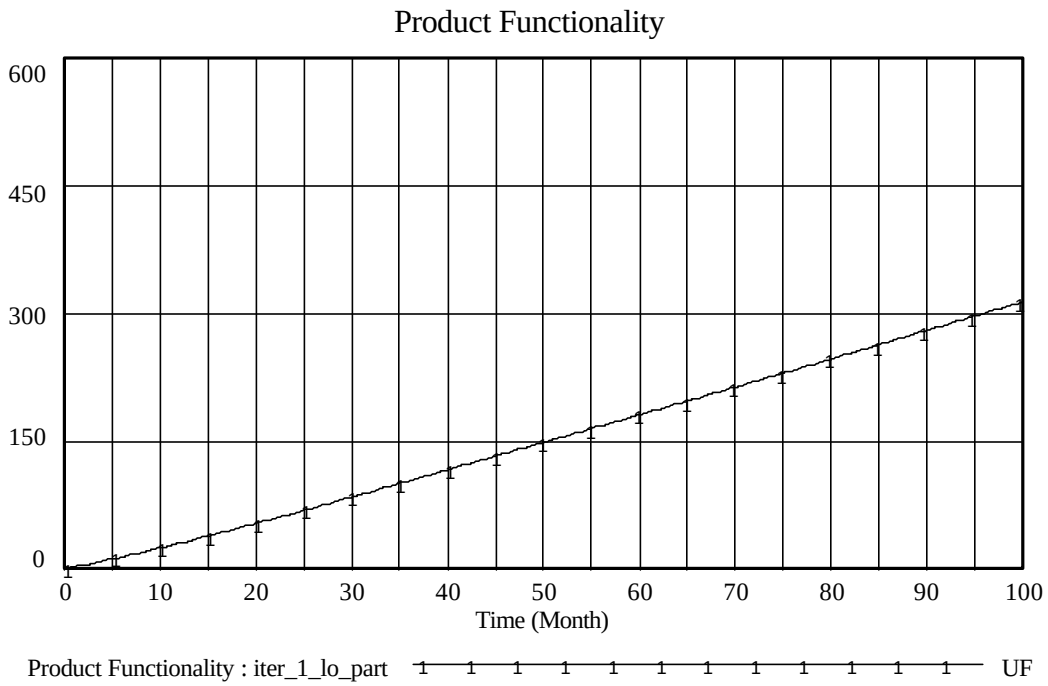


Figure 4.25. OSSD Model (Iteration I) Low Participation Run - Product Functionality

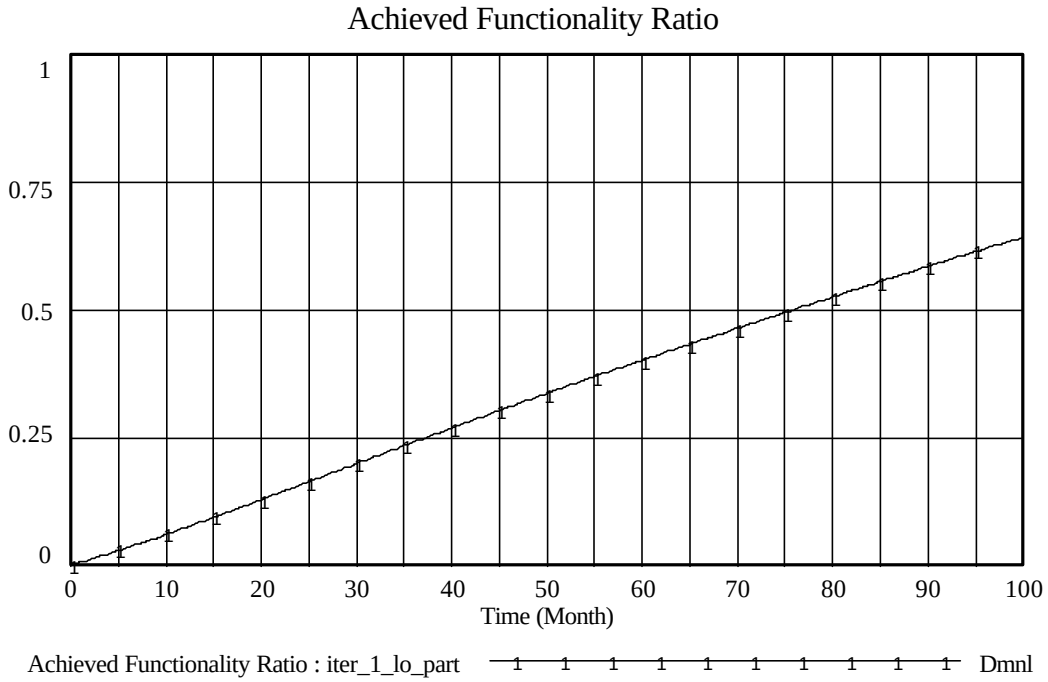


Figure 4.26. OSSD Model (Iteration I) Low Participation Run - Achieved Functionality Ratio

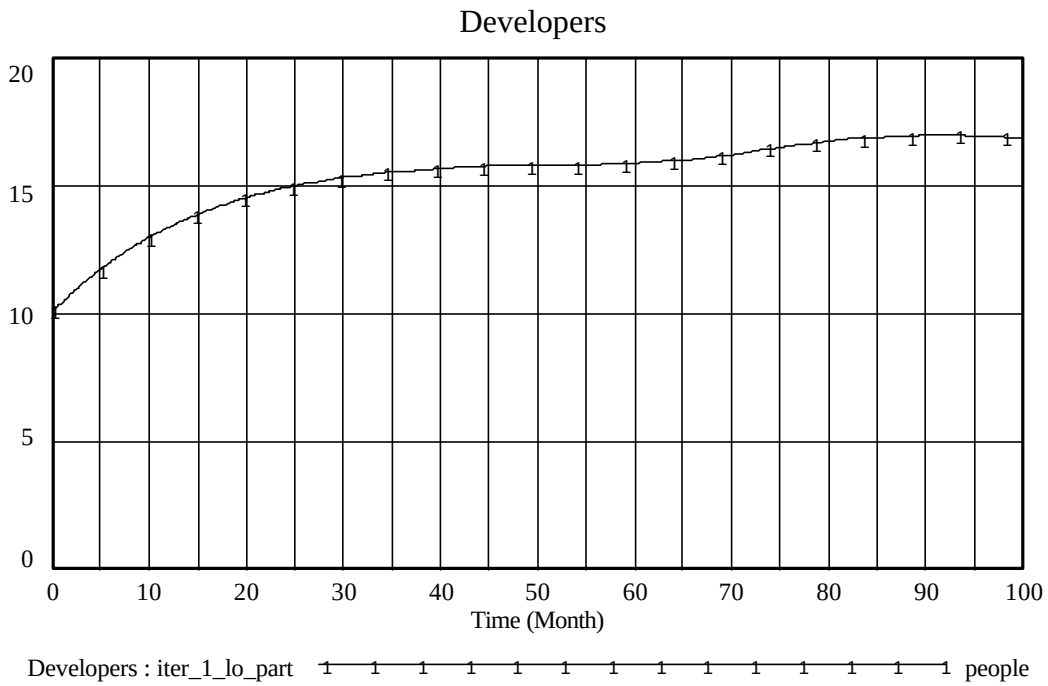


Figure 4.27. OSSD Model (Iteration I) Low Participation Run - Developers

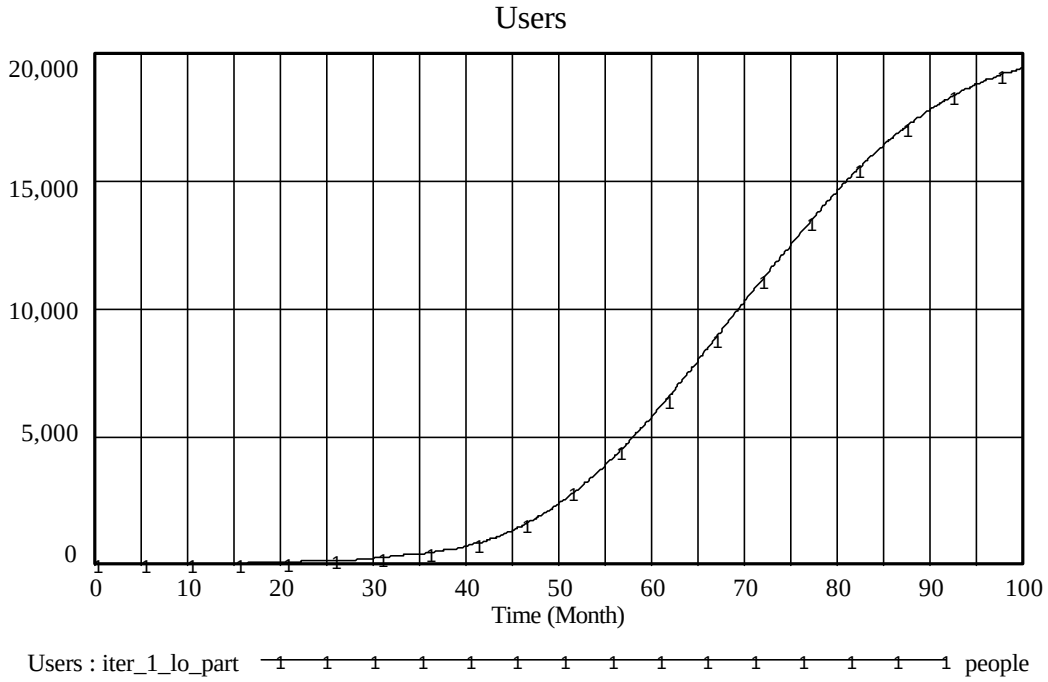


Figure 4.28. OSSD Model (Iteration I) Low Participation Run - Users

4.3. Iteration II: Adding Time Pressure

The aim of the second iteration of the model-building phase is to capture the time pressure factor on the community while developing a product. The Iteration I version of the model cannot explain cases where a community ceases to grow and eventually declines because the product is not delivered in a timely fashion. Under the assumption that there are other proprietary and open source alternatives for the product being developed by the community, it is crucial to deliver the product within the time frame expected by the users.

As portrayed in Figure 4.29, the Iteration II model assumes the existence of a general level of patience on the part of the potential members of the community, both developers and users. The initial limit on product functionality determines the speed with which that patience will run out, and how fast the community will expect the product to mature.. It is assumed that a larger product in terms of the limit on product functionality will bring about a slower rate at which patience runs out. In other words, the community will expect a bigger project to mature over a longer period of time, so they will lose patience more slowly.

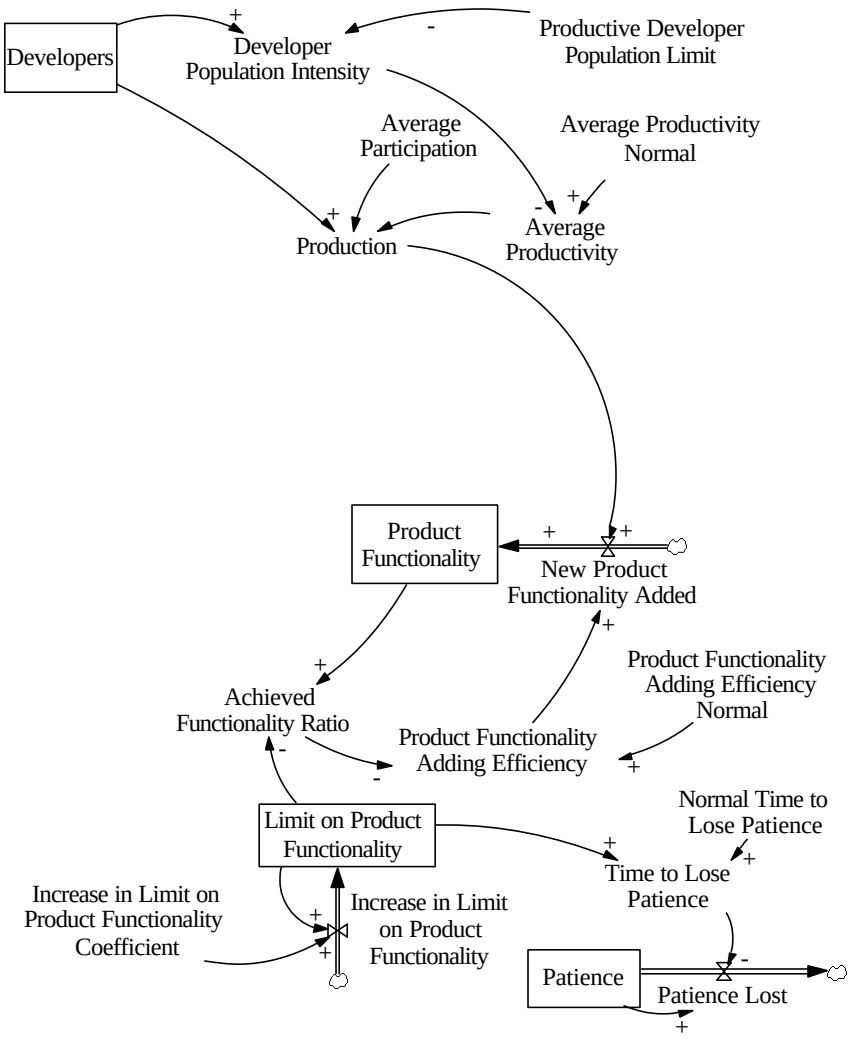


Figure 4.29. OSSD Model (Iteration II) Developers Sector

The level of patience at a given time determines the functionality expectation of the community (See Figure 4.30). The expected functionality ratio would constitute a mental benchmark for community members when they assess the success of the project in terms of delivering functionality in a timely manner. It is assumed that during the initial phases of a project, community members would not focus too much on the actual level of achieved functionality, and give the project a chance even if the achieved functionality ratio is very low. Rather, they would focus on their expectations for a period of time in the hope that the achieved functionality level would approach those expectations in time. As the project unfolds, their focus would shift toward achieved functionality ratio. This shift in the focus for assessment is represented by the operative functionality ratio. Operative functionality ratio is a weighted average of achieved and expected functionality ratios (See Figure 4.30). The weights are determined by the expected functionality ratio. As expectation builds, the weight shifts to the achieved functionality ratio. It is assumed that at the beginning of the project the weight on expected functionality ratio is 1, and it remains 1 until the expected functionality ratio reaches 0.1. From that point on the weight on expected functionality ratio declines, the weight on expected functionality ratio grows, and they both become 0.5 when expected functionality ratio reaches 0.2. By the time expected functionality ratio reaches 0.3, the weight on achieved functionality ratio reaches 1.

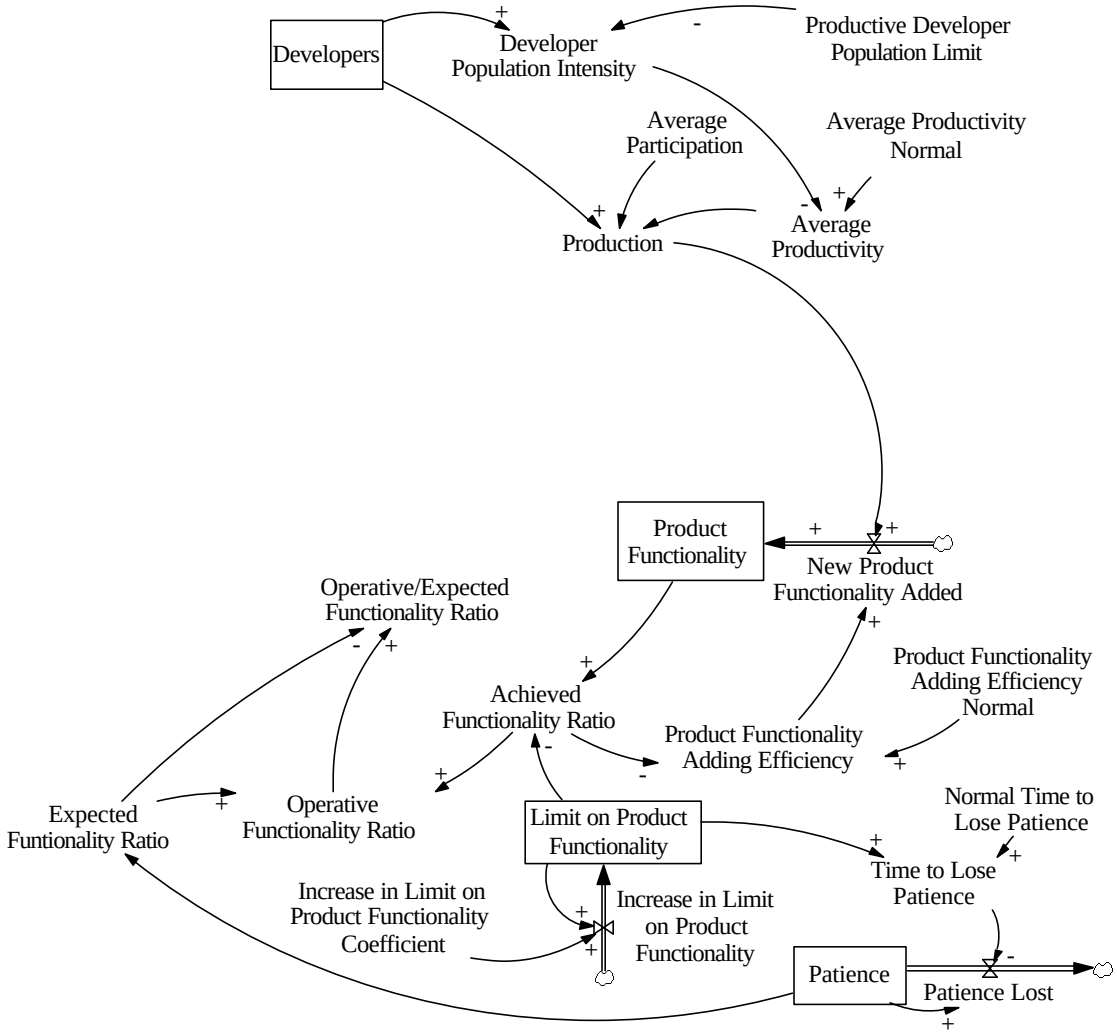


Figure 4.30. OSSD Model (Iteration II) Developers Sector

The success of the community in terms of accommodating the functionality expectation is represented as a ratio between the operative and expected functionality ratios. On the part of the developers, operative vs. expected functionality ratio has two motivational effects. On the positive side, a high operative vs. expected functionality ratio would increase the overall attractiveness of the product for developers, and thus increase the number of new developers joining the community (See Figure 4.31). On the negative side, a low operative vs. expected functionality ratio would discourage the existing developers, and increase the rate of leaving developers (See Figure 4.32).

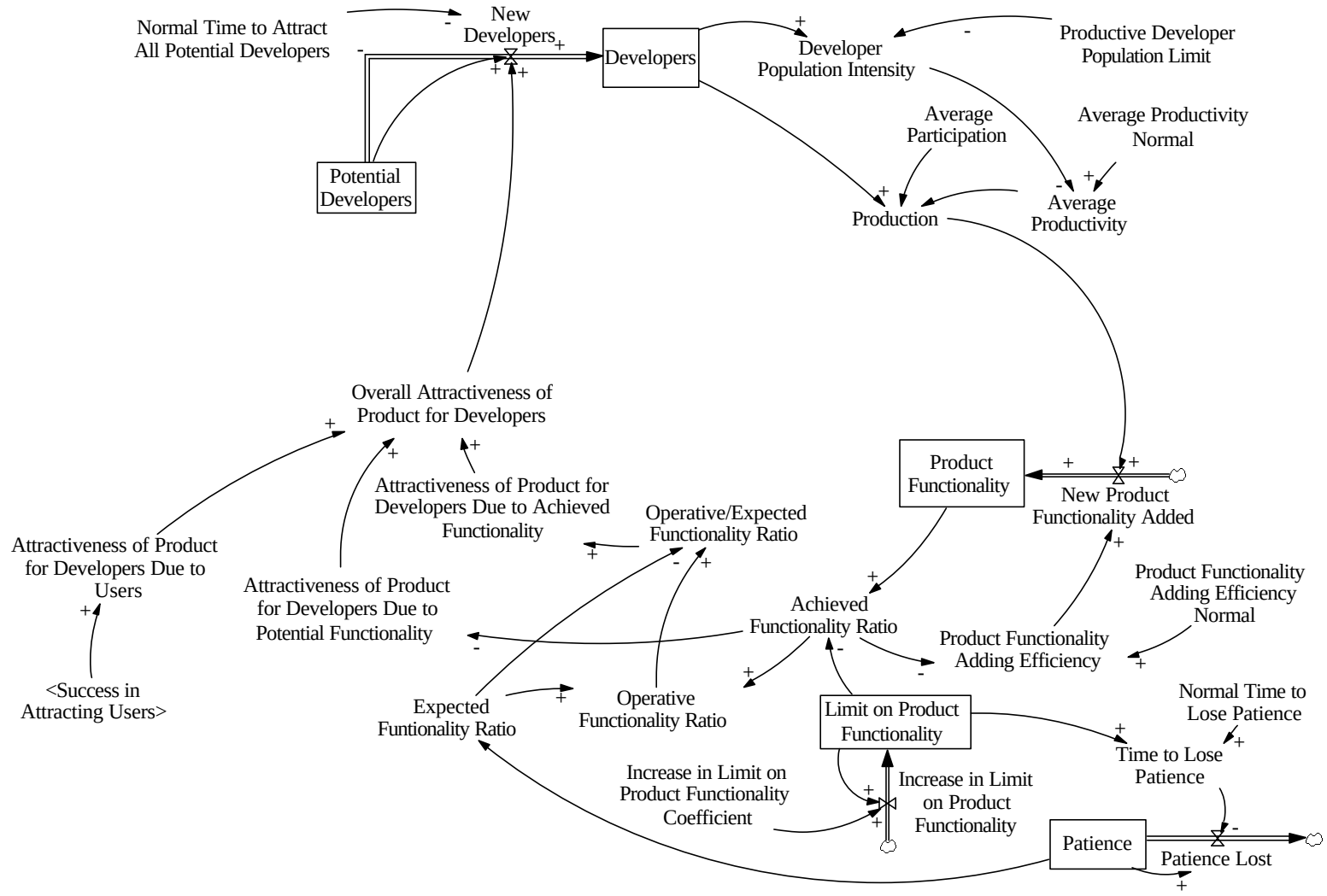


Figure 4.31. OSSD Model (Iteration II) Developers Sector

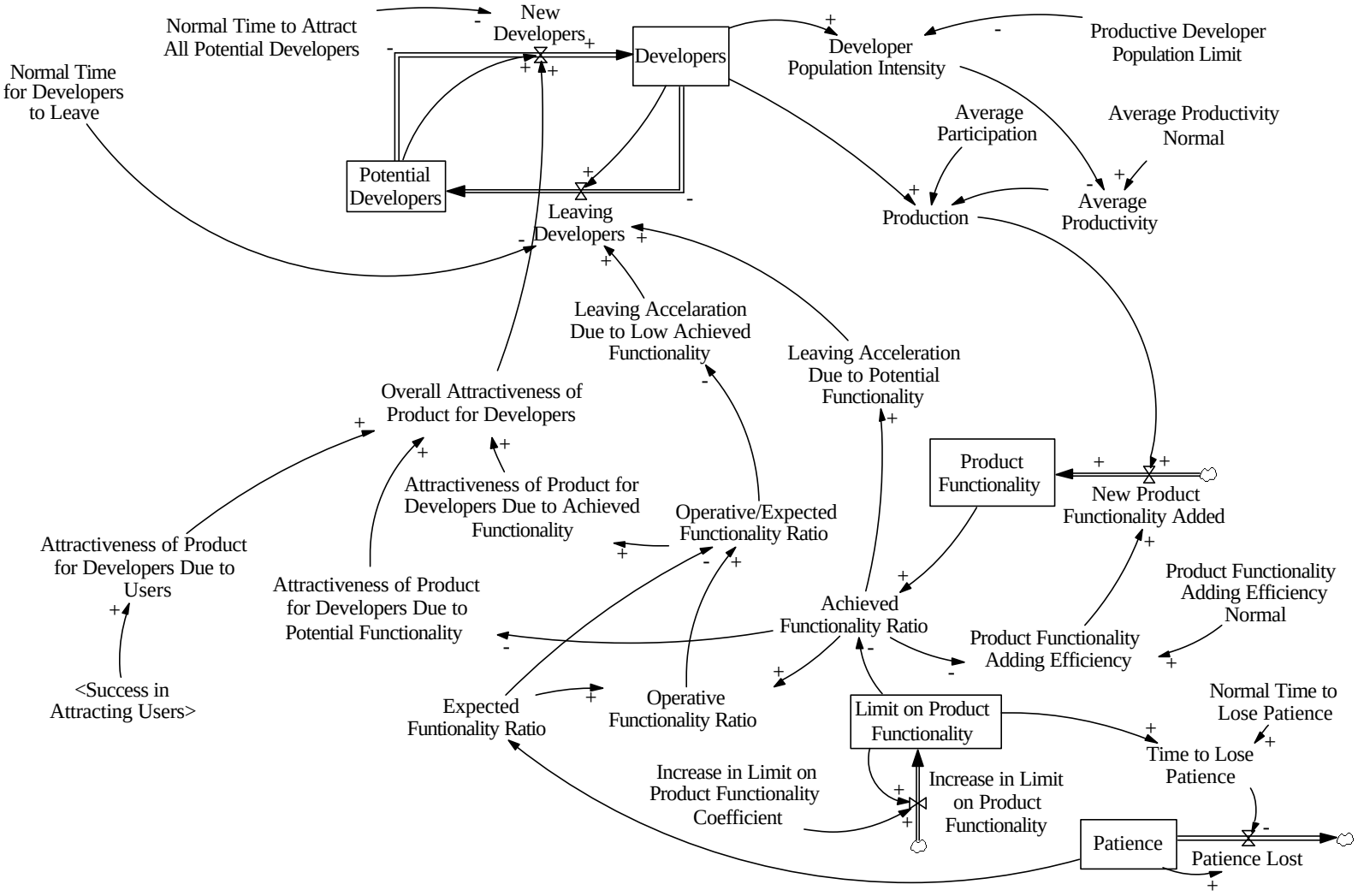


Figure 4.32. OSSD Model (Iteration II) Developers Sector

Another concept that is introduced with the Iteration II version is the pool of developers working on similar projects. These projects are rivals to the community in the sense that they focus on developing similar, alternative products. As shown in Figure 4.33 developers would join and leave the other projects with certain rates, thus adding to and taking from the pool of potential developers. This is a more accurate representation of the competition for developer resources as it happens in open source software development.

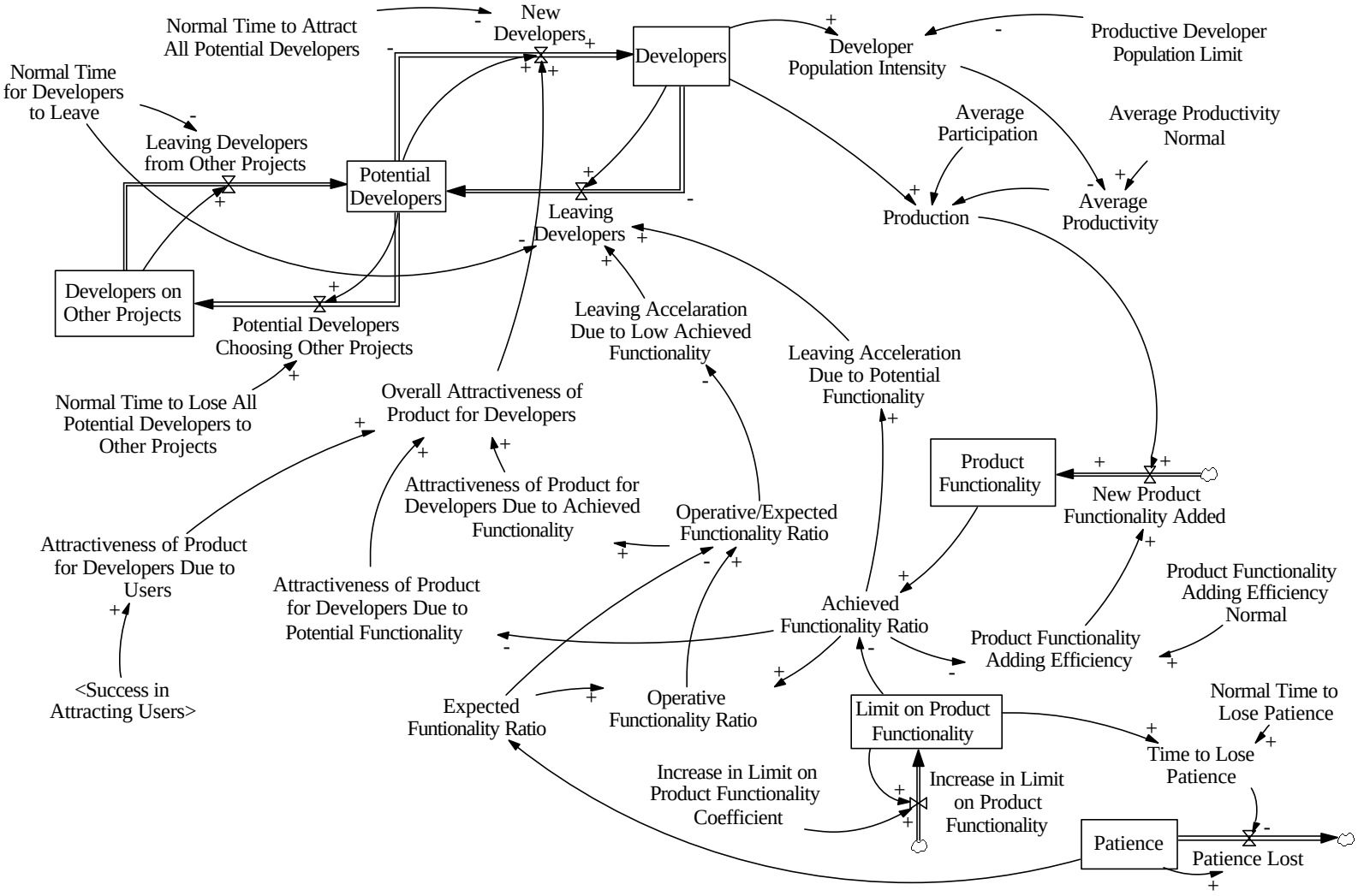


Figure 4.33. OSSD Model (Iteration II) Developers Sector

Leaving users is another concept that is added to the Iteration II version. Operative/Expected Functionality Ratio affects the rate of leaving users. It is assumed that the users would leave the users pool at a certain rate, which is accelerated by low levels of operative/expected functionality ratio. (See Figure 4.34.) It is important to note that while new users are attracted to the product based on the absolute level of achieved functionality ratio, leaving users are influenced by the achieved/expected functionality ratio. The assumption here is new users do not pay attention to how the functionality of the product has increased over time while they are deciding whether to shift to the product. They only look at the absolute functionality level at the time they are making their decision, and base their decision on that.

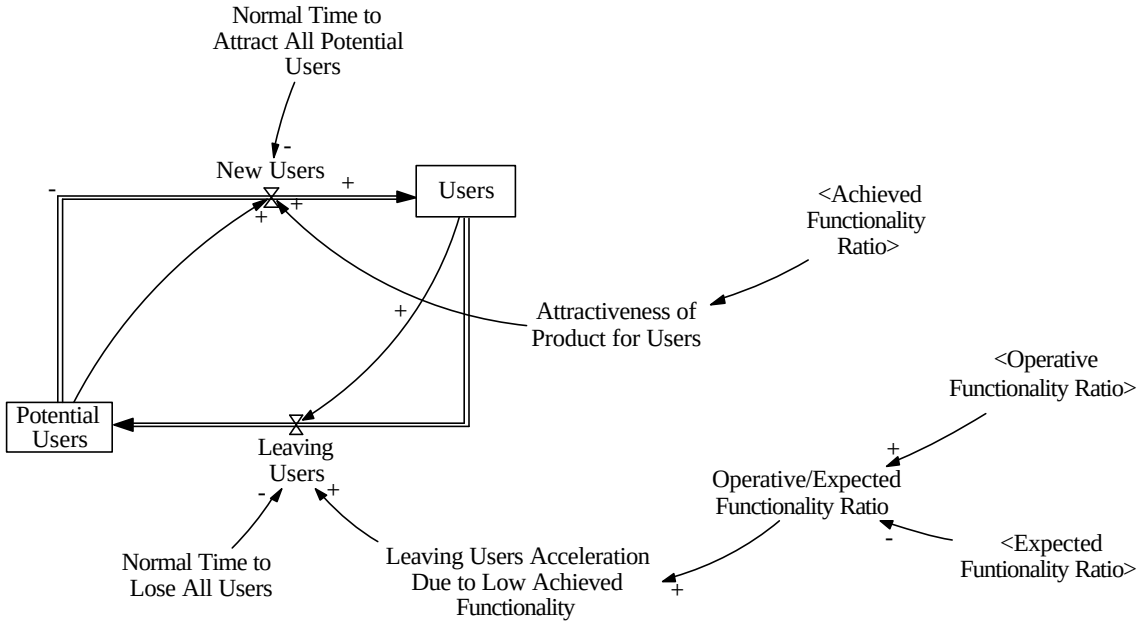


Figure 4.34. OSSD Model (Iteration II) Users Sector

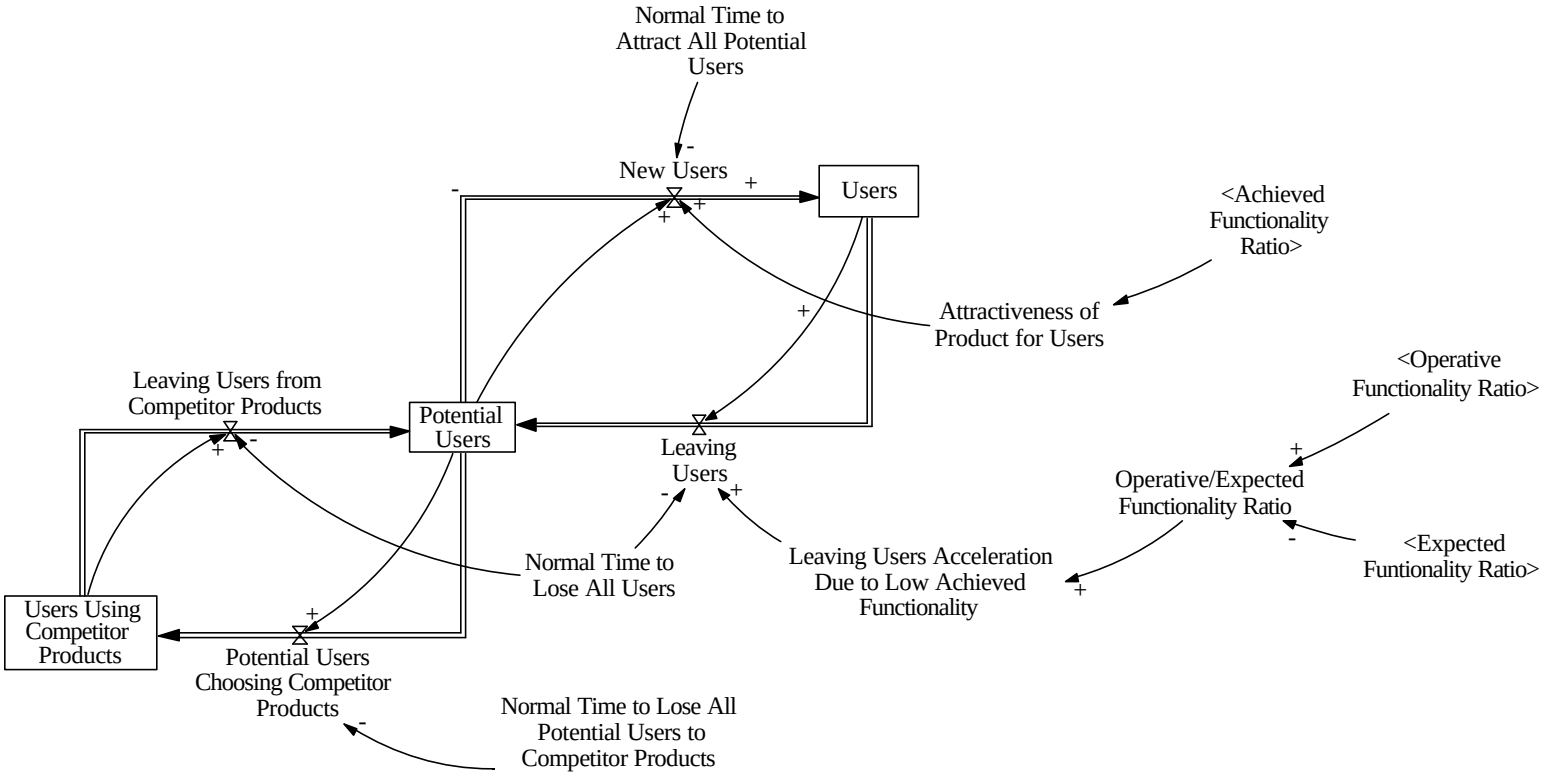


Figure 4.35. OSSD Model (Iteration II) Users Sector

On the other hand, existing users' expectations for functionality grow as the project unfolds, and if achieved functionality does not match their expectations at a given time, they may become impatient and quit using the product. The option of users switching to competing products is also added to the model with the Iteration II version. Potential users may choose to adopt competing products and existing users of competing products may adopt the open source option at certain rates, as shown in Figure 4.35.

As shown in Figure 4.36, success in attracting users is still determined by the ratio between the number of users of the product and the number of total users. Number of total users includes the number of users of the product, number of users of competing products, and number of potential users, in Iteration II model. Success in attracting users influences the number of new users and the attractiveness of the product for developers positively, as in the Iteration I version (See Figure 4.36.)

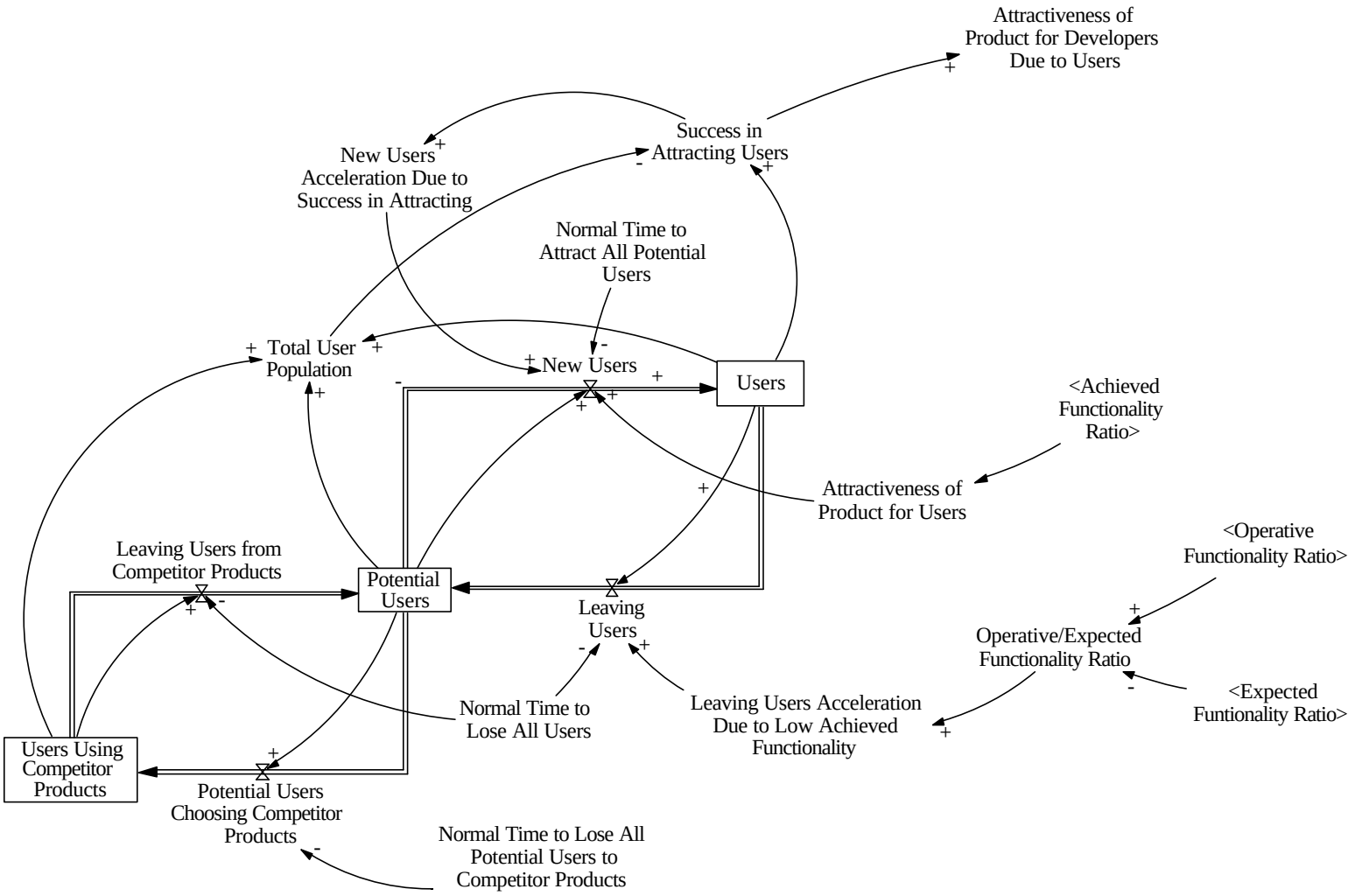


Figure 4.36. OSSD Model (Iteration II) Users Sector

Including time pressure in the model introduces three more major reinforcing loops. The first of the new reinforcing loops (Reinforcing Loop 3) works through the overall attractiveness of the product for developers. As developers participate in production and add functionality to the product operative/expected functionality ratio increases. A higher operative/expected functionality ratio increases the attractiveness of the product for the developers, thus the rate of new developers joining the project increases. (See Figure 4.37.)

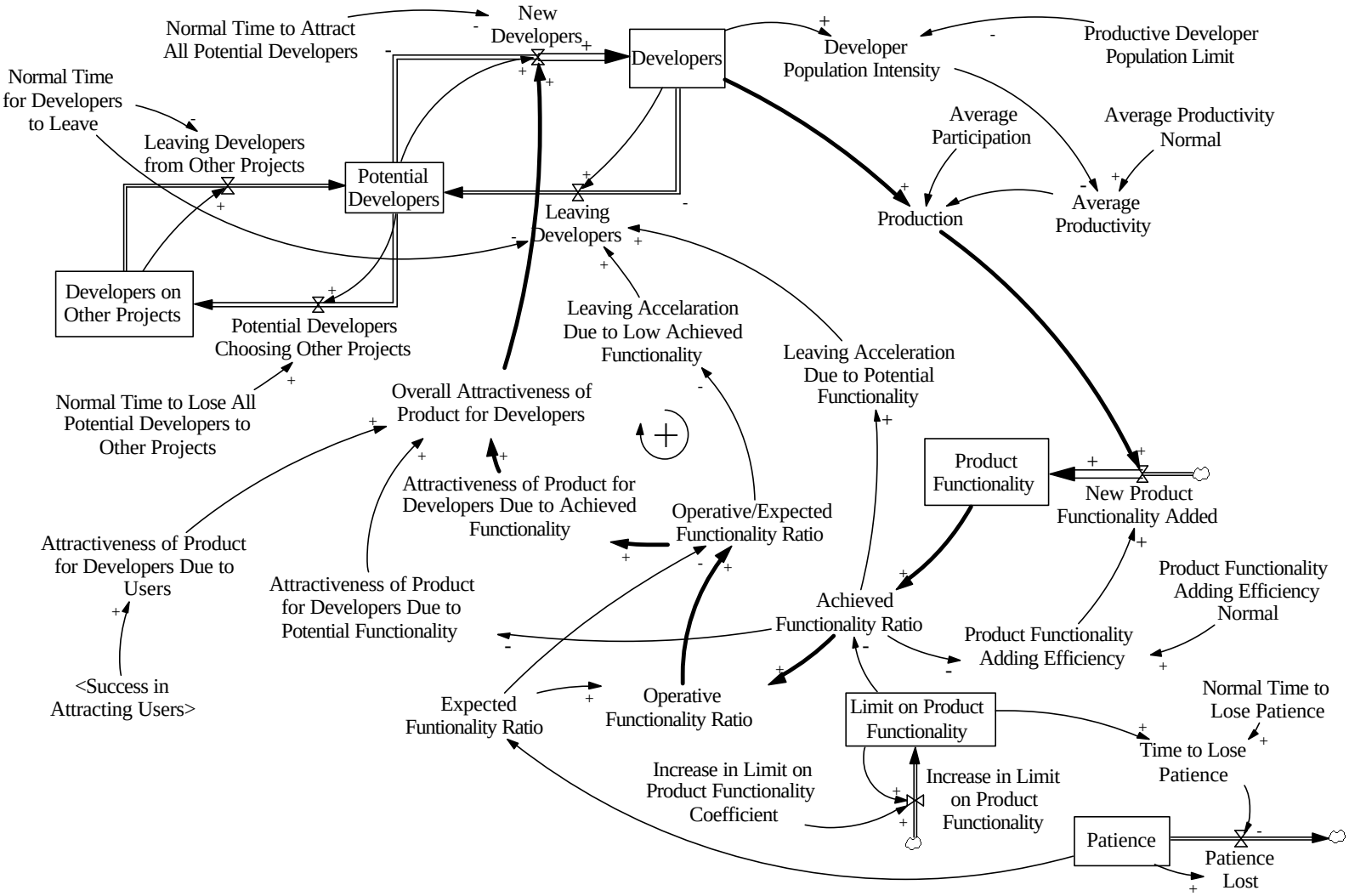


Figure 4.37. OSSD Model (Iteration II) Reinforcing Loop 3: “More Functionality Attracts More New Developers.”

The second newly introduce loop (Reinforcing Loop 4) works through the acceleration of leaving developers due to low achieved functionality. As the operative/expected functionality ratio decreases more developers are inclined to leave the project. This would slow down the growth of the developer pool if the rate of new developers is higher than the rate of leaving developers. If the rate of leaving developers is faster than the rate of new developers it would decrease the number of developers faster. This in turn would affect the production and functionality growth negatively. (See Figure 4.38.)

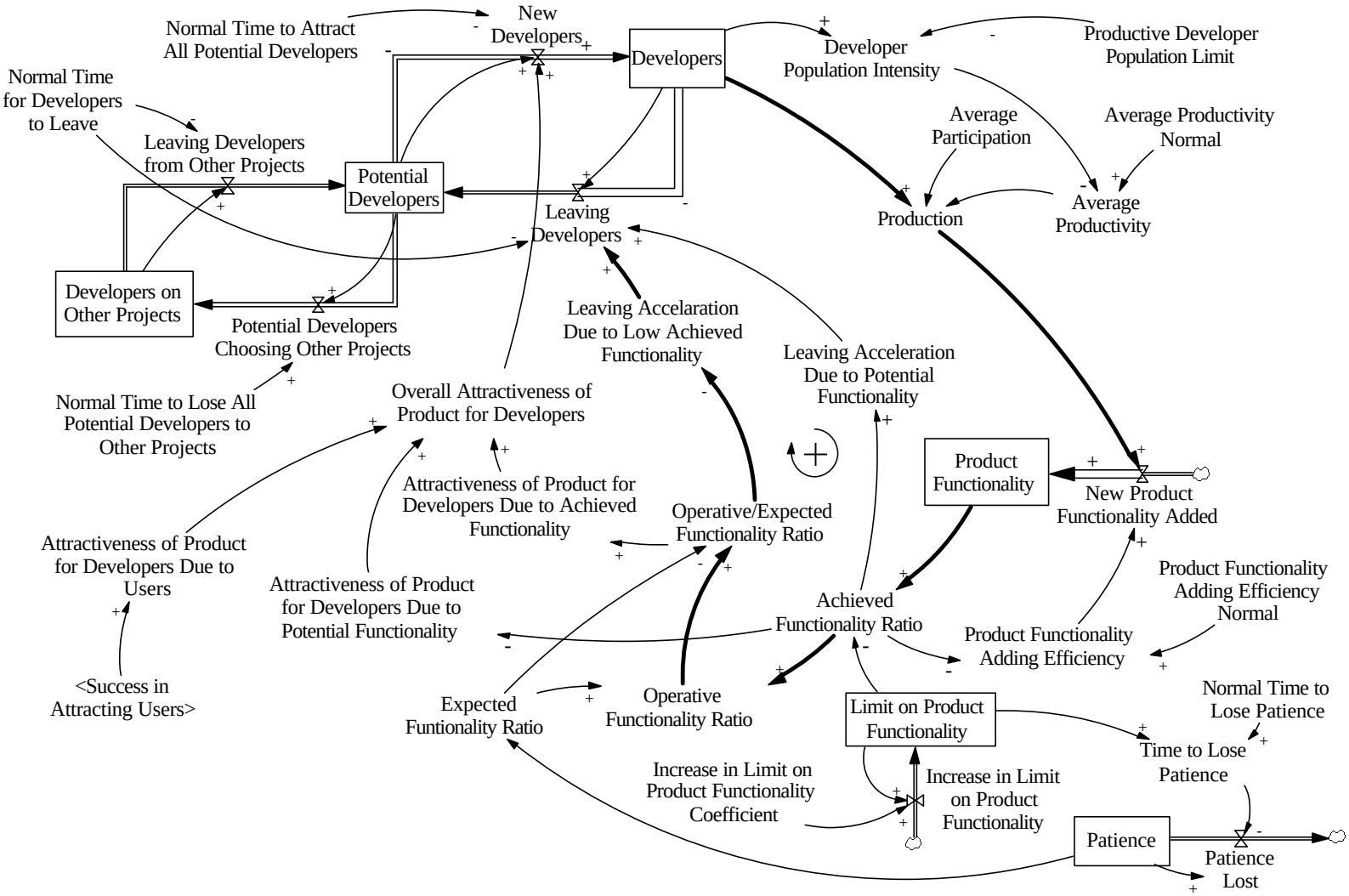


Figure 4.38. OSSD Model (Iteration II) Reinforcing Loop 4: “More Functionality Retains More Existing Developers.”

Reinforcing Loop 5 is the third one of the newly introduced reinforcing loops. (See Figure 4.39.) This loop works through the accelerating effect of low operative/expected ratio values on the leaving users. If operative/expected ratio falls below a certain level, more users would quit using the product in favor of a competing product. This would either decrease the number of users -- or at least keep it from increasing faster -- and ultimately have a negative effect on the rate of user adoption and consequently on the attractiveness of the product for developers, thus slowing down the rate of new developers.

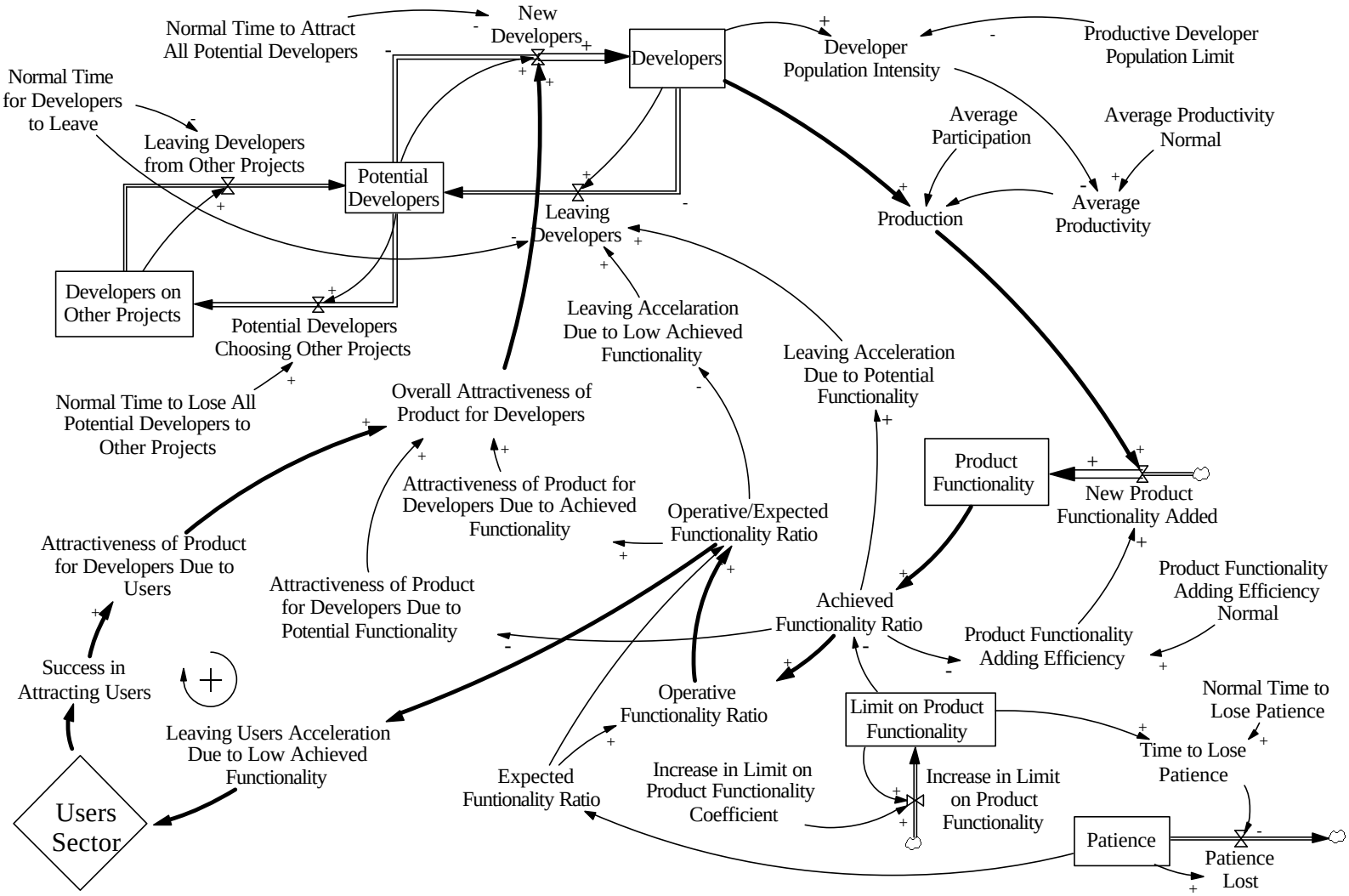


Figure 4.39. OSSD Model (Iteration II) Reinforcing Loop 5: “More Functionality Retains More Existing Users.”

The Iteration II version displayed a behavior that is very similarly to that of the Iteration I version in terms of the main indicators under the base run conditions. The base run is again based on a project with an initial product functionality limit of 400 Units of Functionality (UF). Here again product functionality increases almost linearly until it reaches about 97% of the limit on product functionality (See Figures 6.40 and 6.41). From there on, the rate of increase in product functionality drops, since a healthy level of achieved functionality ratio is reached. As can be observed in Figure 4.41, achieved functionality begins to decrease after reaching a peak around month 65.

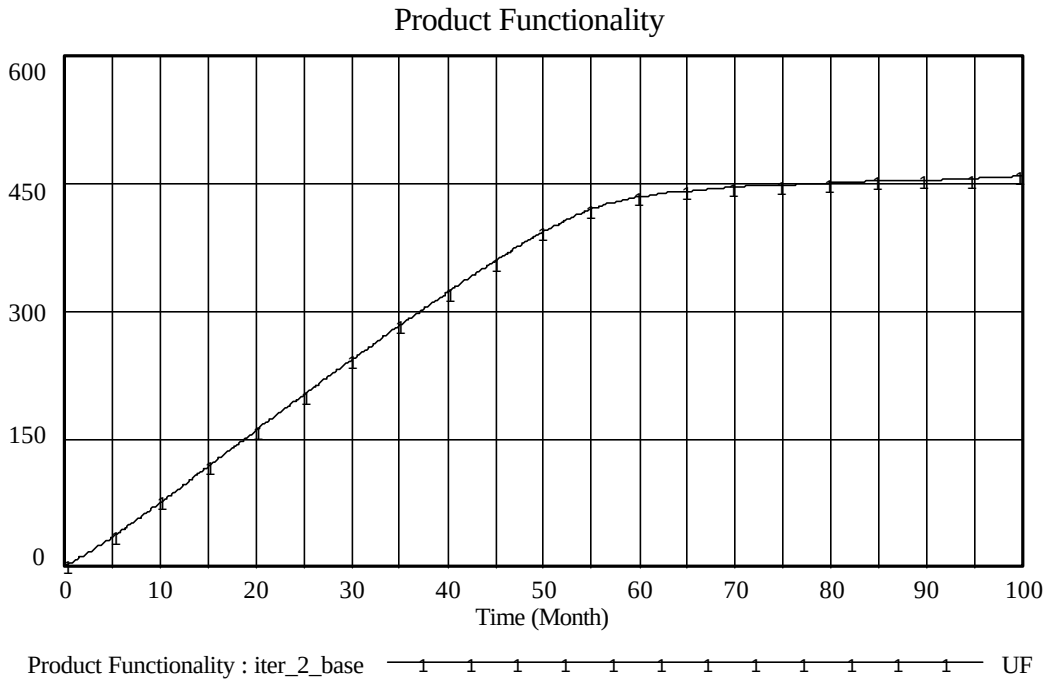


Figure 4.40. OSSD Model (Iteration II) Base Run - Product Functionality

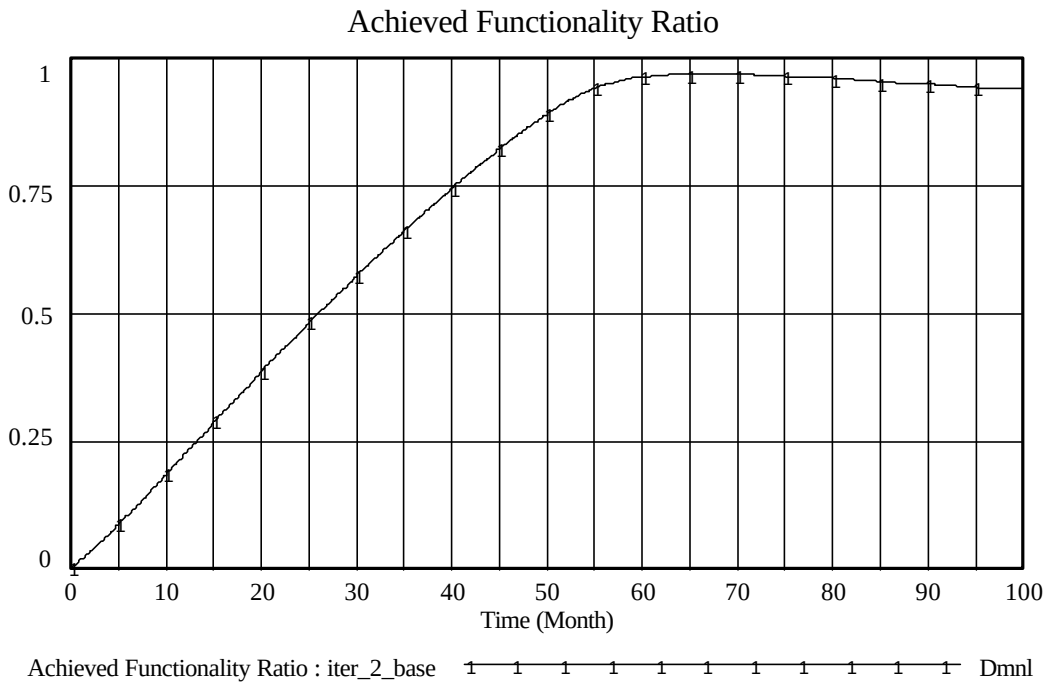


Figure 4.41. OSSD Model (Iteration II) Base Run - Achieved Functionality Ratio

When the model is run for 200 months instead of 100, achieved functionality ratio decreases for a while and than increases again to reach an equilibrium, which is lower than its peak value. (See Figure 4.42.) This again is attributable to the fact that the maintenance efforts within the community in order to keep the product up-to-date have to follow the improvement of the general level of technology with a certain delay, as was discussed within the context of the Iteration I version. In fact, looking closely at the behavior of achieved functionality ratio under the base run of the Iteration I version reveals that it decreases slightly after its peak at around month 75 due to the same reason. (Refer to Figure 4.18.)

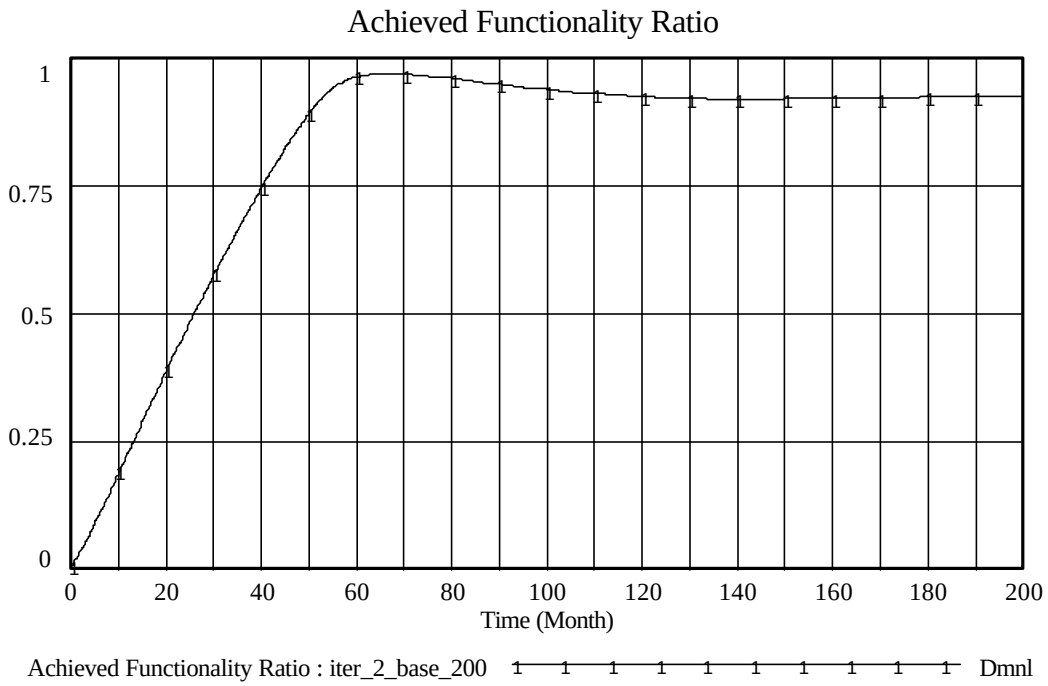


Figure 4.42. OSSD Model (Iteration II) Base Run - Achieved Functionality Ratio - Time Horizon Doubled

The number of developers under the base run conditions exhibits a behavior that is similar to that under Iteration I version in general terms. The number of developers first increases in a sub-linear fashion, reaches a peak level, and then exhibits a reversed-S-shaped decline. The major difference of the two behavior patterns is that the number of developers reaches it s peak earlier under the Iteration II version conditions. (See Figure 4.43.)

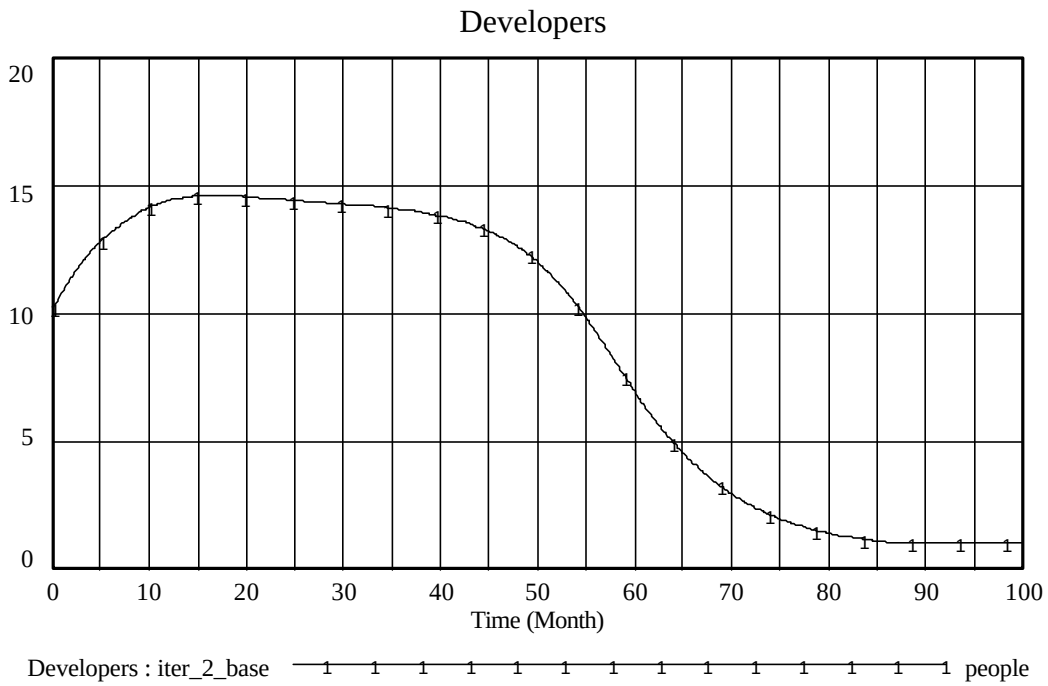


Figure 4.43. OSSD Model (Iteration II) Base Run - Number of Developers

As Figure 4.44 shows, the number of users exhibit an S-shaped growth pattern in general terms; however, that pattern is different than that under the Iteration I conditions. (See Figure 4.20.) Under Iteration II conditions, the growth in number of users does not reach the level of full potential user population by the end of the simulation horizon. , Instead, it continues to grow in a sub-linear fashion. This is due to the existence of competing products, which constitutes another user pool into which potential users may flow. Being successful in terms of operative/expected functionality ratio, the product continues to attract more users; however, the process is slower compared to the Iteration I case, since some potential users are currently using competing products. They have to decide giving up those products before they shift to the open source option. Another important point is that the users pool of the product in question will never reach the full number of potential users, as in the case of the Iteration I version, because there will

always be a portion of users who will chose to use competing products, no matter how successful the open source option is.

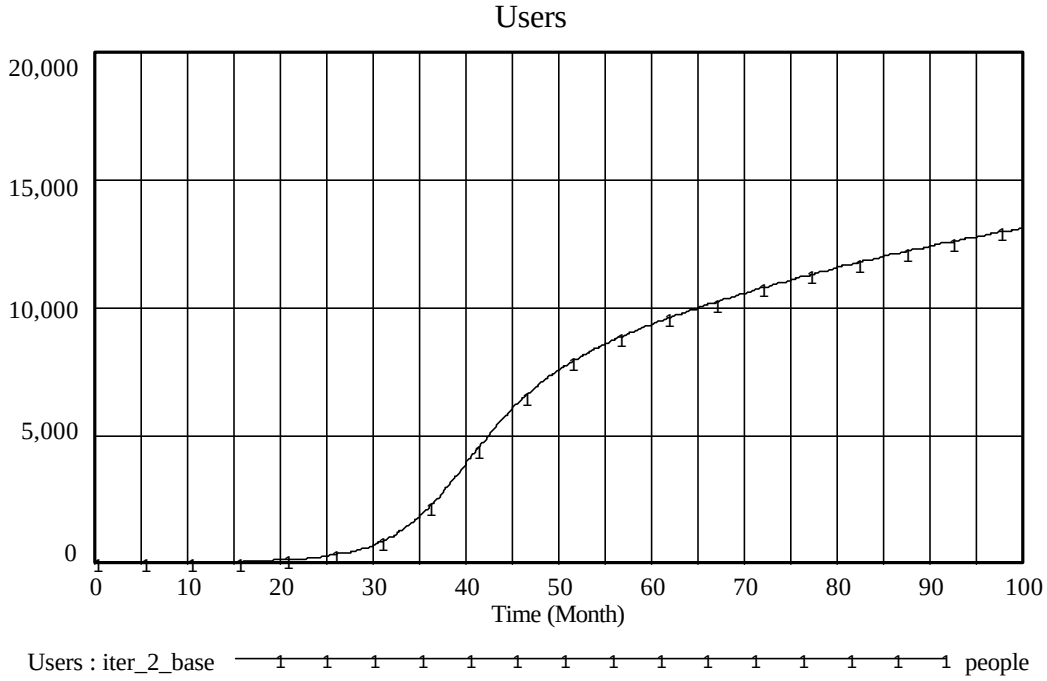


Figure 4.44. OSSD Model (Iteration II) Base Run - Number of Users

Running the Iteration II version for a bigger project yields behaviors similar to those observed under Iteration I version, in general terms (see Figures 6.45 through 6.48.). The initial limit on product functionality is set to 4000 UF for that run. The behaviors of the number of developers and the number of users are somewhat different in terms of the details, and that is attributable to the inclusion of other products competing for developers and users as discussed above about the base case run.

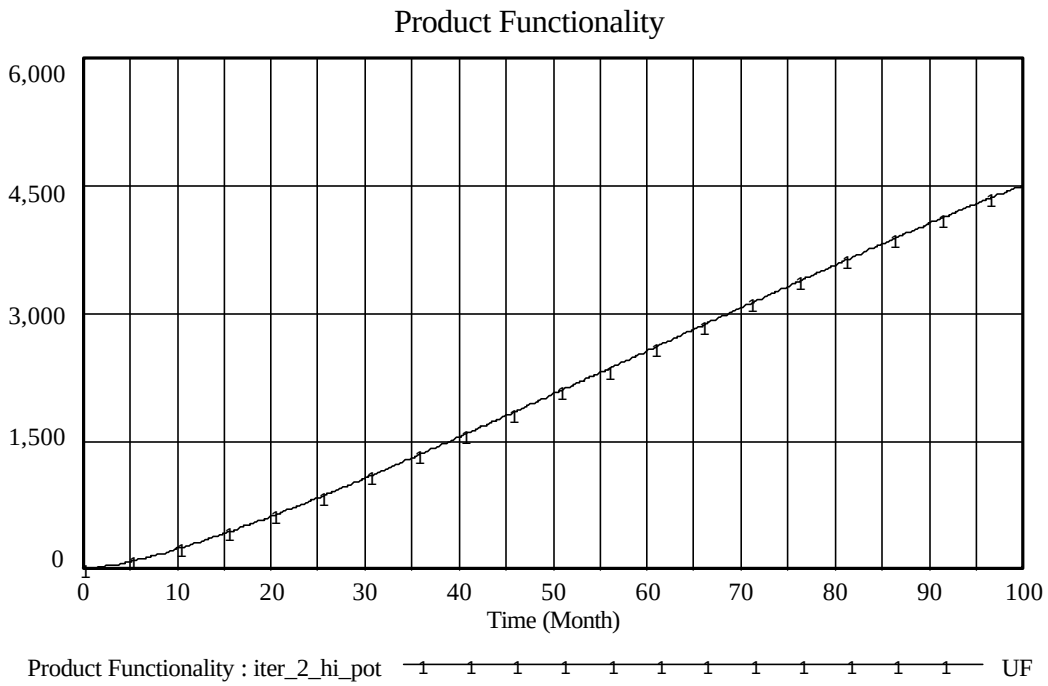


Figure 4.45. OSSD Model (Iteration II) High Potential Functionality Run - Product Functionality

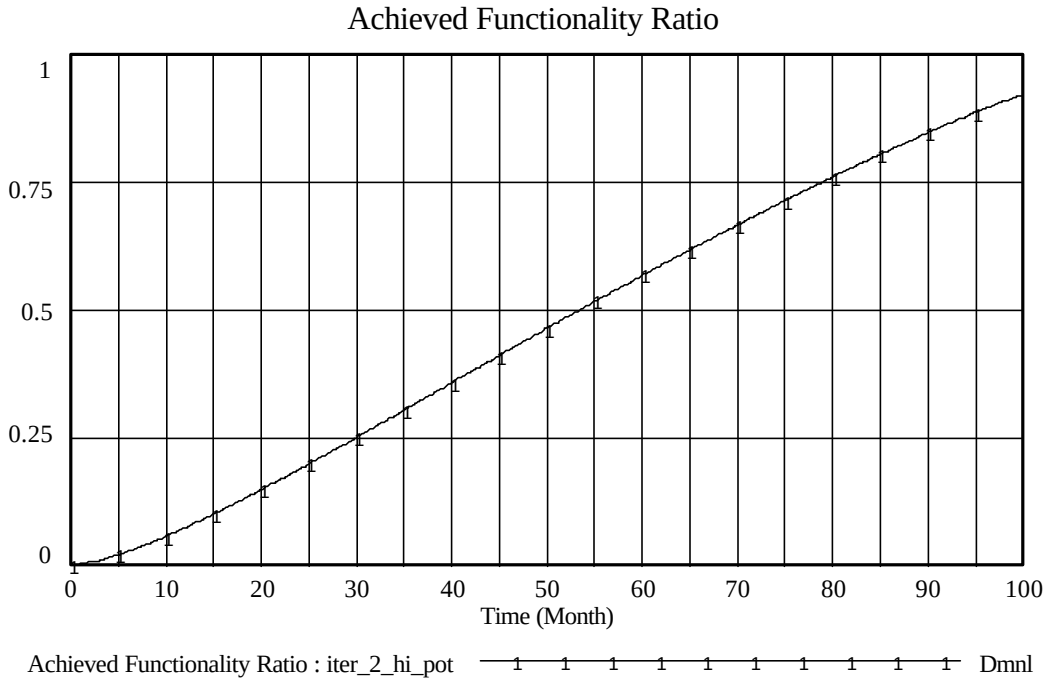


Figure 4.46. OSSD Model (Iteration II) High Potential Functionality Run - Achieved Functionality Ratio

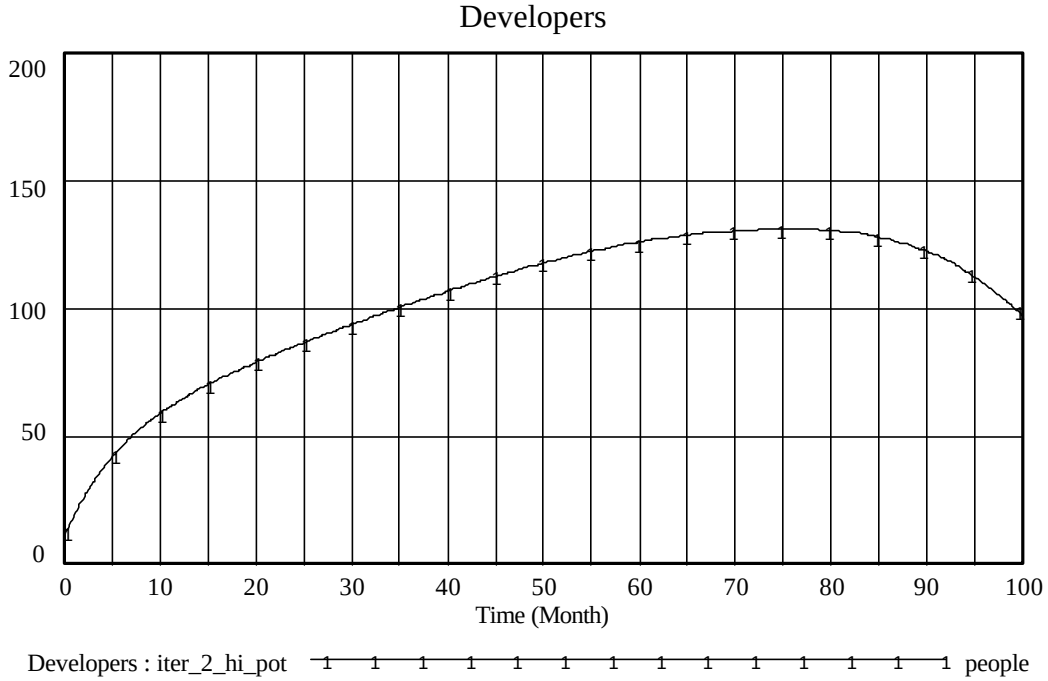


Figure 4.47. OSSD Model (Iteration II) High Potential Functionality Run - Number of Developer

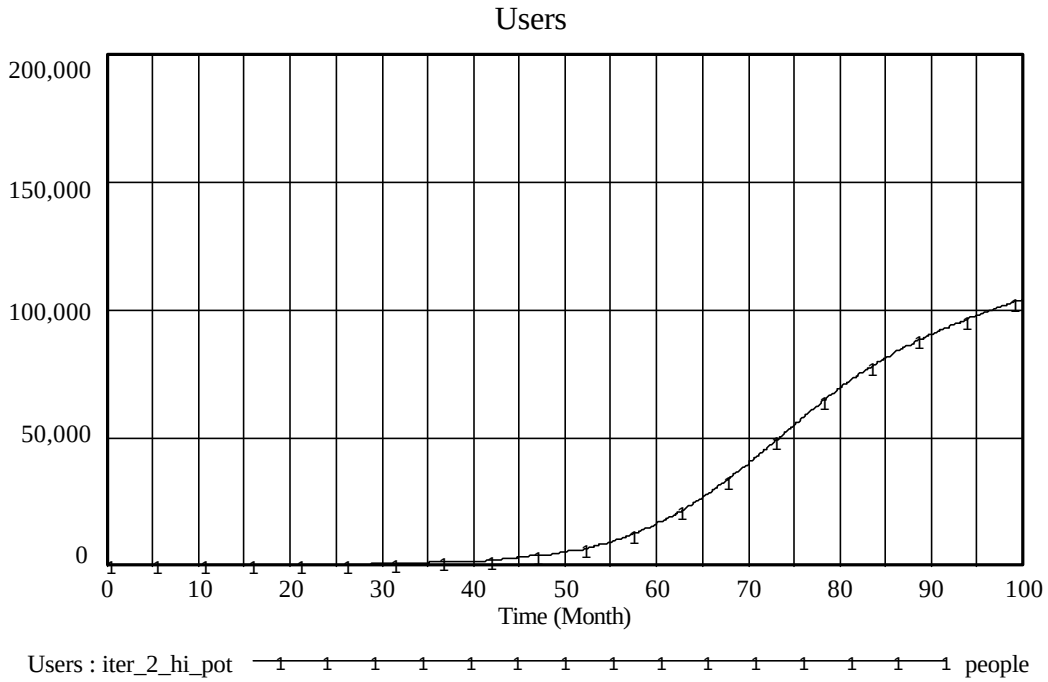


Figure 4.48. OSSD Model (Iteration II) High Potential Functionality Run - Number of Users

The critical runs for Iteration II version are those that are based on conditions that would slow the functionality growth substantially. The working dynamic hypothesis in this new version is that critically lower levels of average participation, or average production, as well as critically slower recruitment of new developers would generate too slow a functionality growth, and that would limit the community's growth in terms of both developers and users. To explore this case, a simulation was run with average participation set to seven hours per month per person, instead of the original value of 20 hours per month per person. As Figures 6.49 and 6.50 show, product functionality does not grow beyond a very low level, and the achieved functionality ratio barely reaches 13%, and then starts to decline.

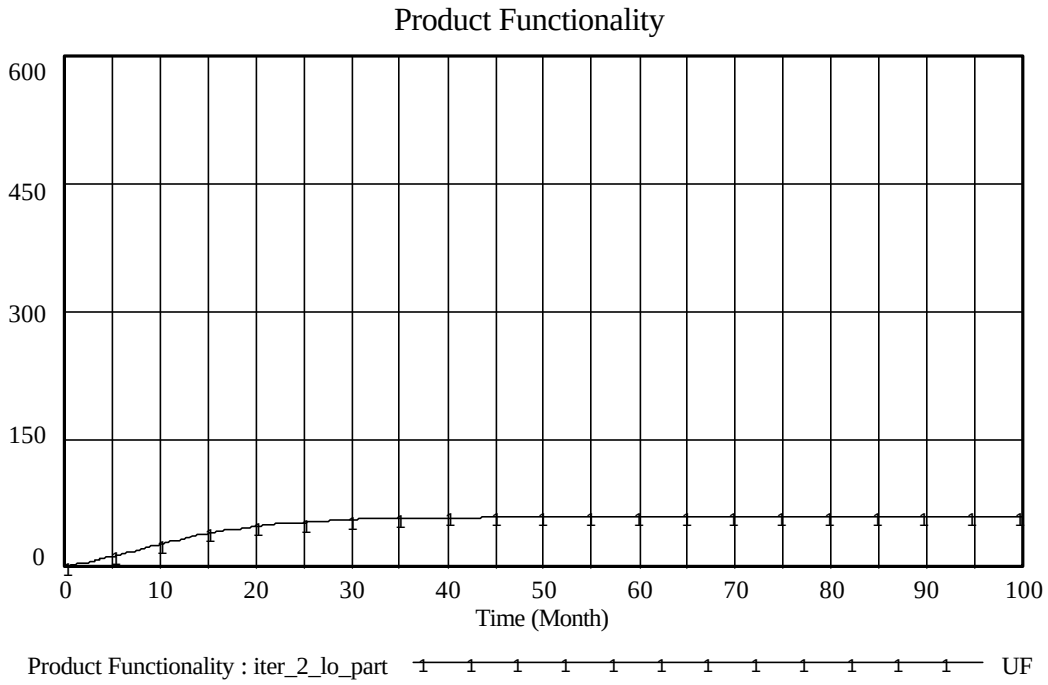


Figure 4.49. OSSD Model (Iteration II) Low Participation Run - Product Functionality

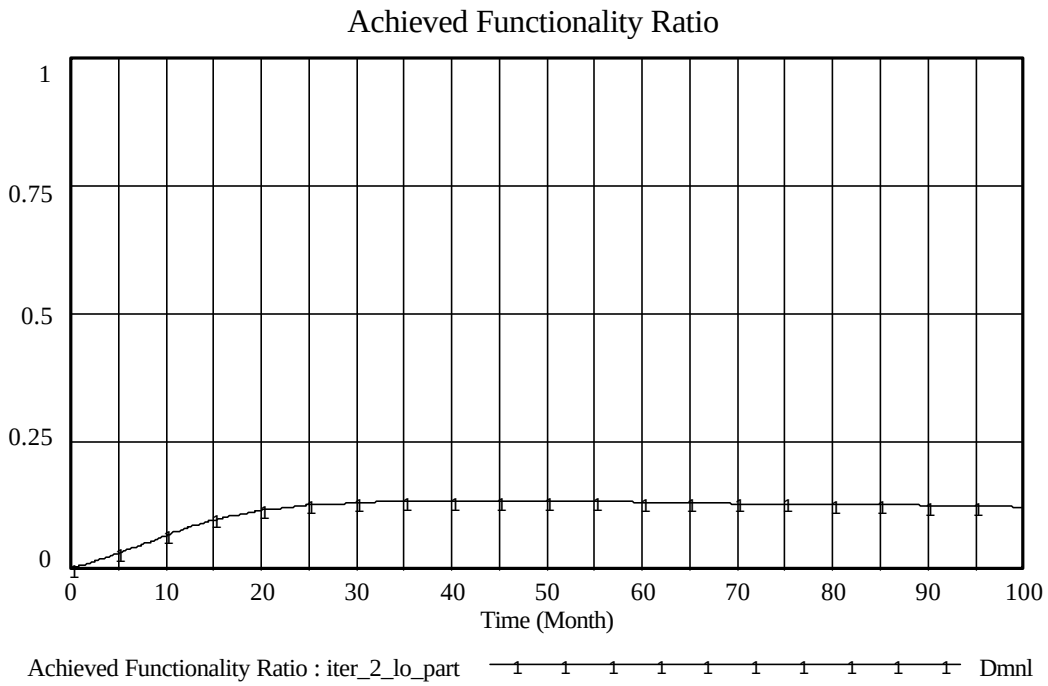


Figure 4.50. OSSD Model (Iteration II) Low Participation Run - Achieved Functionality Ratio

The number of developers increases for the first 10 months, driven by the expectations of the existing and incoming developers. However, as it becomes obvious that the achieved functionality ratio is far from the expected level the developer pool starts to decline. (See Figure 4.51.) The number of users increases slightly for a while, but does not go beyond the level of that of a "cult product," used only by an extremely small number of users for non-mainstream reasons. (See Figure 4.52.) The lack of success in attracting users is another reason that causes the developer pool to decline.

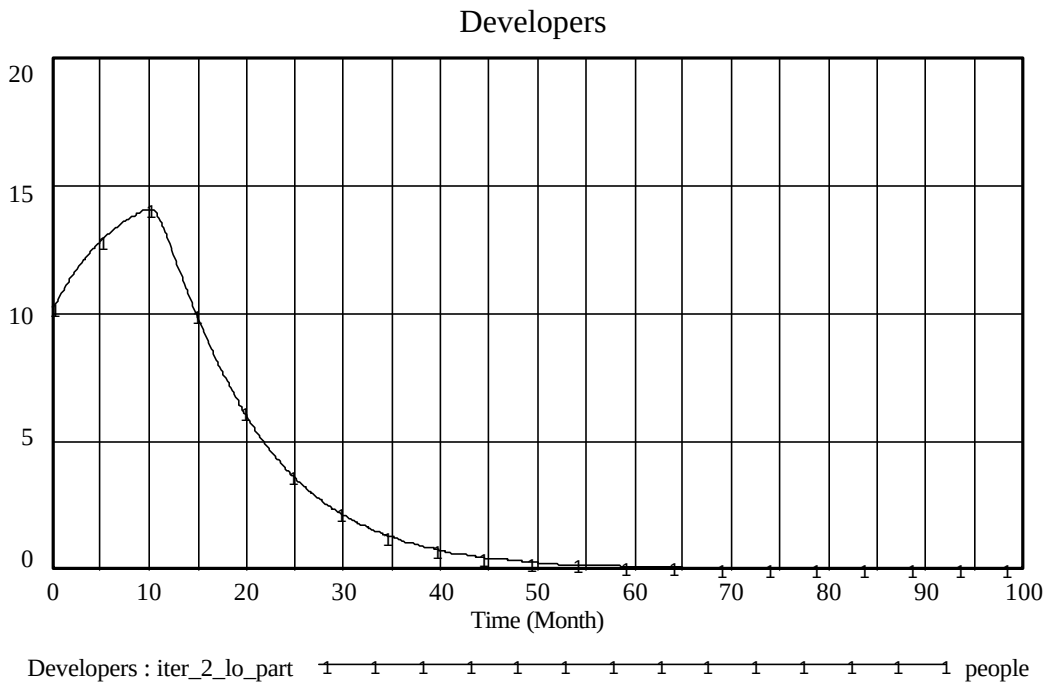


Figure 4.51. OSSD Model (Iteration II) Low Participation Run - Number of Developers

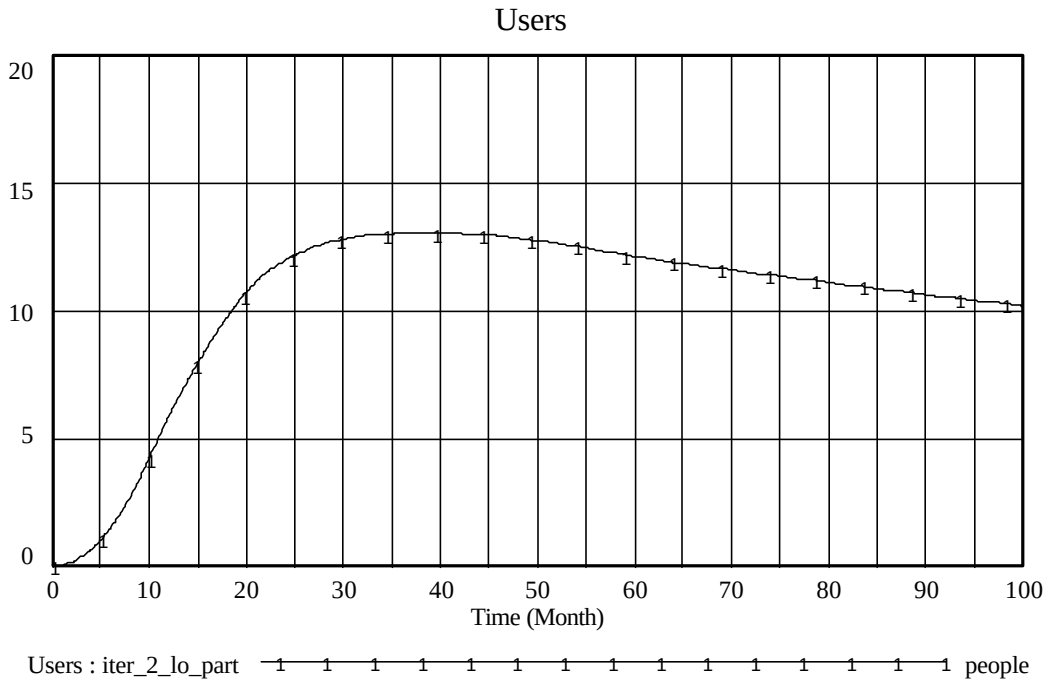


Figure 4.52. OSSD Model (Iteration II) Low Participation Run - Number of Users

This second iteration of the model building process provided a version that can explain failed communities and projects as well as those that succeed. As such, it has more explanatory power than the previous version. However, the Iteration II version does not include product quality, which is an important factor in terms of attracting and retaining developers and users. Quality control and maintenance is also important for the purposes of the model, since it occupies a certain portion of developers' time spent on the project. The Iteration III version is developed to address these concerns.

4.4. Iteration III: Adding Quality

The Iteration III version of the model involves major changes over the previous version, including the addition of three new sectors (Quality, Developer Time Allocation, Leader Time Allocation), and the separation of the developer population into two conceptual groups. The developer population is grouped under regular developers and leaders. Regular developers are called "Developers" for the purposes of the model. "Developers" are conceptualized as participants who have more moderate levels of talent and participation compared to those of the leaders. While each leader spends 30 hours per month on the project, developers spend 20 hours per month per person. Though the talent factor is taken into account while conceptualizing the two participant populations, it is not addressed with this version of the model. For the purposes of the Iteration III version, there is no difference between the talent levels of developers and leaders. The talent factor is addressed with the Iteration IV version. Developer and leader populations together form the "Participants" population.

Figure 4.53 shows the changes in the developers sector due to adding the concept of Leaders to the model. Total production is divided into two -- production by developers

and production by leaders. Another change in the developers sector, which is caused by adding quality control and maintenance functions to the model, is that production by developers is not based on the total time developers spend on the project, but on the number of total developers hours allocated to production.

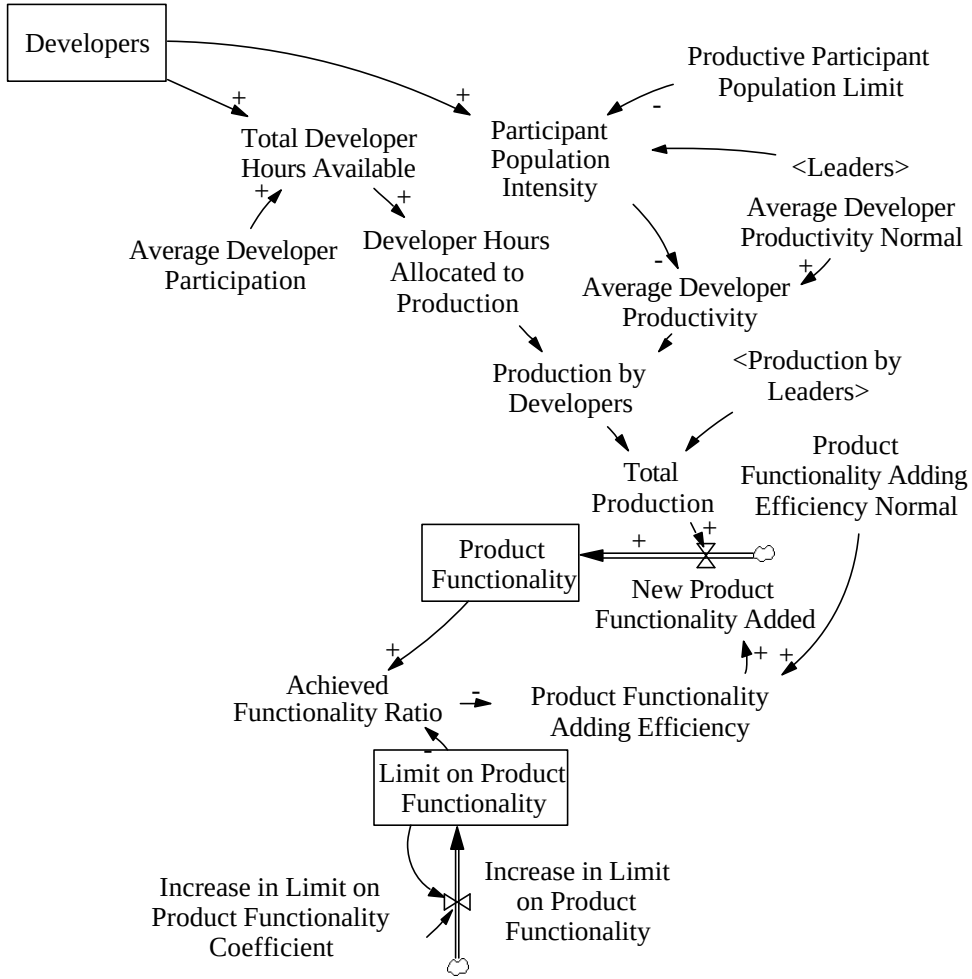


Figure 4.53. OSSD Model (Iteration III) Changes in the Developers Sector due to Adding Leaders to the Model

One of the three sectors added to the model with this iteration is the quality sector. As shown in Figure 4.54, production by developers and leaders add to the size of the product, which is defined as lines of code. Production generates new functionality, which adds to product functionality and new bugs in the code. This, in turn, adds to the pool of unknown bugs in the code. Developers and leaders work on detecting the unknown bugs in the code and move the ones they detect to the pool of known bugs in the code. (See Figure 4.54.)

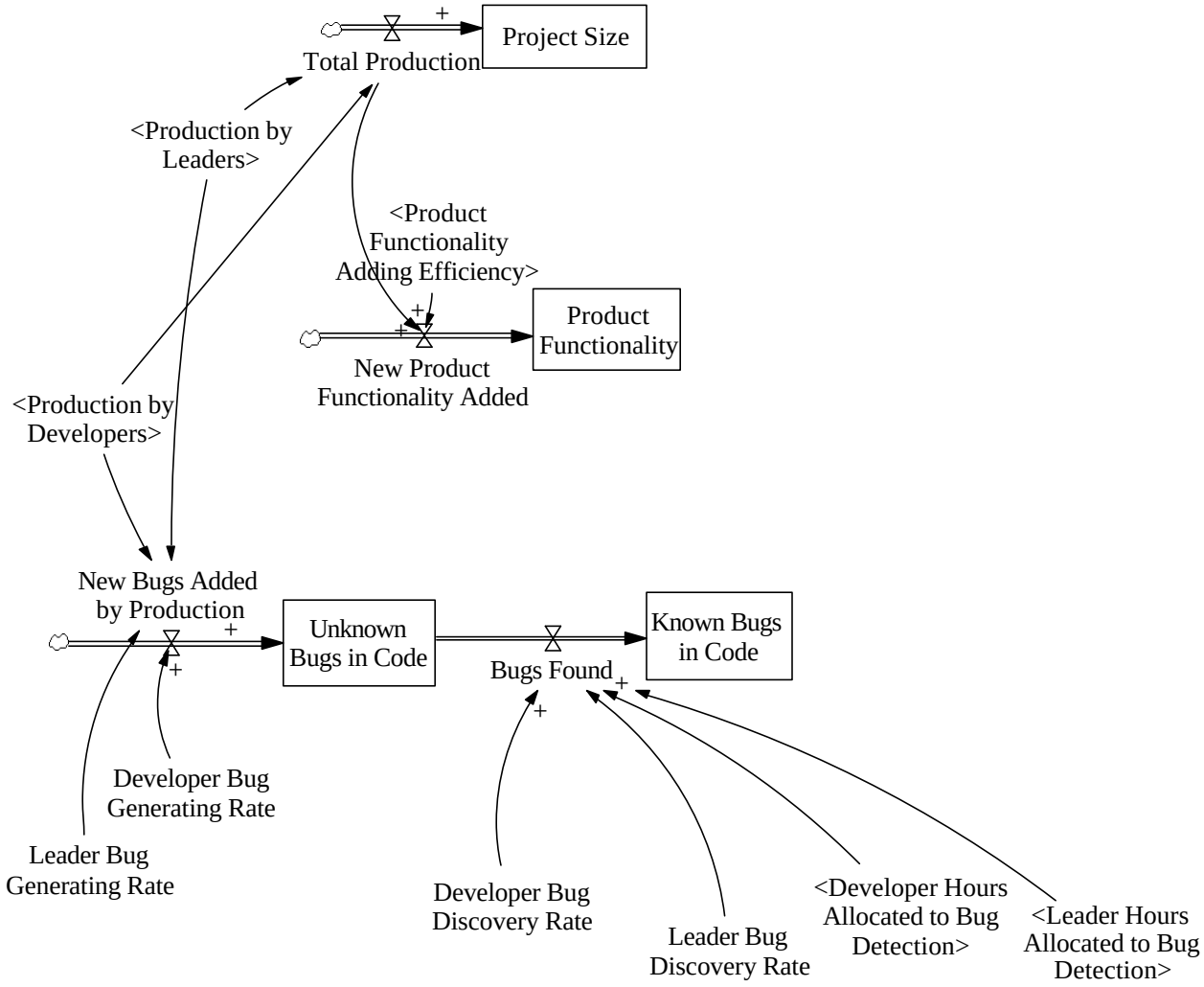


Figure 4.54. OSSD Model (Iteration III) Quality Sector

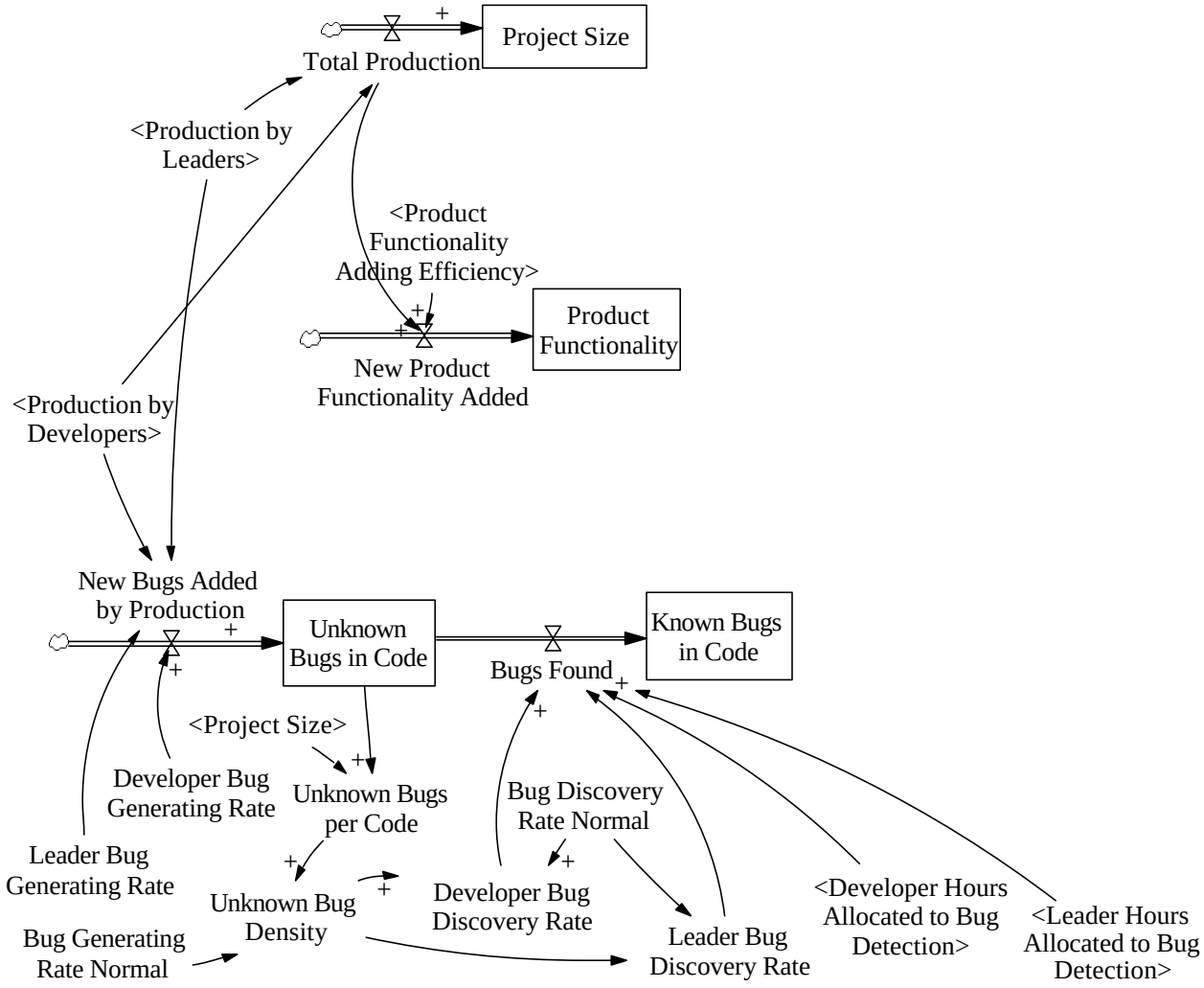


Figure 4.55. OSSD Model (Iteration III) Quality Sector

Several factors affect the number of unknown bugs developers and leaders discover in a given month. The main factor is the time developers and leaders spend on detecting bugs. The other factor that determines the rate of bug discovery is the density of unknown bugs in the code (Abdel-Hamid and Madnick 1991 pp.105). It is assumed that as the unknown bug density increases, it becomes easier, and consequently faster to discover unknown bugs. Unknown bug density is defined as a normalized ratio of relative number of unknown bugs per line of code. The benchmark used for normalization is the normal rate of bugs generated by participants. (See Figure 4.55.)

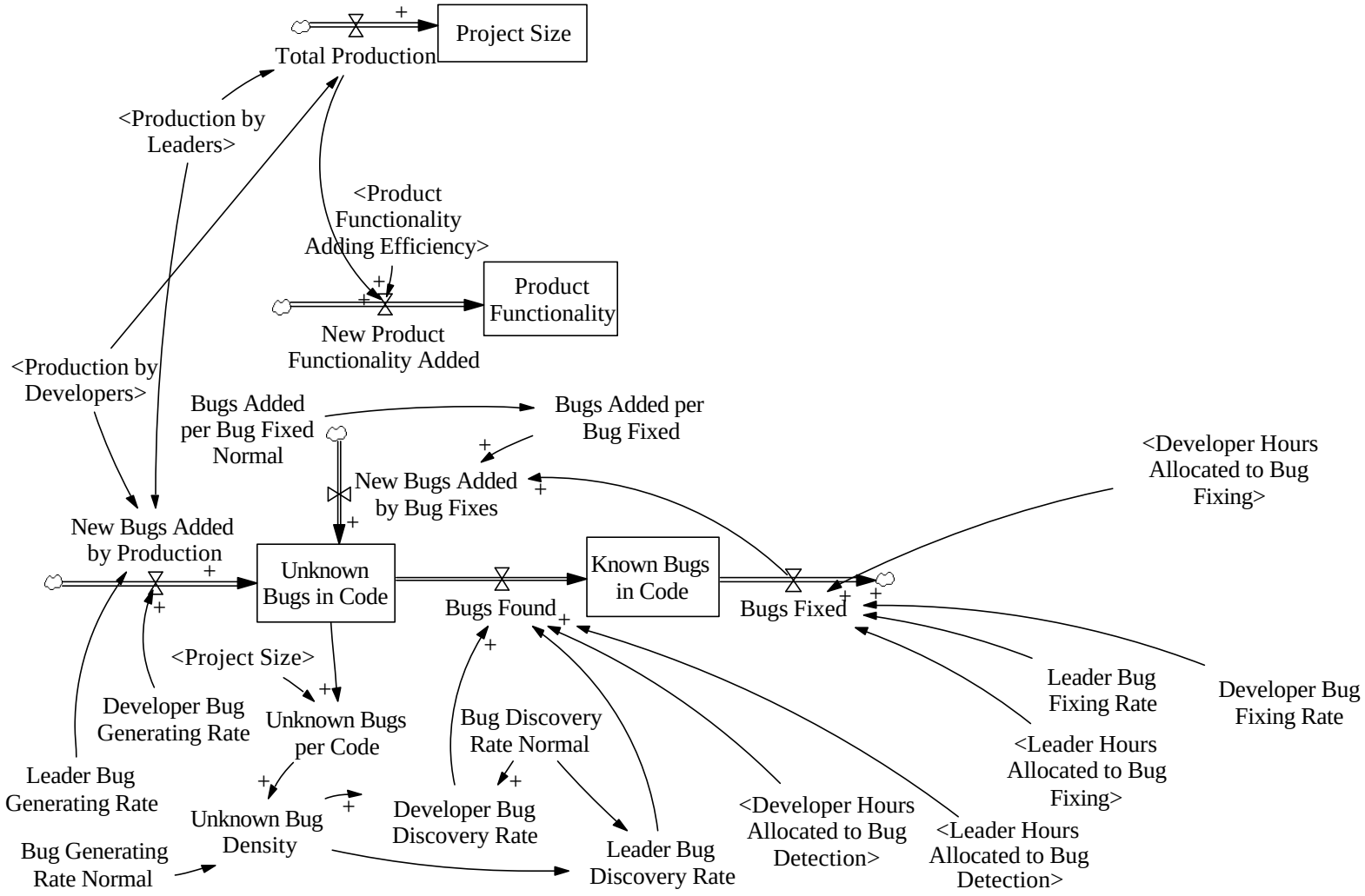


Figure 4.56. OSSD Model (Iteration III) Quality Sector

Developers and leaders also spend time on fixing the known bugs in the code. The number of known bugs developers and leaders fix in a given month is a function of allocated time developers and leaders spend on the specific activity of bug fixing. The other factor that determines the number of bugs fixed per month is the base rates at which developers and leaders fix bugs. These are defined as constants for the purposes of the Iteration II version of the model. Bug fixing is an activity that is known to generate bugs itself (Abdel-Hamid and Madnick 1991 pp.108). Developers and leaders add new bugs to the pool of unknown bugs as they fix known bugs. (See Figure 4.56.) The rate at which new bugs are added during bug fixing is determined by the quality of the bug fixing activity. Quality of bug fixing is defined as a constant for the purposes of the Iteration III version. (See Figure 4.57.)

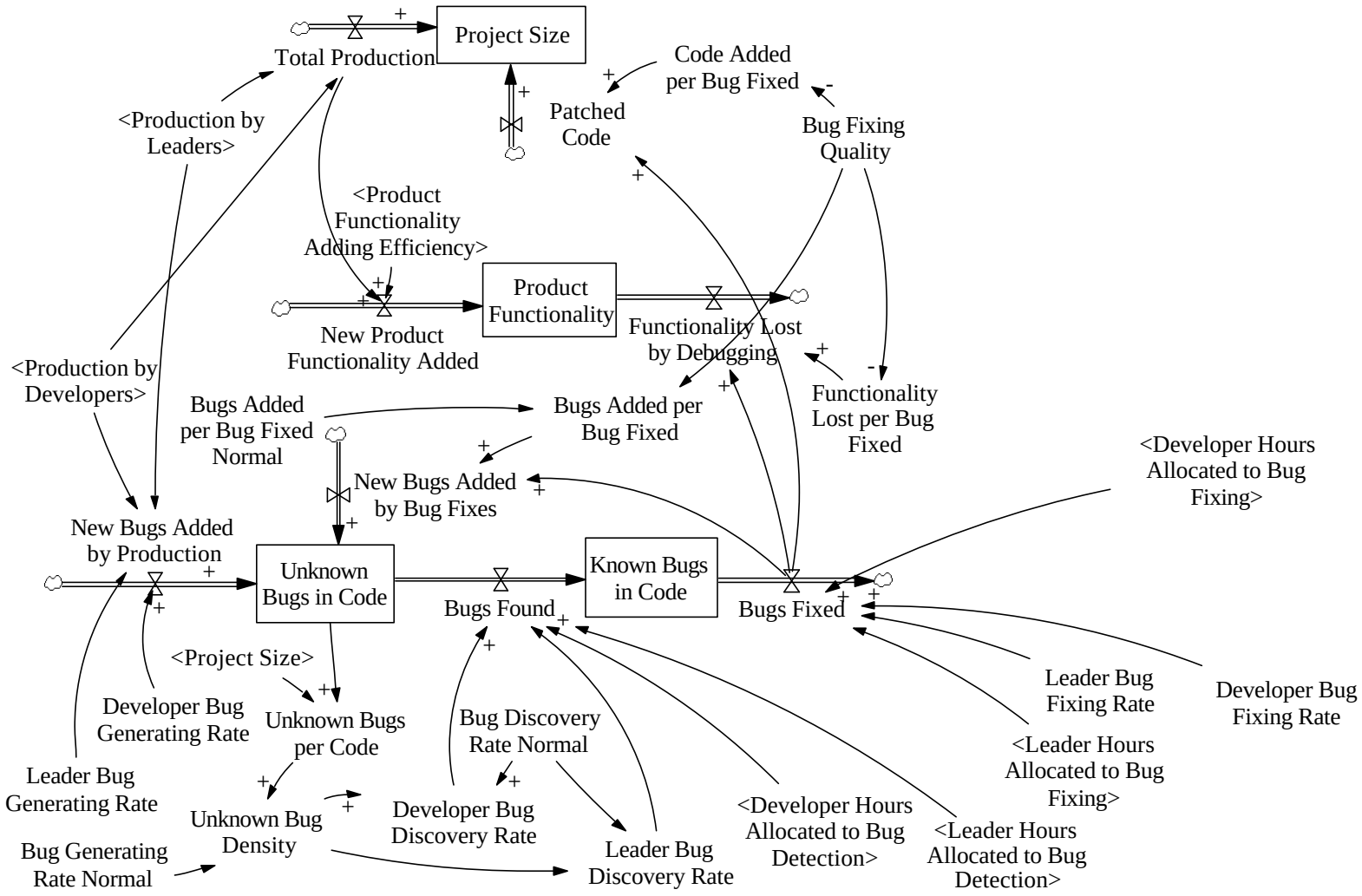


Figure 4.57. OSSD Model (Iteration III) Quality Sector

Figure 4.57 shows two more adverse effects of bug fixing. As developers and leaders fix bugs they add extra code, and thus increase the project size without adding any functionality. Furthermore, they inadvertently lose existing functionality as they fix bugs. Both the amount of code added and the amount of functionality lost per bug fix depends on the bug fixing quality.

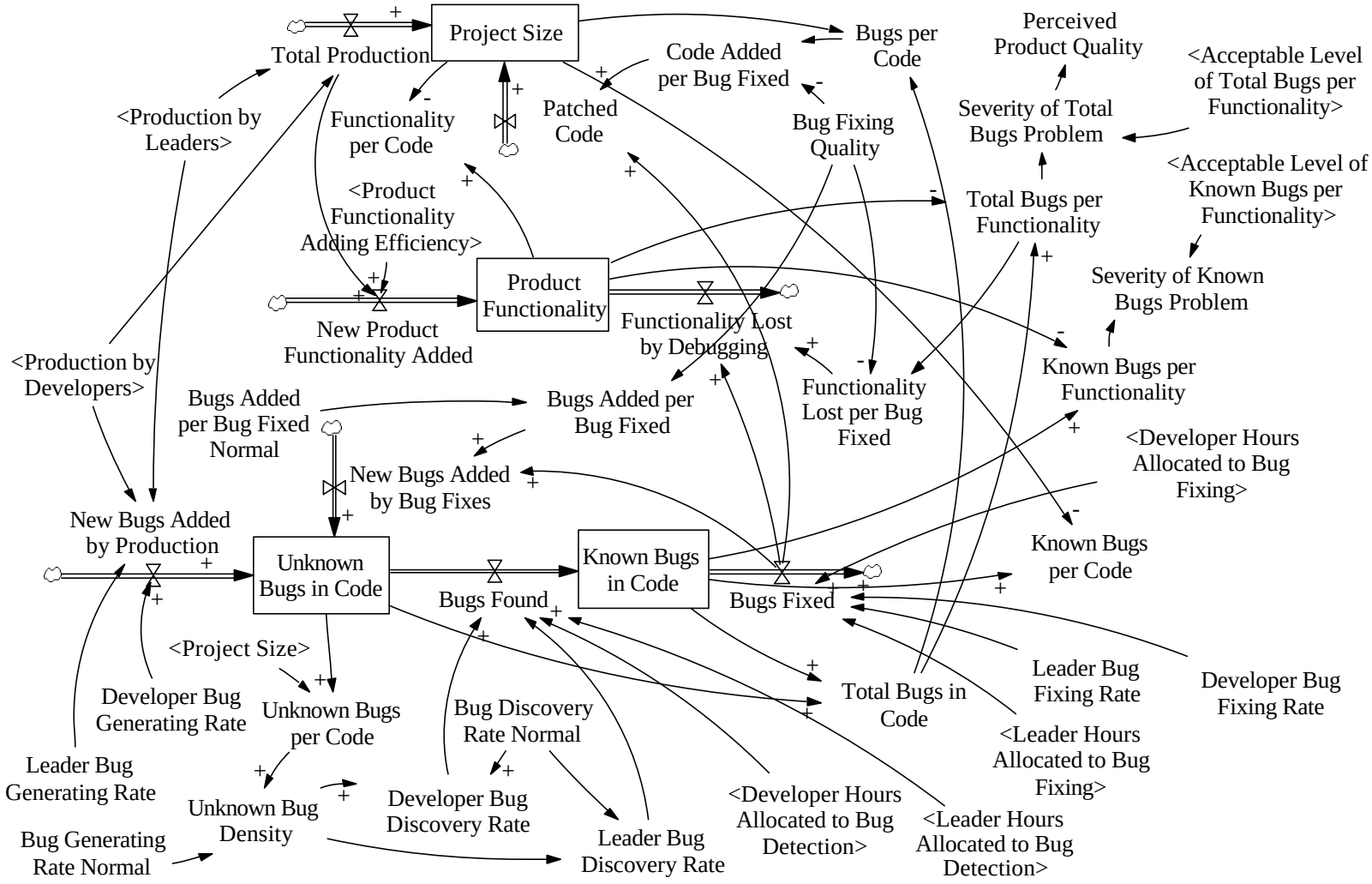


Figure 4.58. OSSD Model (Iteration III) Quality Sector

Brooks (1995 pp.121) argues that the longer users use a software product, the further they push the product to the limits of its capabilities. Thus they increase the probability of bugs manifesting themselves through use. Consequently, this study assumes that the bugs in the code would manifest themselves as the product is used and pushed toward its limits of functionality. Under this assumption, the probability of bugs manifesting themselves becomes greater as the number of bugs per unit of functionality increases, and this ultimately decreases the perceived quality of the product. As shown in Figure 4.58 the number of total bugs per functionality induces a relative severity level, with respect to an acceptable level of bugs per functionality. The severity of the total bugs problem determines the perceived quality of the product. Severity of the level of known bugs in the code is another manifestation of the bugs problems, which is assessed by the participants. This ratio affects the level of concern about fixing bugs, and ultimately determines the participant time allocated to bug fixing activity.

Figure 4.59 shows how the perceived quality level of the product affects the developer sector. Perceived quality level has a negative effect on the rate of leaving developers. Everything else being equal, as the quality increases fewer developers will be inclined to leave the community.

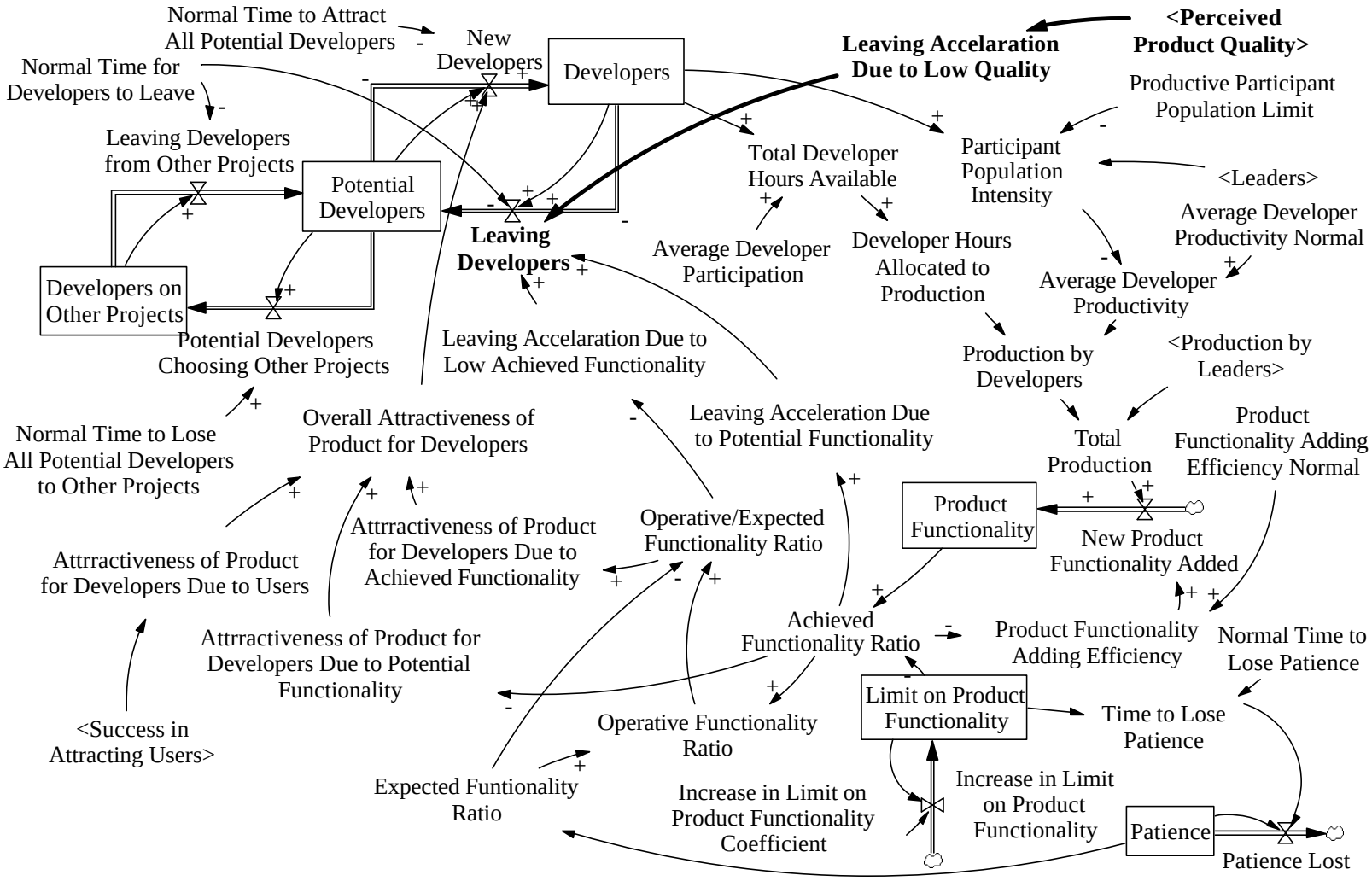


Figure 4.59. OSSD Model (Iteration III) Changes in the Developers Sector due to Adding Quality Factor to the Model

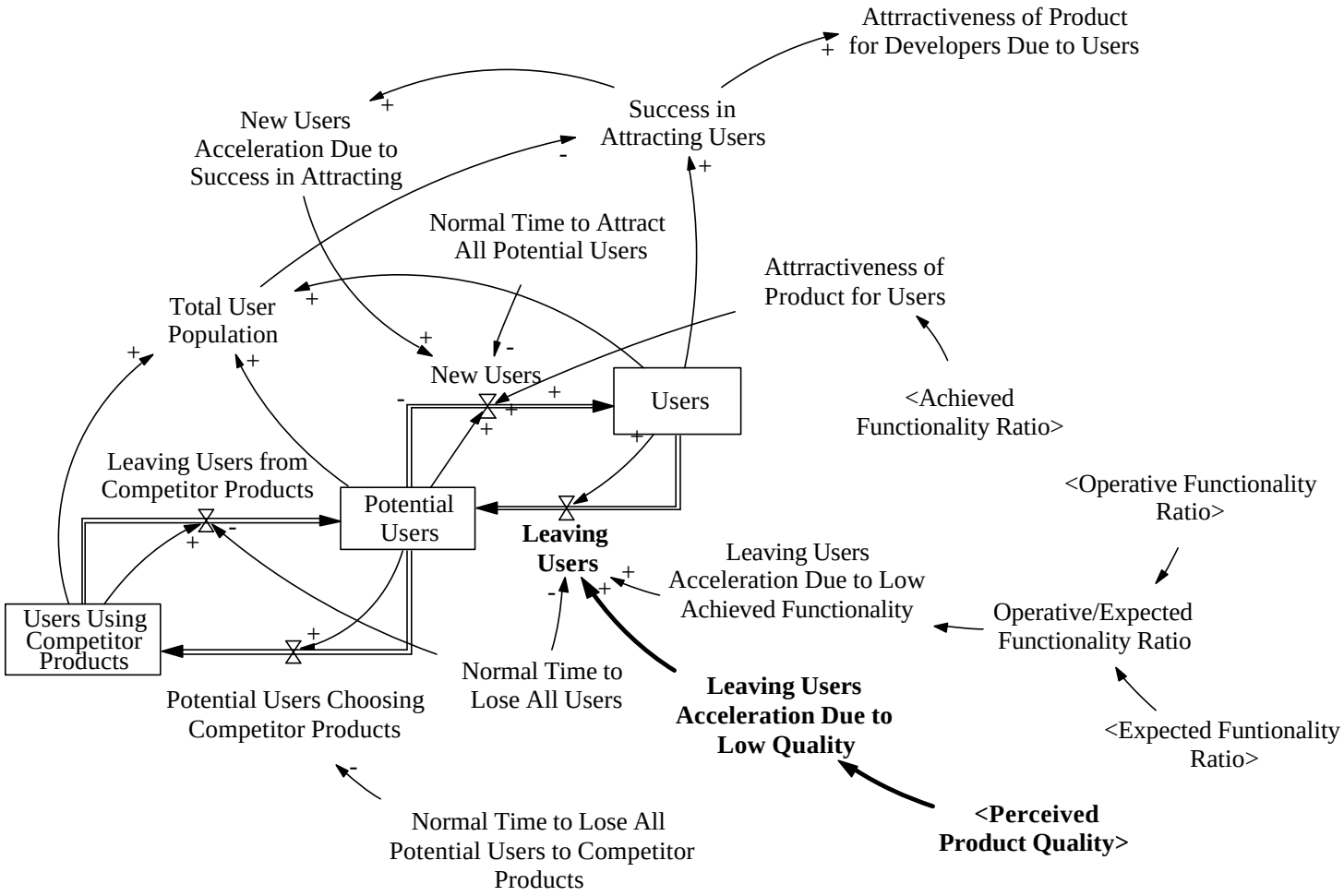


Figure 4.60. OSSD Model (Iteration III) Changes in the Users Sector due to Adding Quality Factor to the Model

Perceived quality level has a similar effect on the users sector, as shown in Figure 4.60. As the quality level increases fewer users will want to quit using the product. The addition of these effects introduces two important balancing loops to the model. (See Figure 4.61.) In balancing loop 4, as developers participate in the production and produce functionality they add new bugs to the product. Everything else being equal, new bugs increase the number of total bugs and this decreases the perceived quality, which in turn accelerate the rate of leaving developers. In balancing loop 5, as perceived quality decreases, more users quit using the product, and that negatively affects success in attracting users. This, in turn, decreases the attractiveness of the product for developers (because the number of users is lower), and decreases the number of new developers.

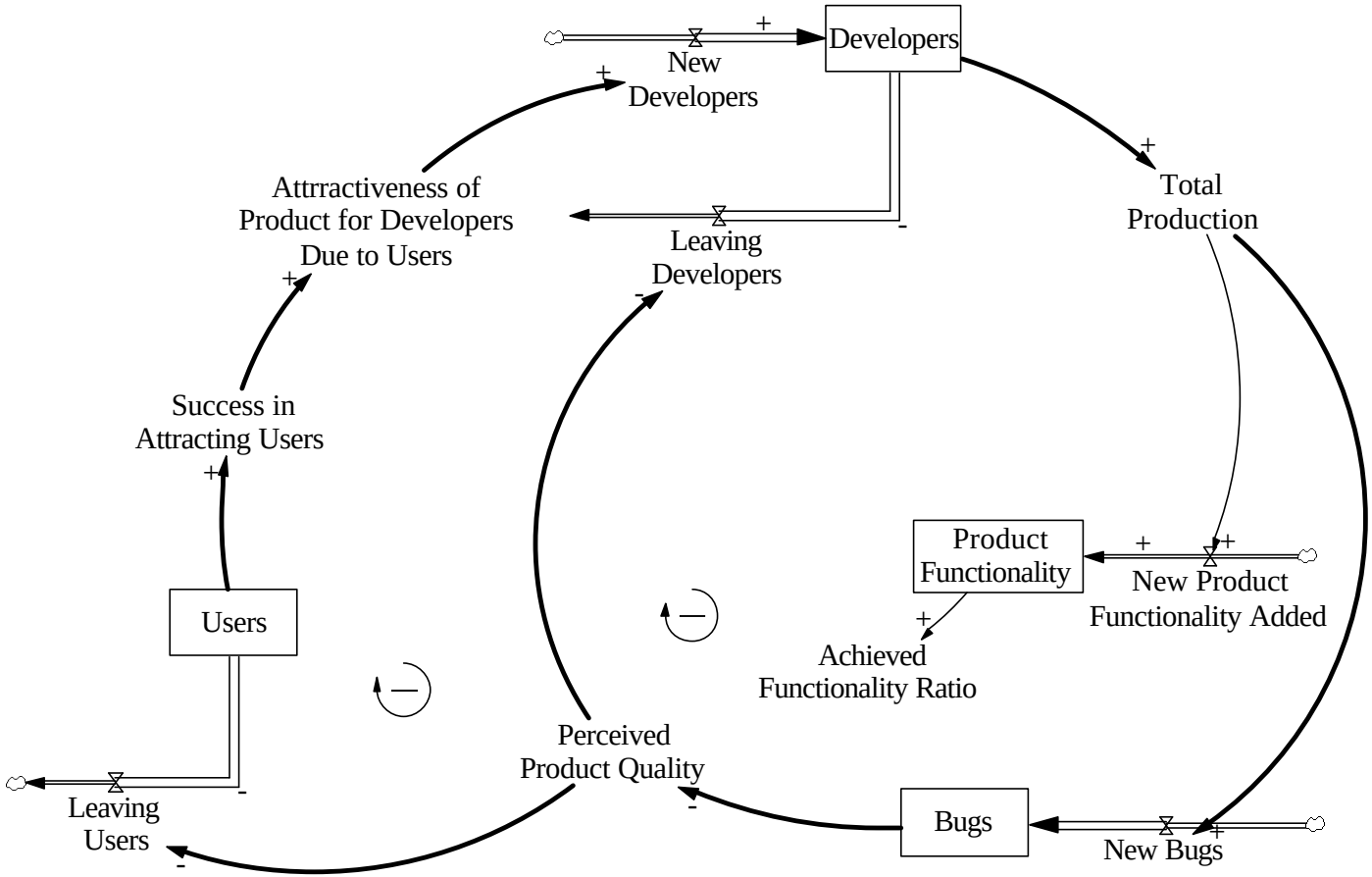


Figure 4.61. OSSD Model (Iteration III) Balancing Loop 4 and Balancing Loop 5: “More Production Causes More Bugs, and That Retains Fewer Existing Developers,” and “More Production Causes More Bugs, That Retains Fewer Existing Users, and Attracts Fewer New Developers.”

The second sector added to the model in Iteration III is the developer time allocation. Here, the total developer hours available are allocated to production, bug detection, and bug fixing. The severity levels of the total bugs problem, and the known bugs problem indicate certain levels of pressure for bug detection and bug fixing respectively. These pressures in turn determine the developer hours needed for bug detection and bug fixing. Developer hours needed for these two activities constitute the total developer hours needed for non-production tasks, which together with developer hours planned for production constitute total developer hours needed. (See Figure 4.62.)

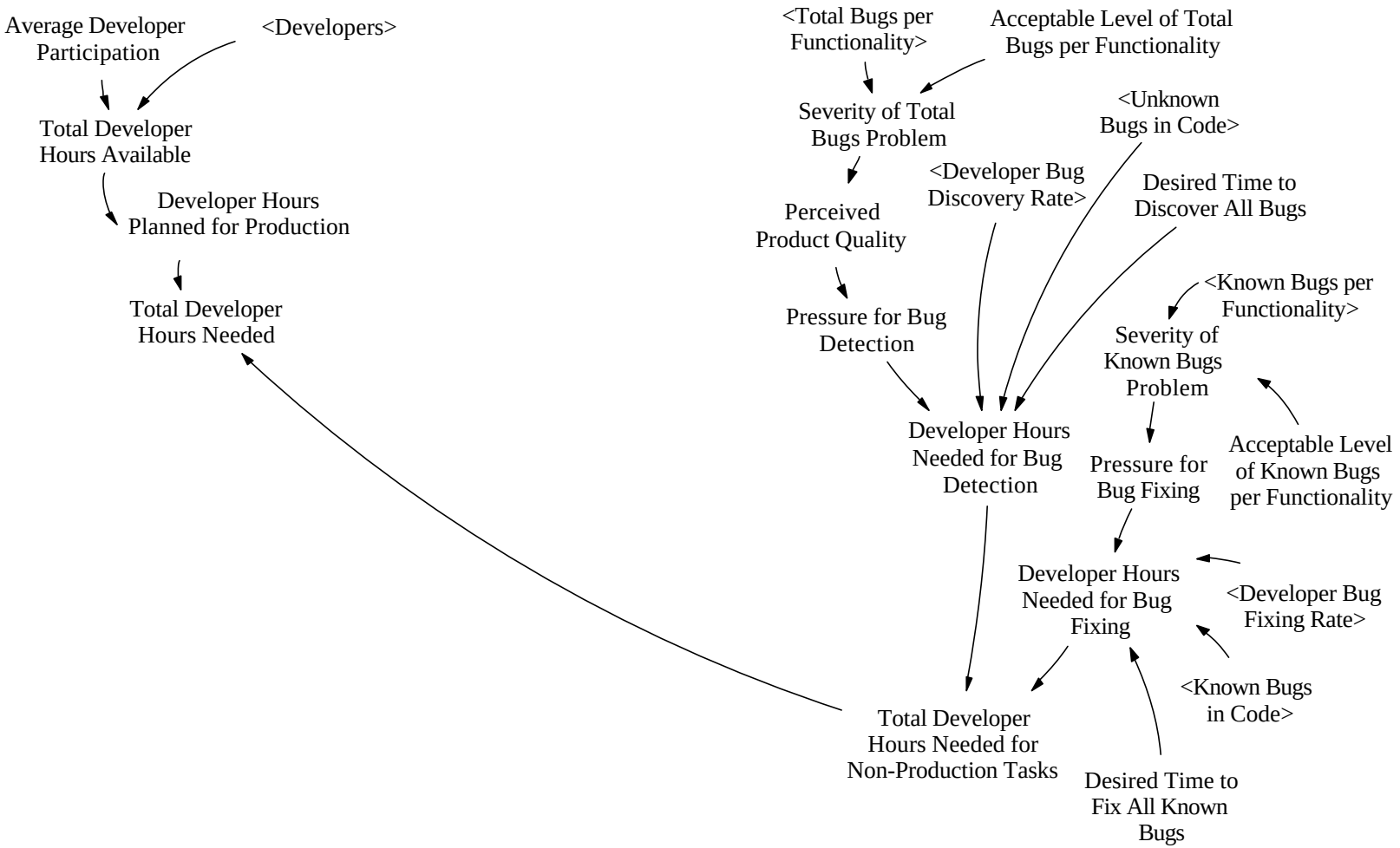


Figure 4.62. OSSD Model (Iteration III) Developer Time Allocation Sector

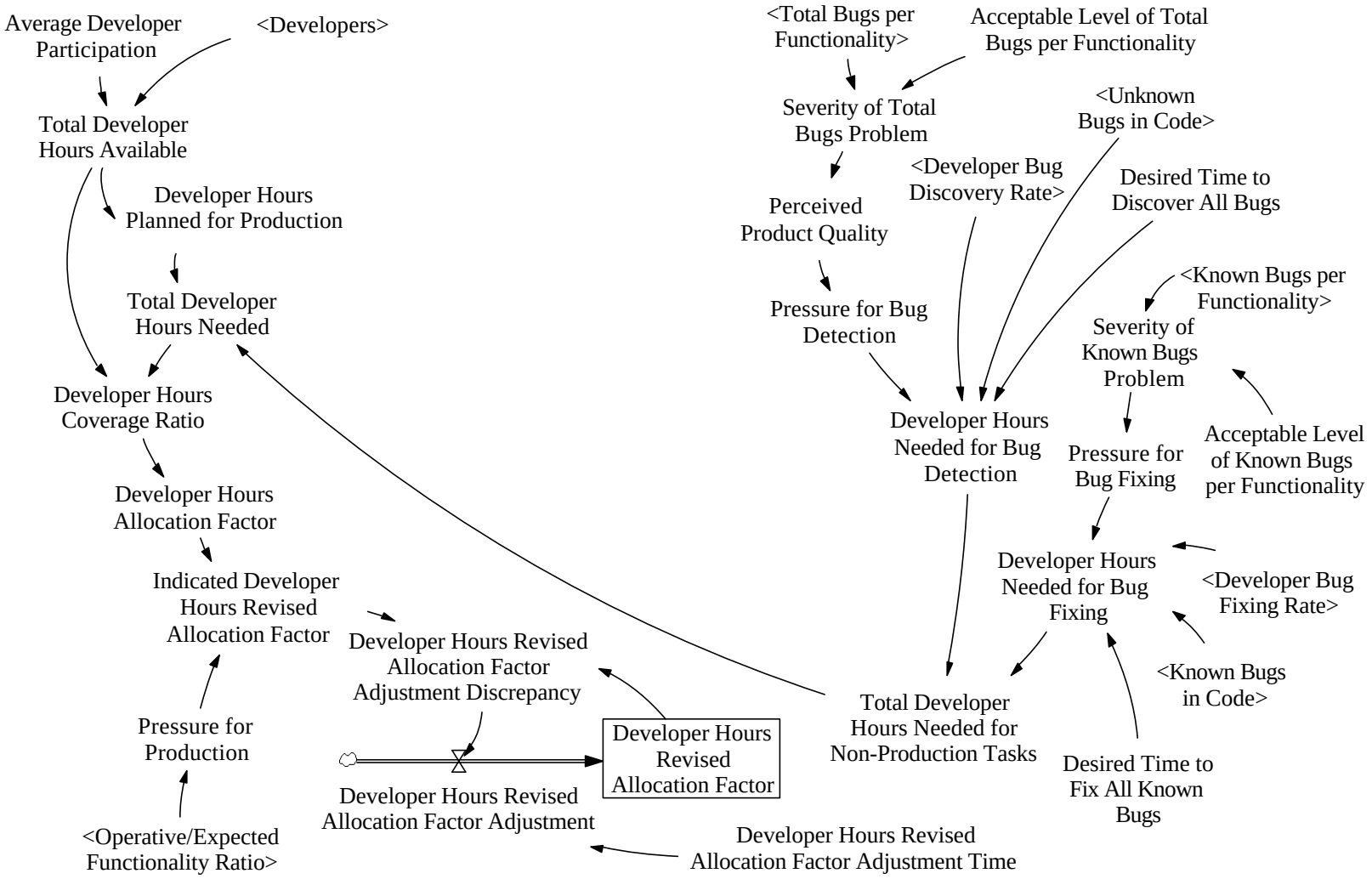


Figure 4.63. OSSD Model (Iteration III) Developer Time Allocation Sector

The ratio of developer hours available and developer hours needed indicates the developer hours allocation factor, which determines what percentage of the needed time is actually allocated to non-production tasks. If the operative/expected functionality ratio is too low, this allocation factor decreases further. This revised factor is assumed to change gradually over time, and is consequently represented as a smooth, or in other words a historical average of the indicated revised allocation factor. (See Figure 4.63.)

As Figure 4.64 shows, the revised allocation factor determines the actual hours allocated to each non-production task. The difference between the total developer hours available and the total developer hours allocated to non-production tasks is the number of actual developer hours allocated to production.

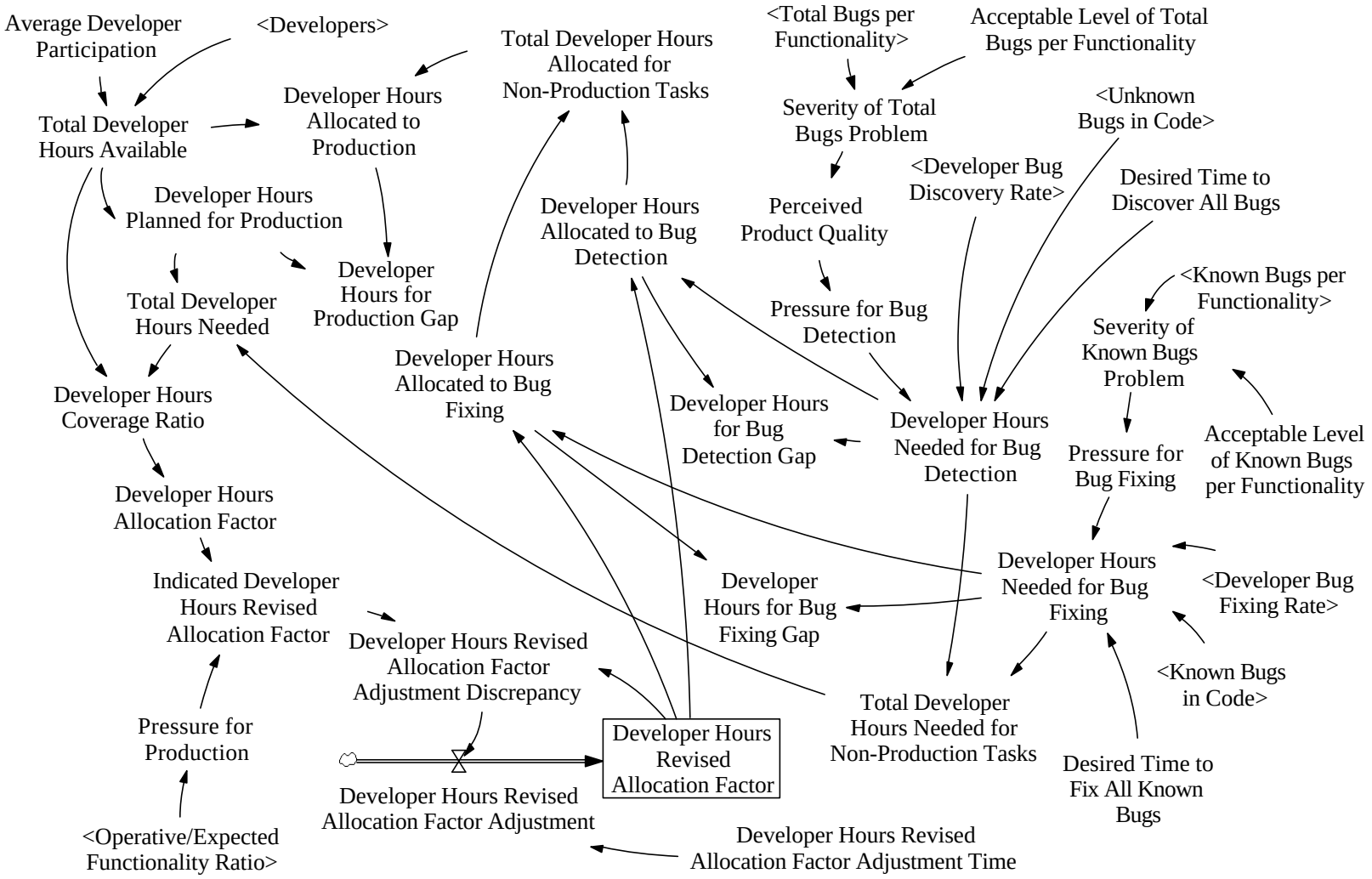


Figure 4.64. OSSD Model (Iteration III) Developer Time Allocation Sector

The leader time allocation sector is the third sector added to the model in Iteration III. Here, the total leader hours available are allocated to production and non-production tasks in more or less the same way as in the developer time allocation sector. Leader hours needed for bug detection and bug fixing are determined by the respective gaps between the needed and allocated developer hours for each task. Leader hours needed for bug detection, leader hours needed for bug fixing and leader hours planned for production together constitute the total leader hours needed. (See Figure 4.65.)

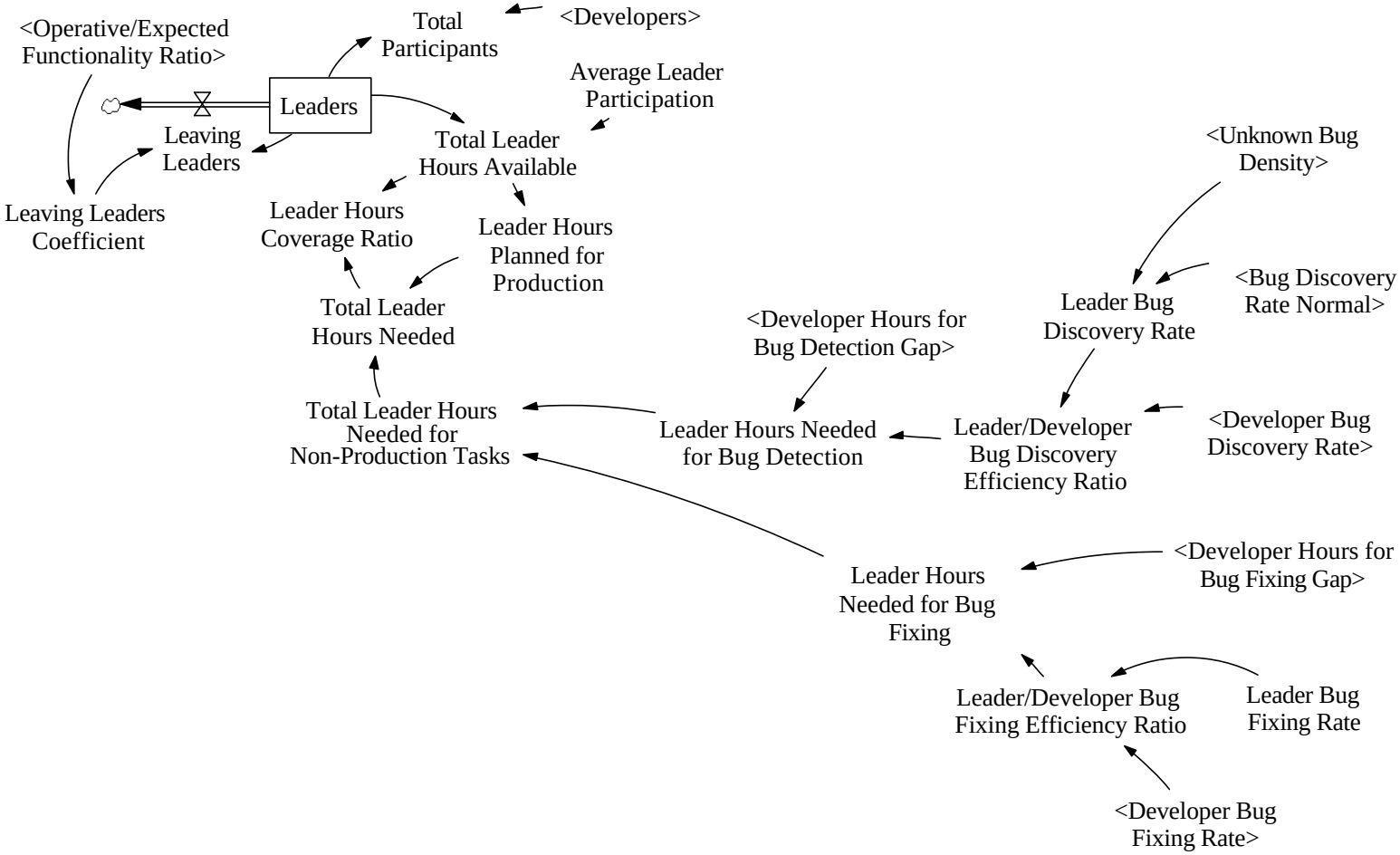


Figure 4.65. OSSD Model (Iteration III) Leaders Time Allocation Sector

Leader hours coverage ratio, which is the ratio of leader hours available and leader hours needed, indicates the leader hours allocation factor. Under production pressure conditions, this factor decreases further, and indicates a revised allocation factor just like the one in the developer time allocation sector. (See Figure 4.66.)

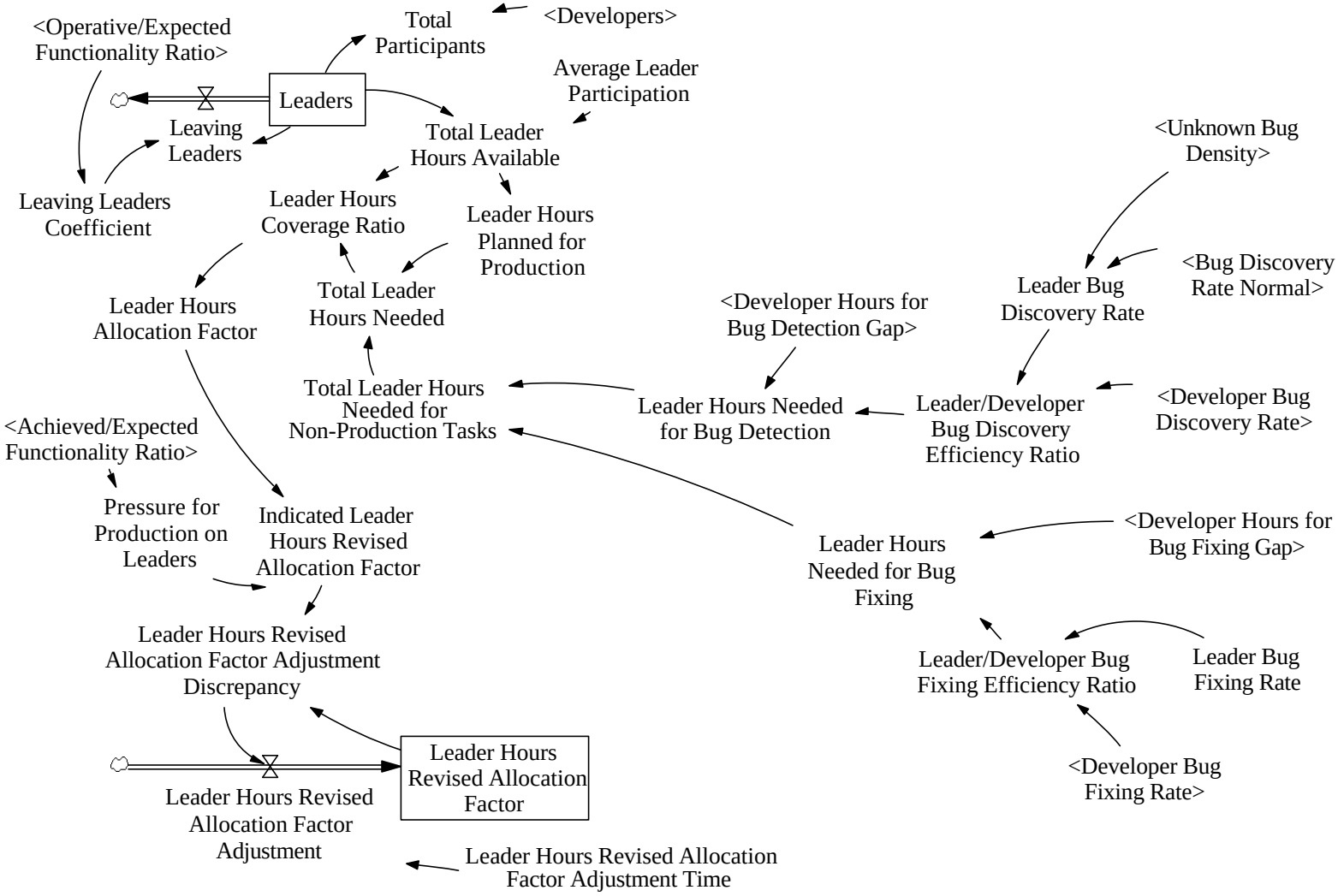


Figure 4.66. OSSD Model (Iteration III) Leaders Time Allocation Sector

Leader hours revised allocation factor determines what percentage of the needed hours for non-production tasks will be allocated. Actual leader hours allocated to production is determined by the difference between the total leader hours available and the total leader hours allocated to non-production tasks. (See Figure 4.67.)

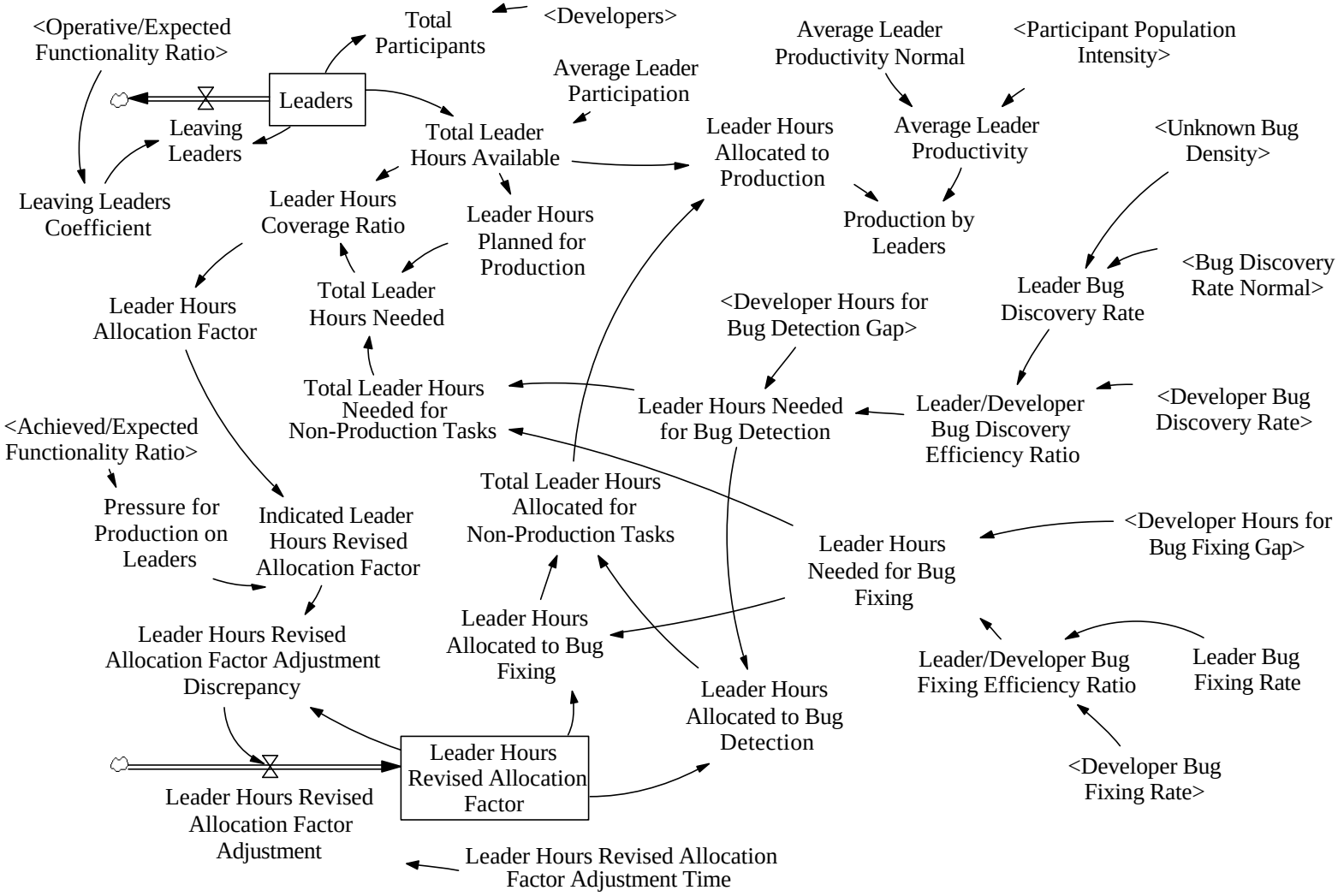


Figure 4.67. OSSD Model (Iteration III) Leaders Time Allocation Sector

Under the base case conditions, Iteration III version displays behaviors similar to those of the previous versions in terms of the main indicators such as product functionality, number of developers, and number of users. (See Figures 4.68 through 4.71.)

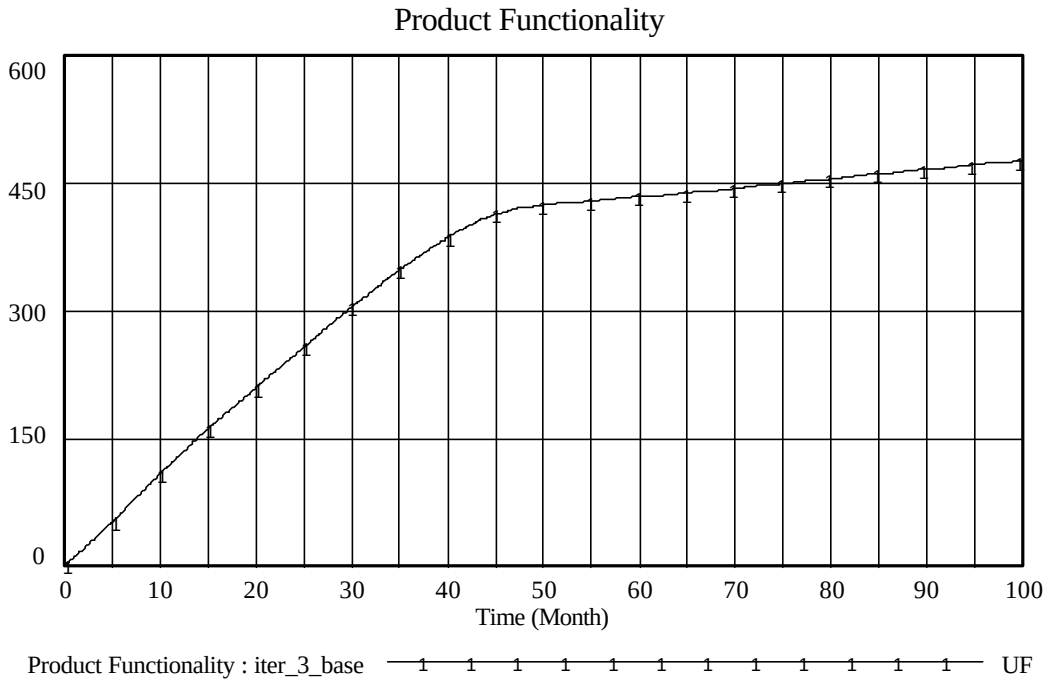


Figure 4.68. OSSD Model (Iteration III) Base Run - Product Functionality

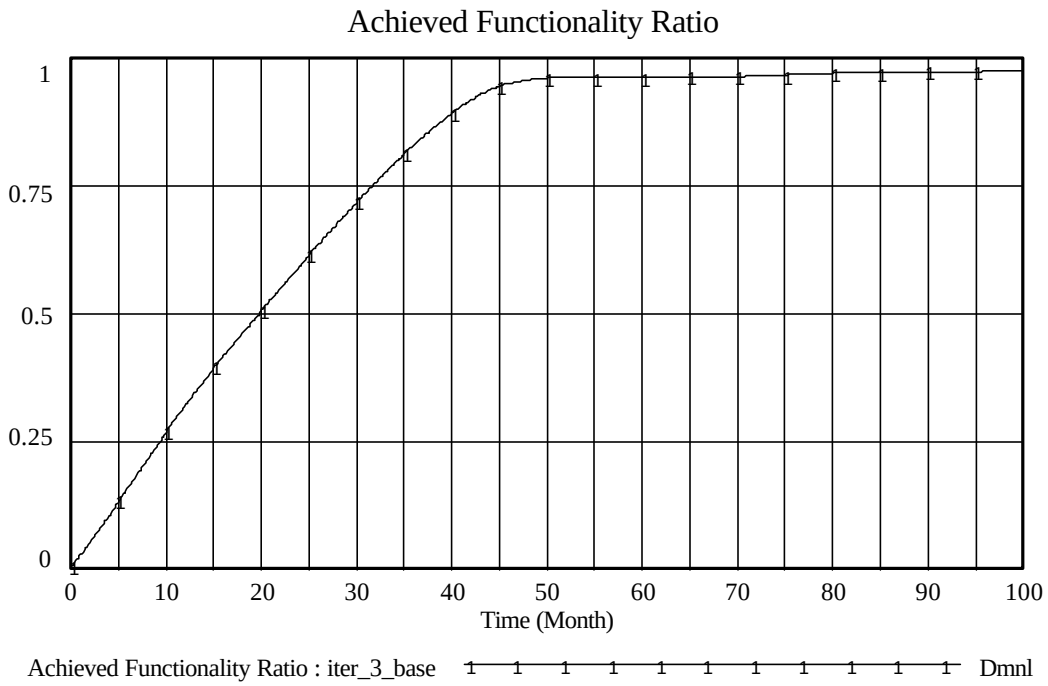


Figure 4.69. OSSD Model (Iteration III) Base Run - Achieved Functionality Ratio

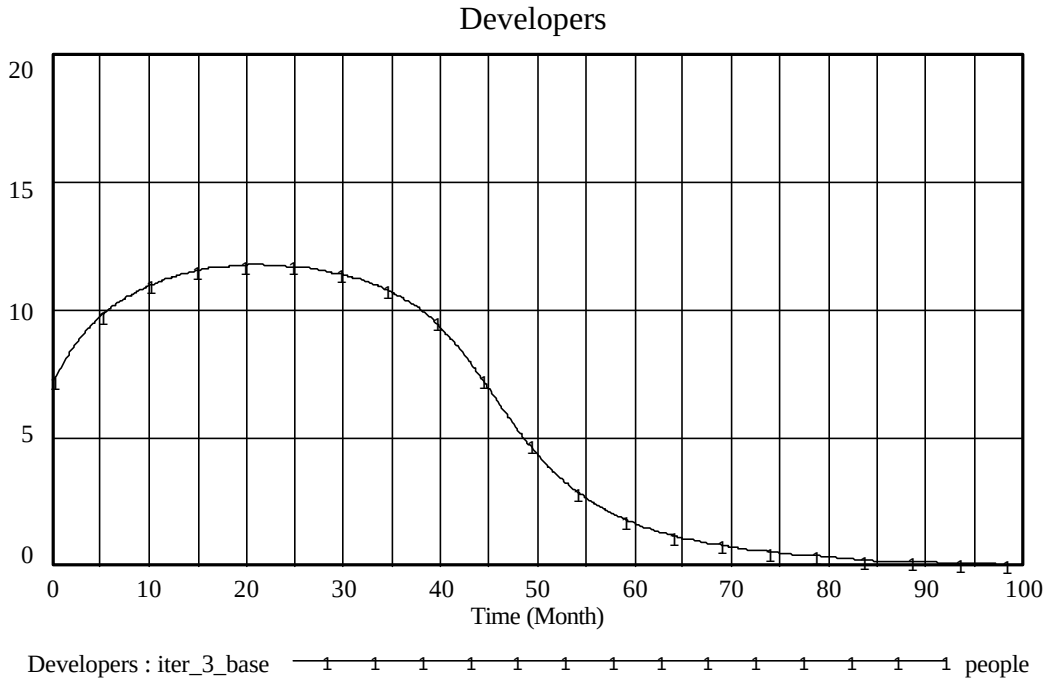


Figure 4.70. OSSD Model (Iteration III) Base Run - Developers

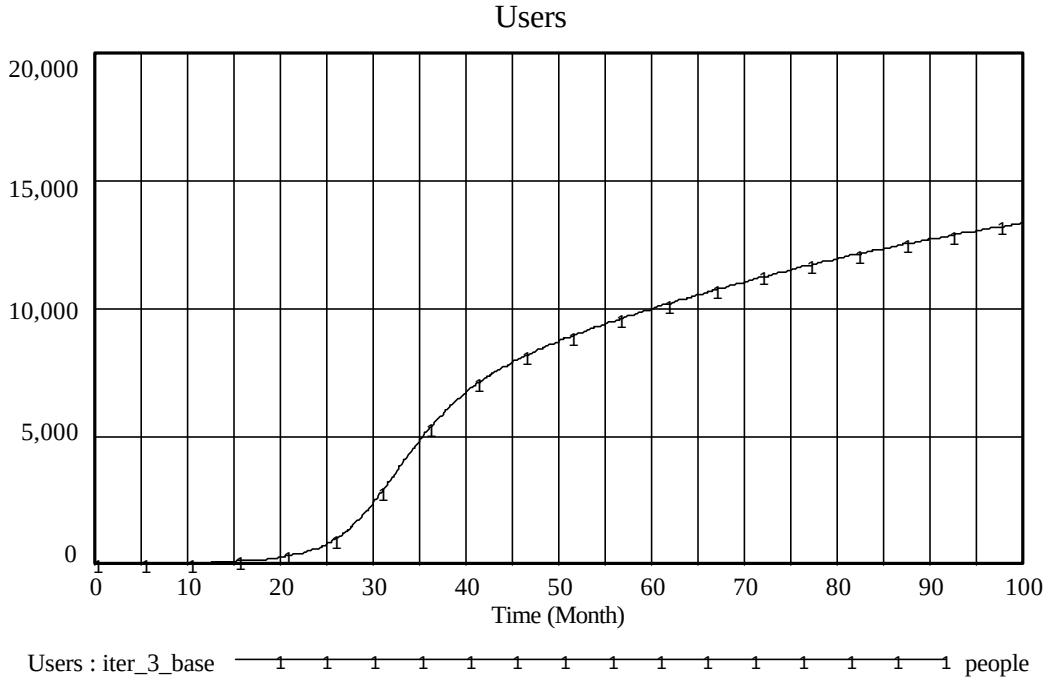


Figure 4.71. OSSD Model (Iteration III) Base Run - Users

As discussed before, maintaining plausible behaviors with respect to those indicators is considered critical while adding more structure to the model. This way, confidence in the model is maintained while adding more explanatory power to it. An important indicator added to the model is the number of total participants, which is the total of number of developers and number of leaders. The behavior of this indicator under the base conditions is shown in Figure 4.72. Here, the number of participants increase during the initial stages of the project, reaching its peak around month 20, and start to decrease from there on, to reach an equilibrium around month 85.

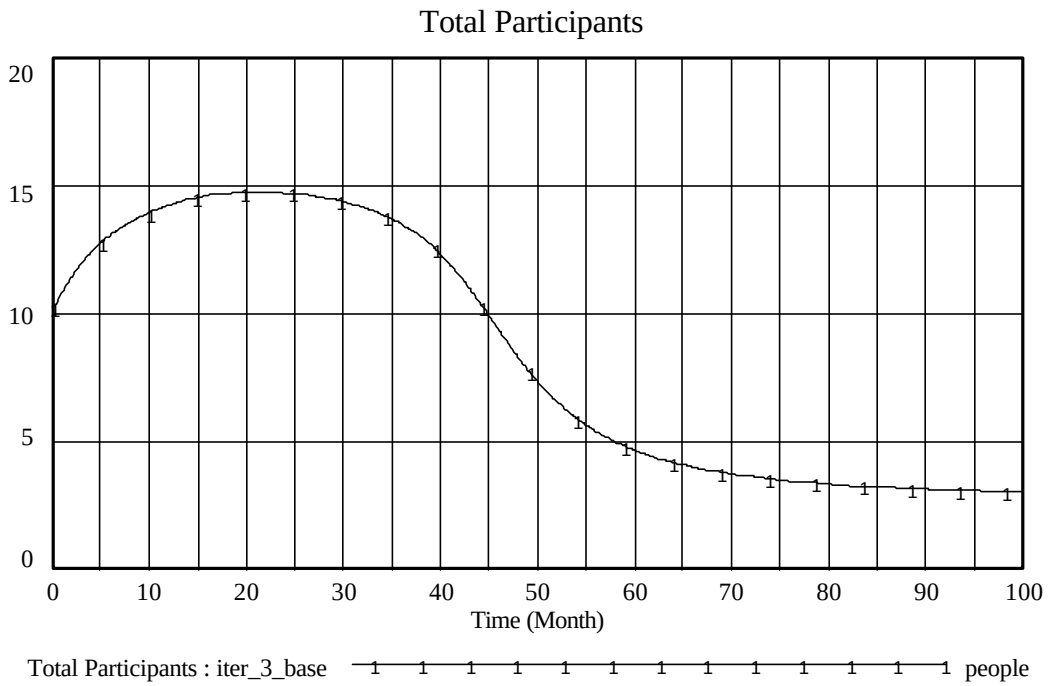


Figure 4.72. OSSD Model (Iteration III) Base Run - Total Participants

Figure 4.73 shows the behavior of total bugs per functionality. This variable increases during the very early stages of the project when a lot of bugs are introduced to the code along with the functionality added to the product. At around month 10, total bugs per functionality starts to decrease, as the developers and leaders start to find and fix many of the bugs, thus bringing the speed of the increase in the number of bugs below the speed of the increase in functionality. That way, even though the total number of bugs continues to increase as shown in Figure 4.74, bugs per functionality decreases, approaching the acceptable level of bugs per functionality. (See Figure 4.73.) Consequently, the severity of the total bugs problem starts to decrease and the perceived quality of the product starts to increase around month 10. (See Figure 4.75 and Figure 4.76.)

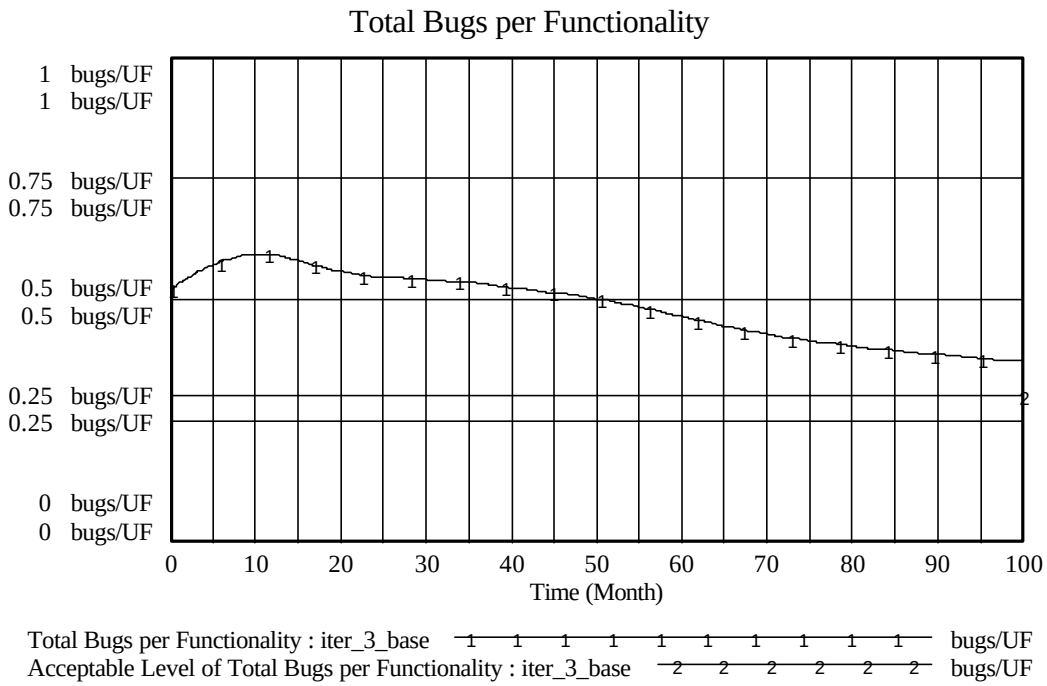


Figure 4.73. OSSD Model (Iteration III) Base Run - Total Bugs per Functionality

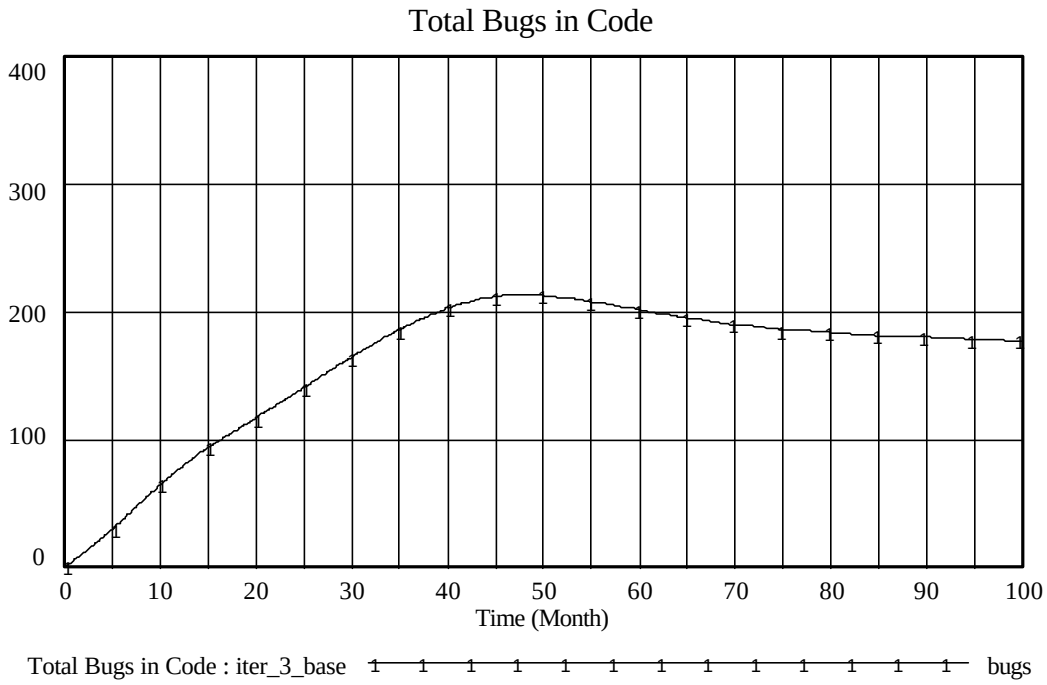


Figure 4.74. OSSD Model (Iteration III) Base Run - Total Bugs in Code

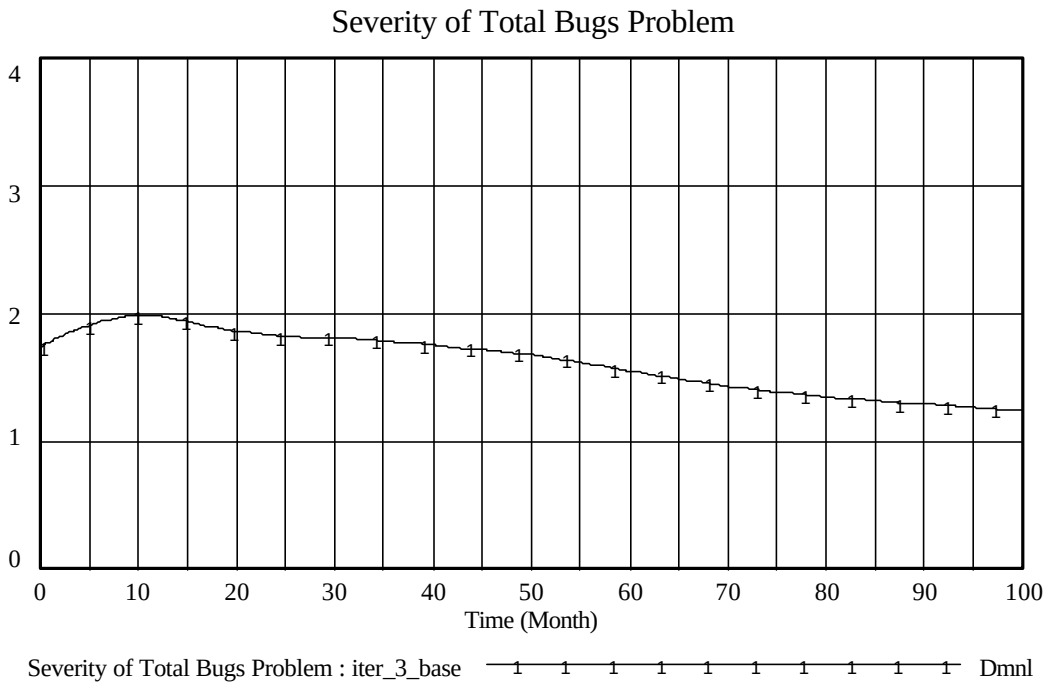


Figure 4.75. OSSD Model (Iteration III) Base Run - Severity of Total Bugs Problem

Figure 4.76. OSSD Model (Iteration III) Base Run - Perceived Product Quality

The behavior of the Iteration III version when the initial limit on product functionality is set high is not different than was found in the previous versions, except that the changes happen much more slowly, and thus the general behavior pattern are "stretched" in time. The main reason for that is that the contributors (developers and leaders) spend a considerable portion of their project time on non-production tasks, namely bug discovery and bug fixing, while the average participation stays the same among versions of the model. This causes the total time allocated to production to decrease in this version of the model, and as a result the functionality growth slows down considerably. Figure 4.77 through Figure 4.80 show the behaviors of product functionality, achieved functionality ratio, number of developers and number of users, respectively, when initial limit on product functionality is set to 4000 UF. By comparing these figures with those of the previous versions, it can be seen that the general behavior patterns stay the same between versions of the model. Figure 4.81 displays the behavior of the number of users under high initial limit on product functionality condition when the simulation horizon is increased to 200 months. It can be seen that the growth pattern is "stretched" over time.

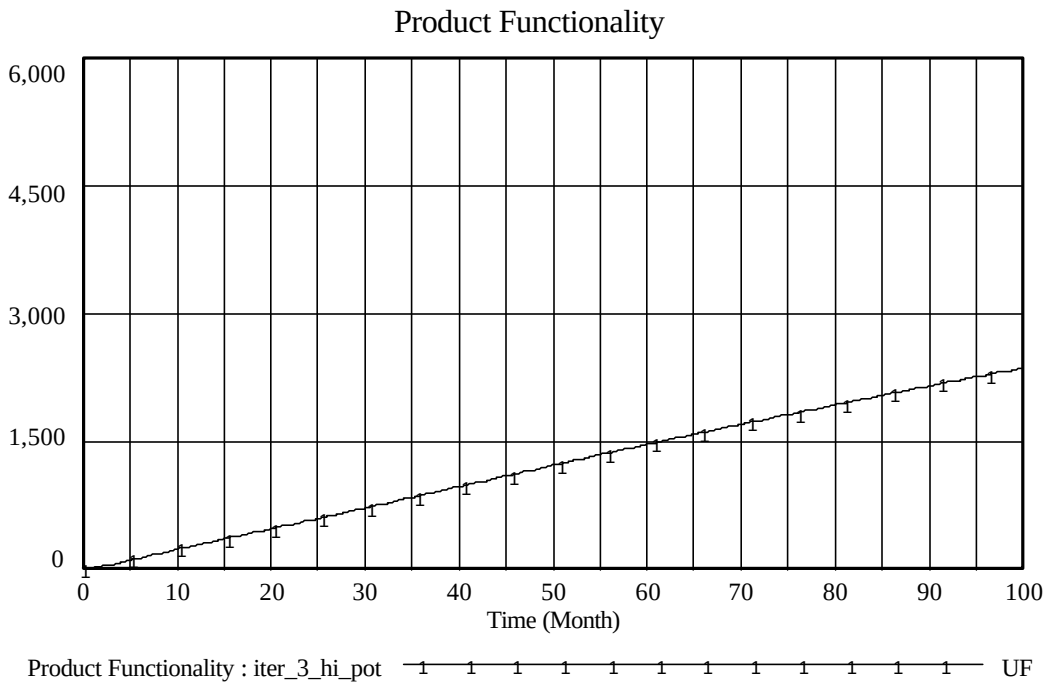


Figure 4.77. OSSD Model (Iteration III) High Potential Run - Product Functionality

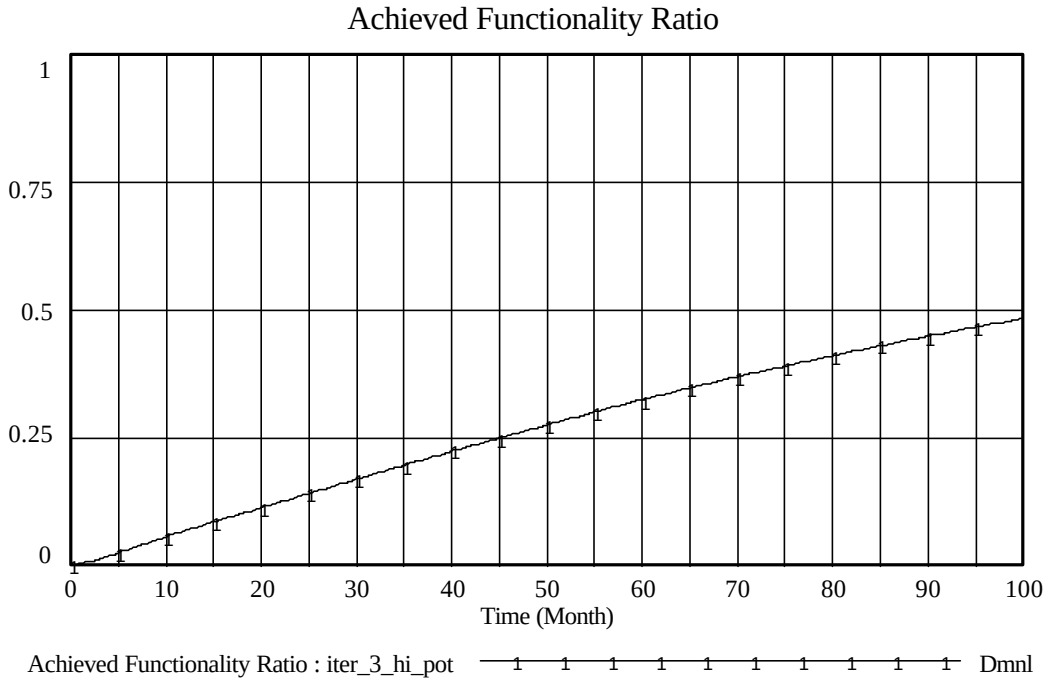


Figure 4.78. OSSD Model (Iteration III) High Potential Run - Achieved Functionality Ratio

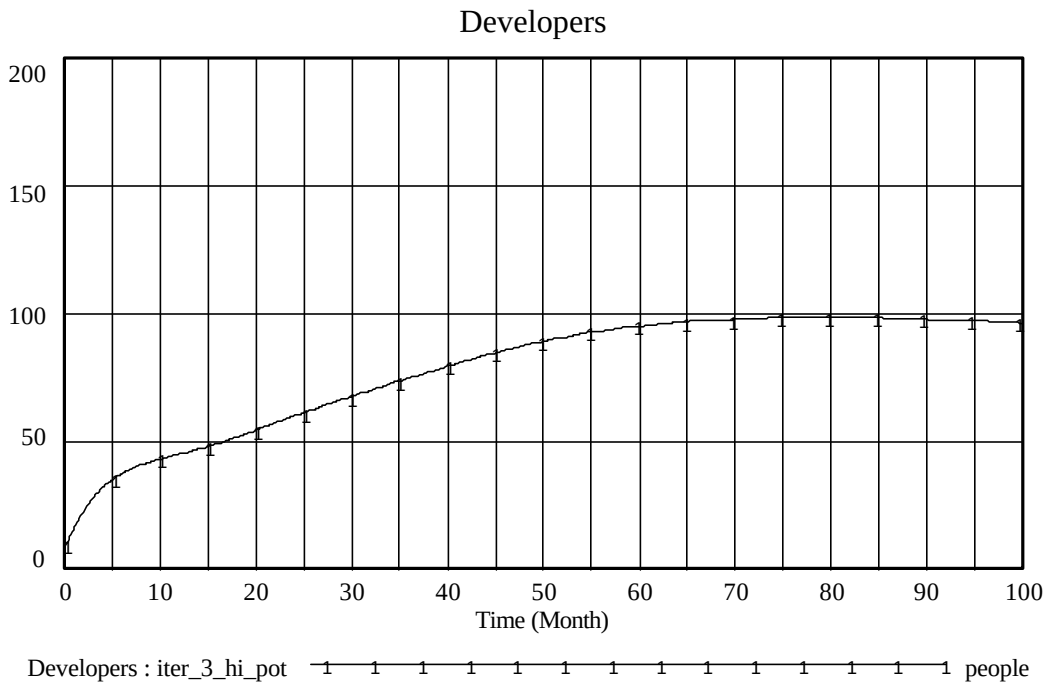


Figure 4.79. OSSD Model (Iteration III) High Potential Run - Developers

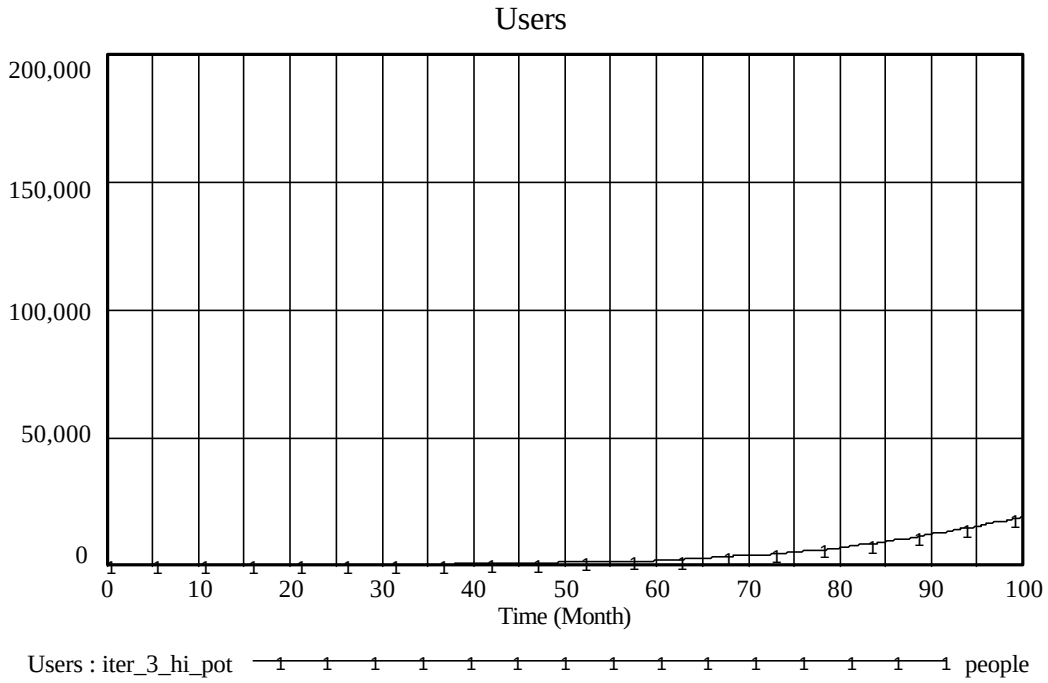


Figure 4.80. OSSD Model (Iteration III) High Potential Run - Users

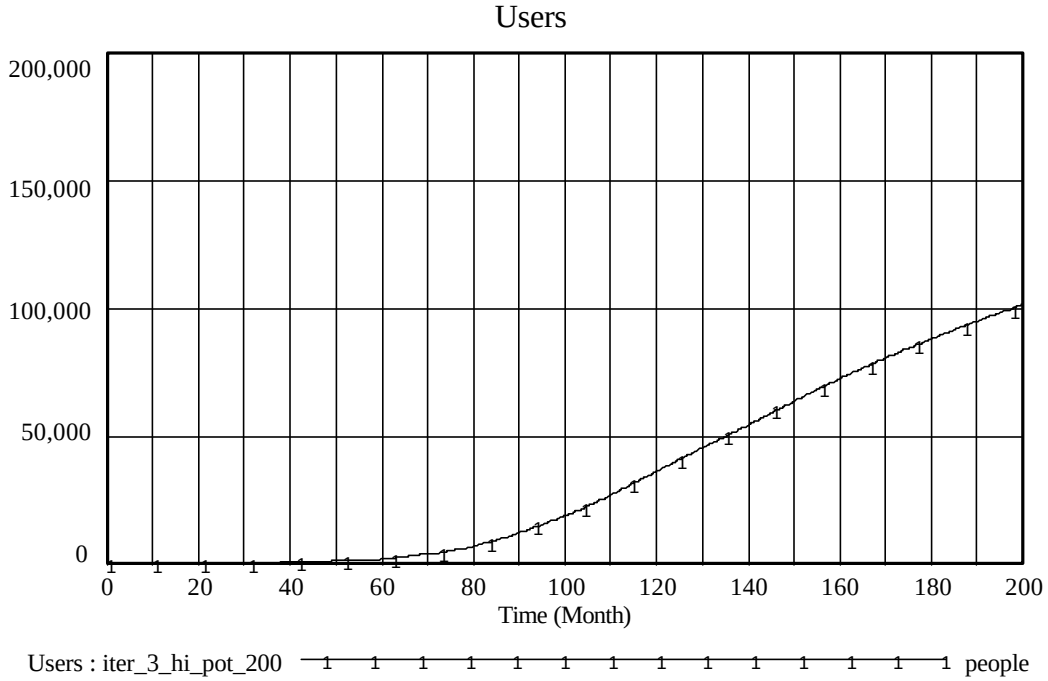


Figure 4.81. OSSD Model (Iteration III) High Potential Run - Users - Time Horizon Doubled

Figure 4.82 shows that total bugs per functionality increases faster and reaches a higher peak under high initial limit on product functionality conditions than it does under the base condition. Also, under this condition total bugs per functionality does not decrease as much as it does under the base case condition, though it decreases faster so the equilibrium it reaches in the long run is higher than that under the base case condition.

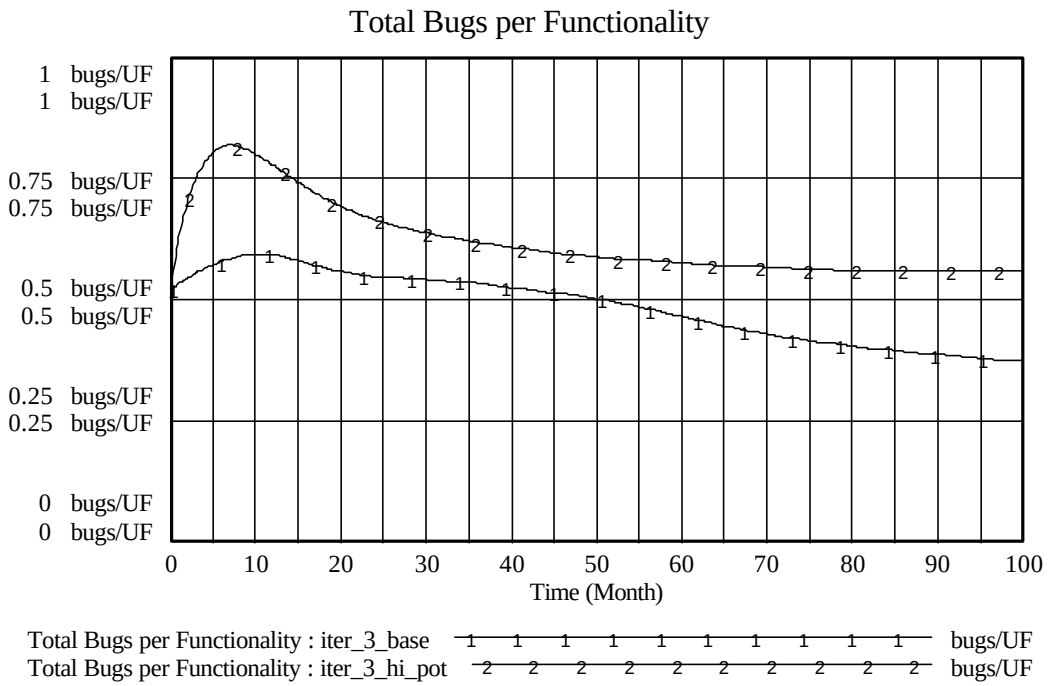


Figure 4.82. OSSD Model (Iteration III) High Potential Run - Total Bugs per Functionality

The severity of total bugs problem too, increases faster, reaches a higher peak, and settles on a higher equilibrium than that under the base case condition. (See Figure 4.83.) Consequently, perceived quality decreases faster, reaches a lower minimum, and converges to a lower equilibrium value than that under the base case condition. (See Figure 4.84.) The main reason for this is the higher developers per leader ratio under the high initial limit on product functionality condition than that under the base case condition. The number of leaders stays the same (three people, in both cases), though the number of developers reaches much higher levels in the high initial limit on product functionality condition.

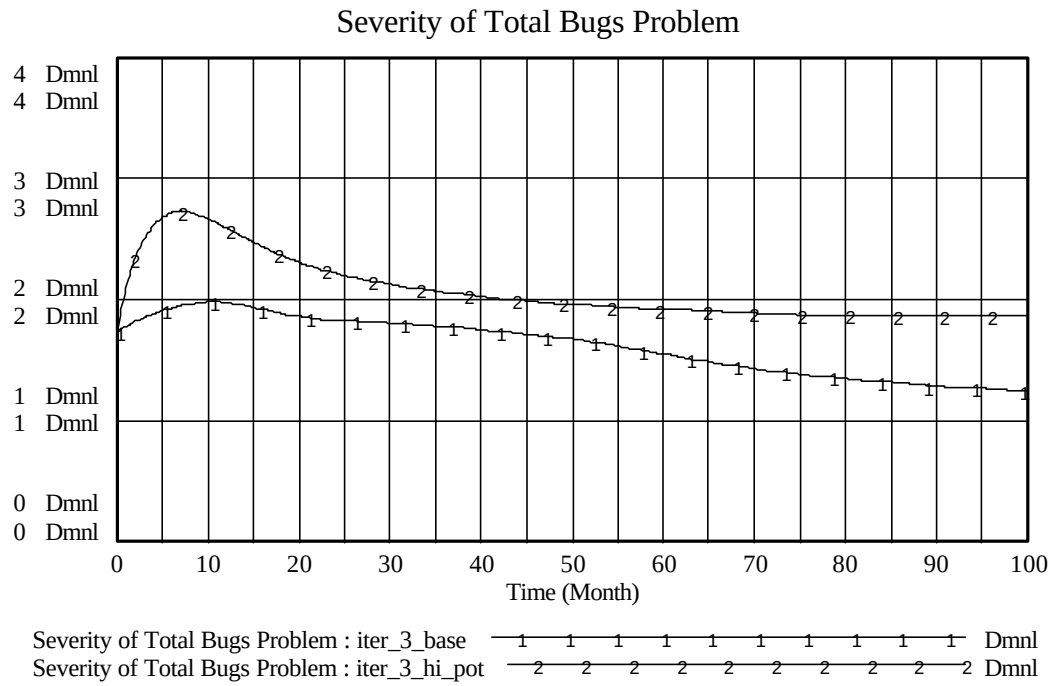


Figure 4.83. OSSD Model (Iteration III) High Potential Run - Severity of Total Bugs Problem

Figure 4.84. OSSD Model (Iteration III) High Potential Run - Perceived Product Quality

A low participation simulation is also run with the Iteration III version of the model. Once again average developer participation is set to 7 hours per month, and the average leader participation is set to 10 hours per month. As observed in Figure 4.85 through Figure 4.90, the community fails to grow under this condition in the Iteration III version, as well. The newly introduced stock of leaders also decline during this run, as the leaders decide to leave the community due to the low achieved functionality ratio. (See Figure 4.90.)

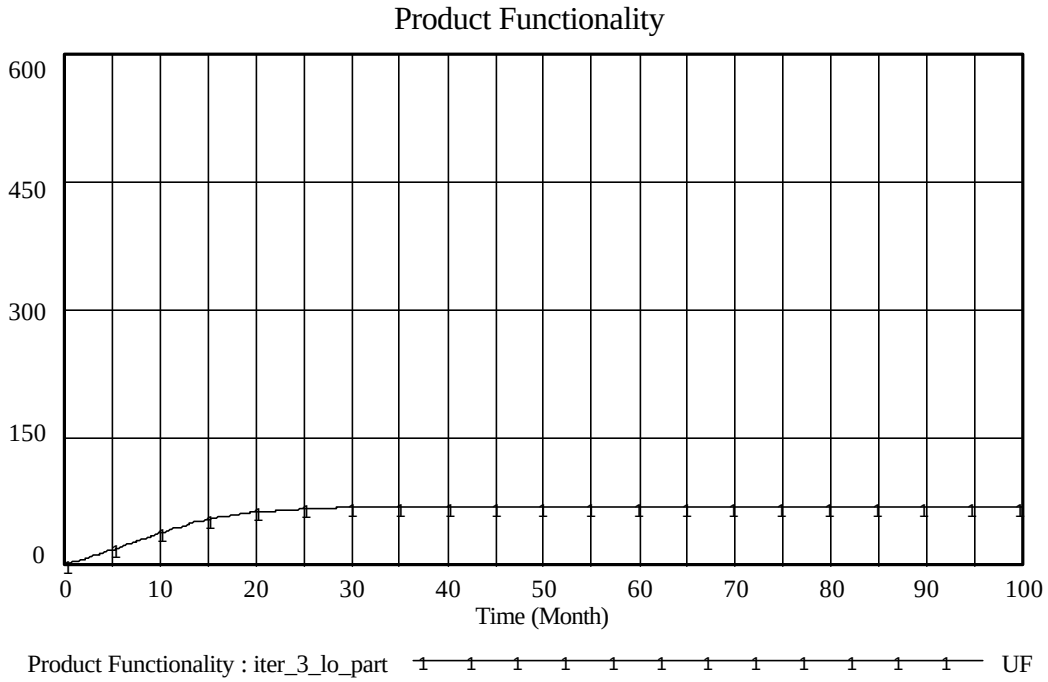


Figure 4.85. OSSD Model (Iteration III) Low Participation Run - Product Functionality

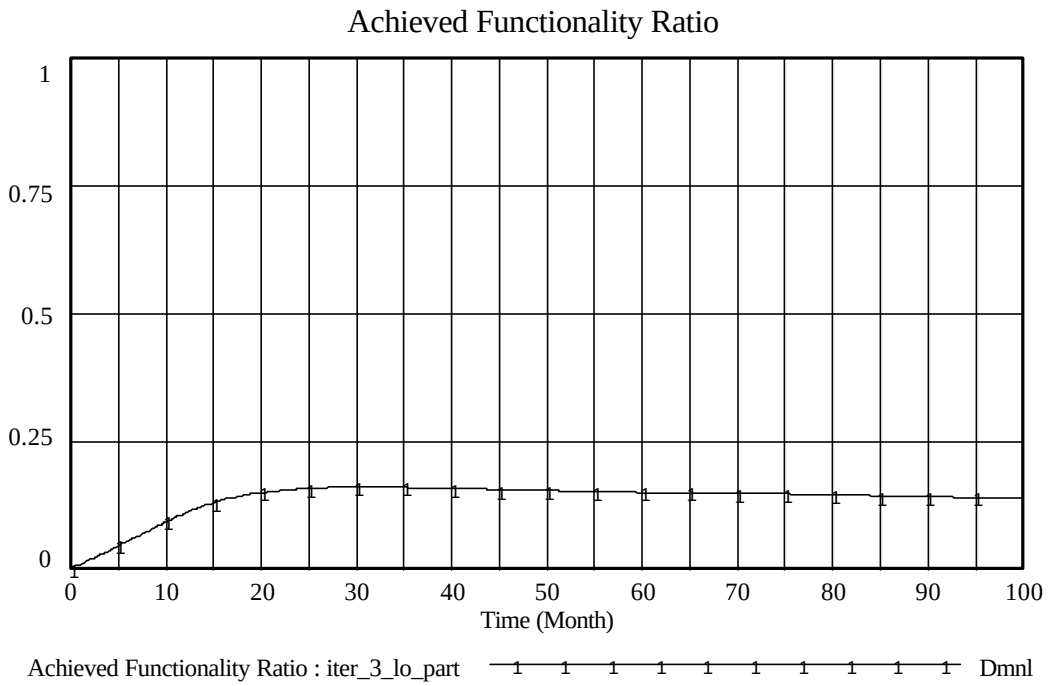


Figure 4.86. OSSD Model (Iteration III) Low Participation Run - Achieved Functionality Ratio

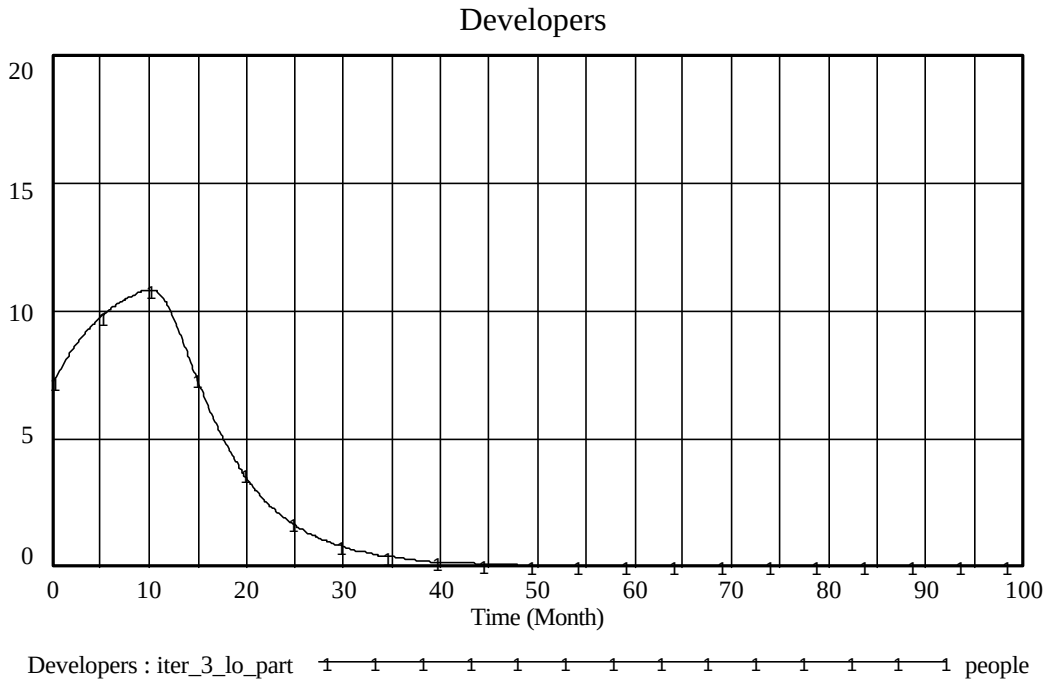


Figure 4.87. OSSD Model (Iteration III) Low Participation Run - Developers

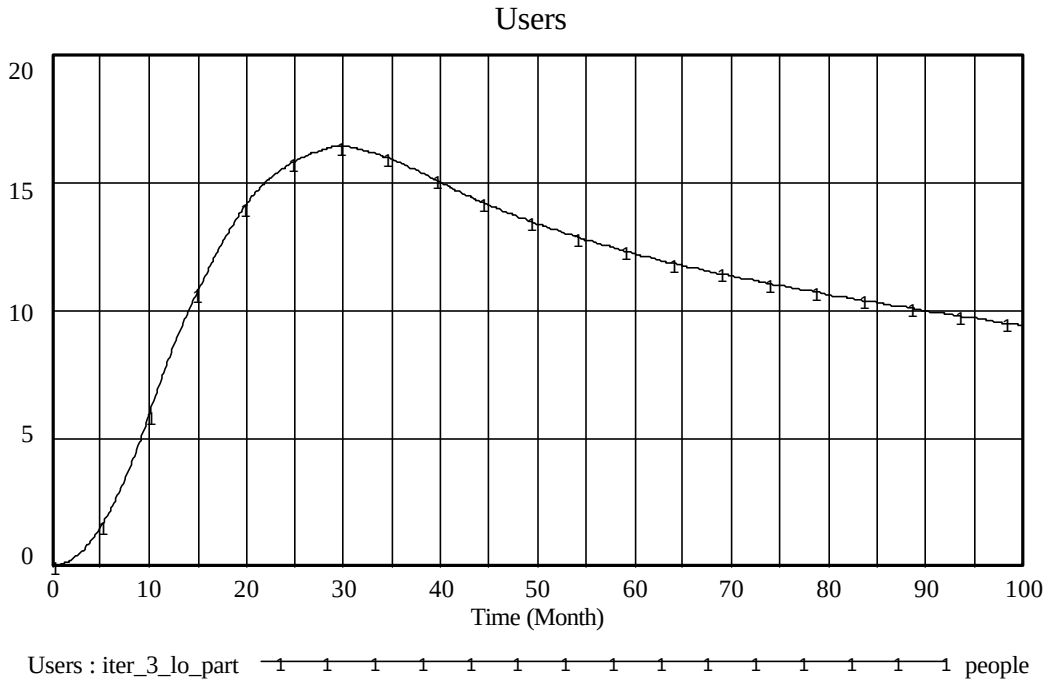


Figure 4. 88. OSSD Model (Iteration III) Low Participation Run - Users

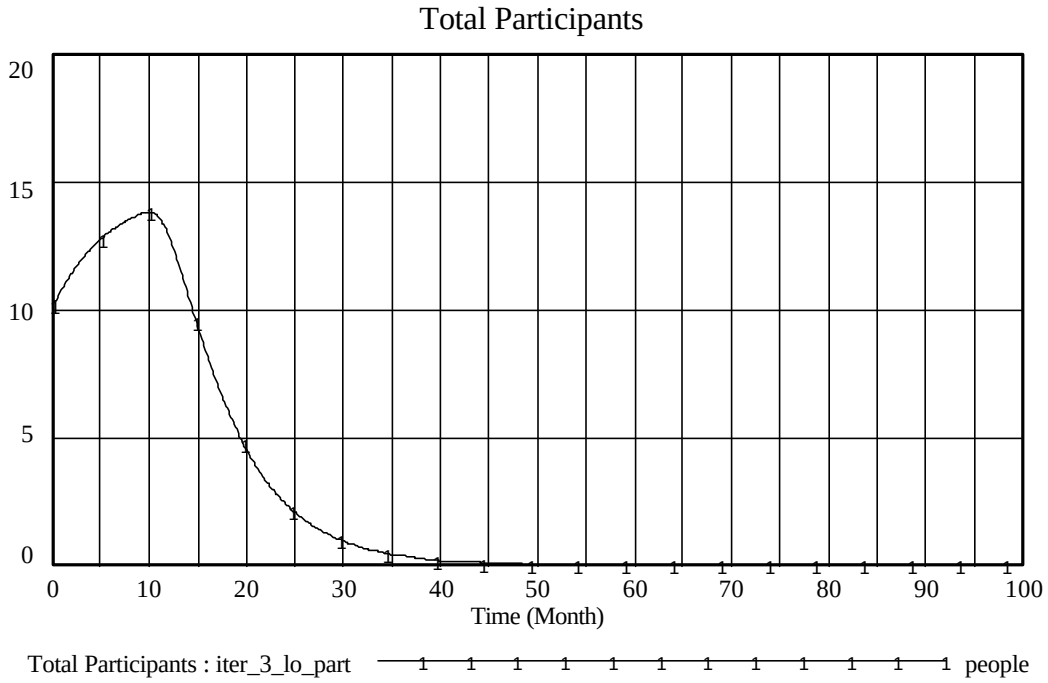


Figure 4.89. OSSD Model (Iteration III) Low Participation Run - Total Participants

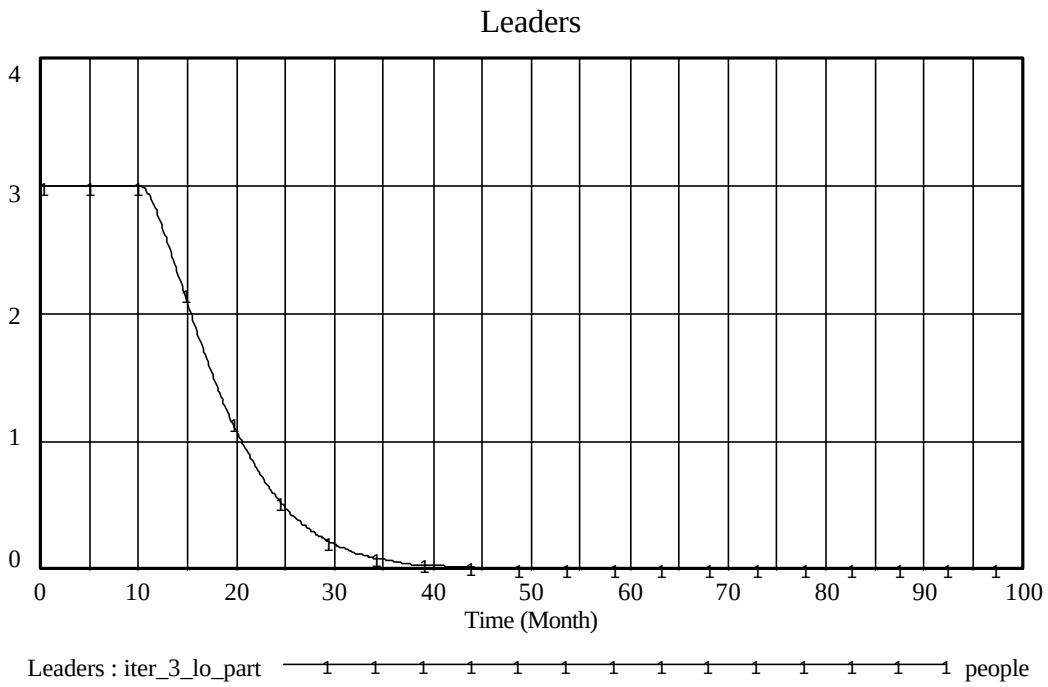


Figure 4. 90. OSSD Model (Iteration III) Low Participation Run - Leaders

Quality-related variables like total bugs per functionality, severity of the total bugs problem and perceived product quality exhibit behaviors very close to those under the base case conditions during the initial stages of the project. However, as developers and leaders start to leave the community in greater numbers, quality related functions suffer just like production, and this causes the quality related variables to reach premature equilibriums which are worse than those under the base case conditions. (See Figure 4.91 through Figure 4.93.)

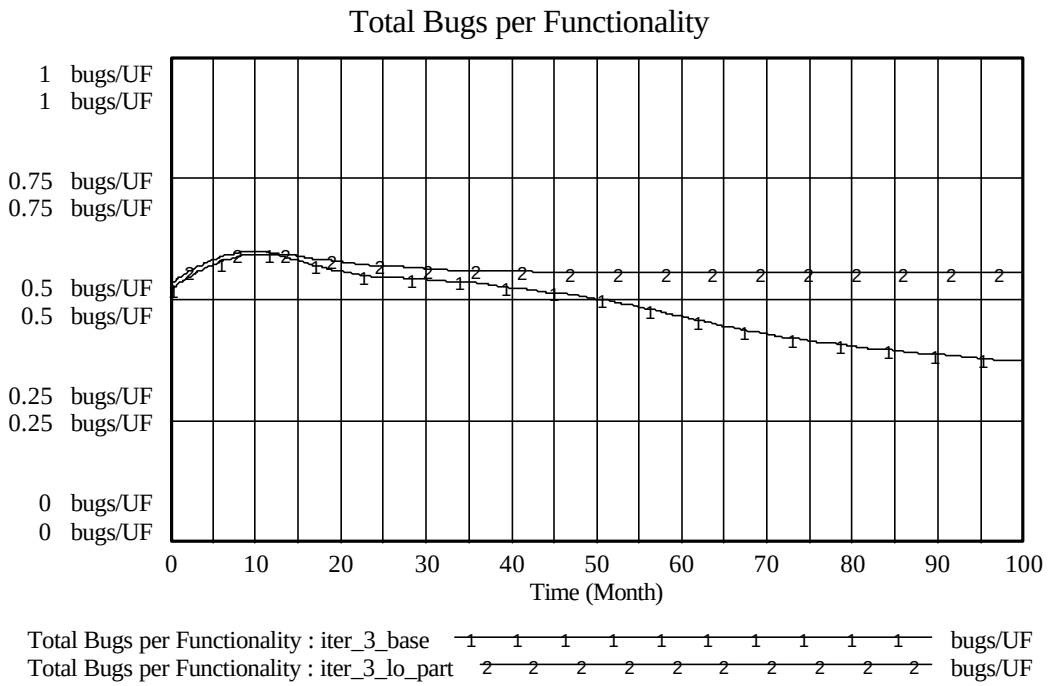


Figure 4.91. OSSD Model (Iteration III) Low Participation Run - Total Bugs per Functionality

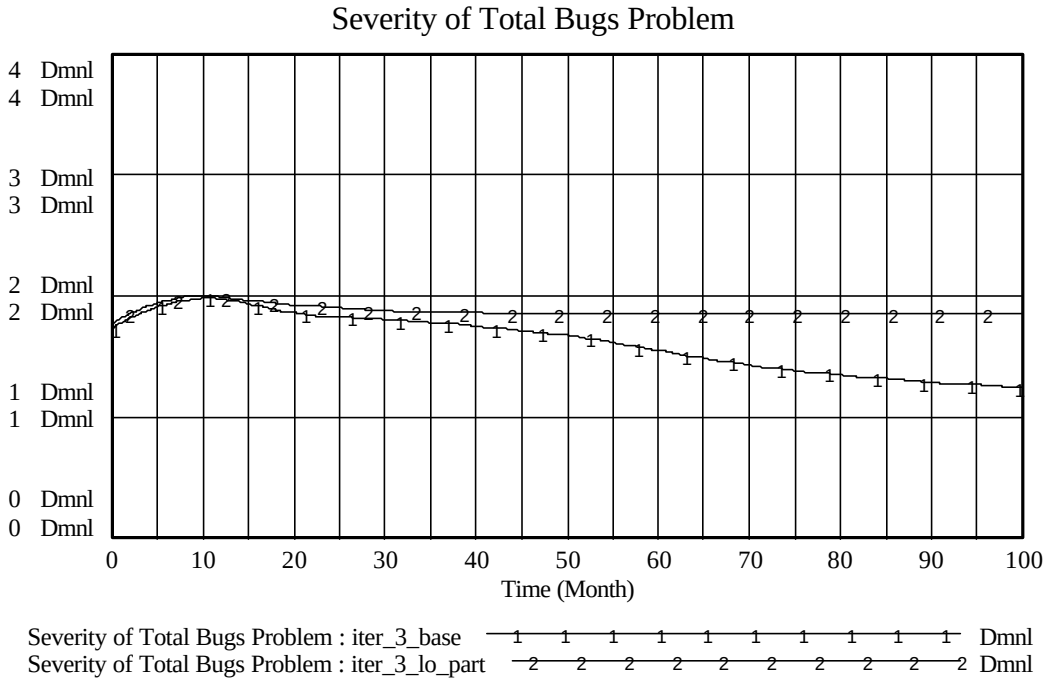


Figure 4.92. OSSD Model (Iteration III) Low Participation Run - Severity of Total Bugs Problem

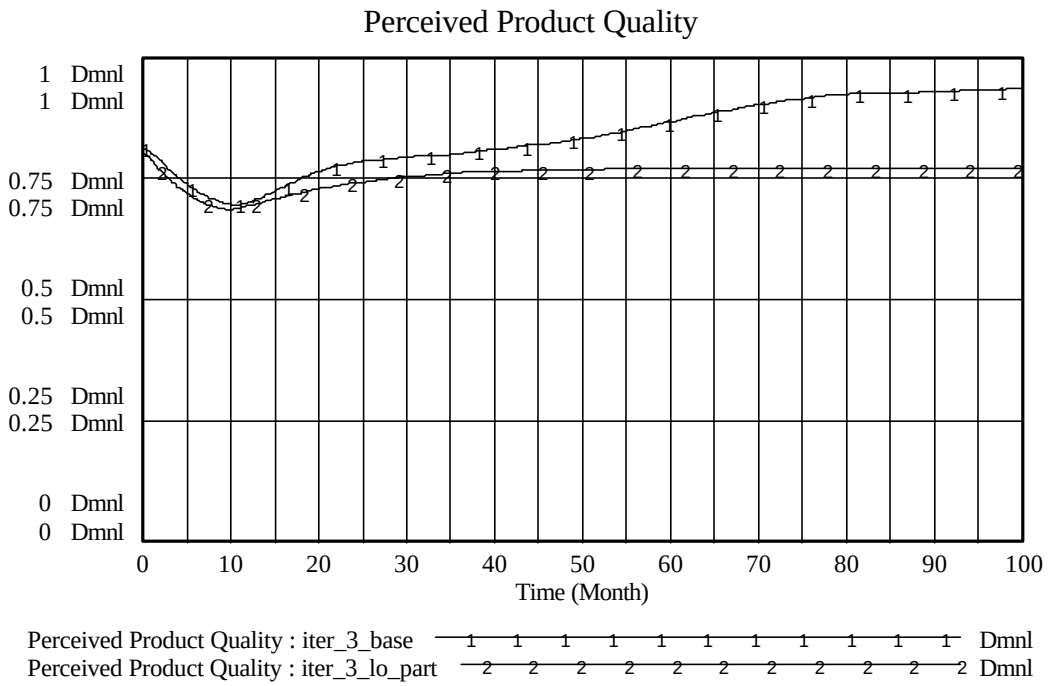


Figure 4.93. OSSD Model (Iteration III) Low Participation Run - Perceived Product Quality

4.5. Iteration IV: Adding Developer Talent

The main change the Iteration IV version of the model introduces is the addition of the Developer Talent sector. Average developer talent is a relative indicator of the overall talent level of the developers with respect to the overall talent level of the leaders, which is defined as an absolute ceiling of talent for the purposes of the model. Average developer talent is conceptualized as a variable that varies between zero and one, zero being the lowest, and one being the highest possible talent level for a developer. The arbitrary name Relative Talent Units is used as the unit for this variable. One RTU represent a talent level that is equal to that of an average leader, thus representing the ceiling for developer talent. As shown in Figure 4.94, average developer talent is in fact an average of the total developer talent pool with respect to the number of developers.

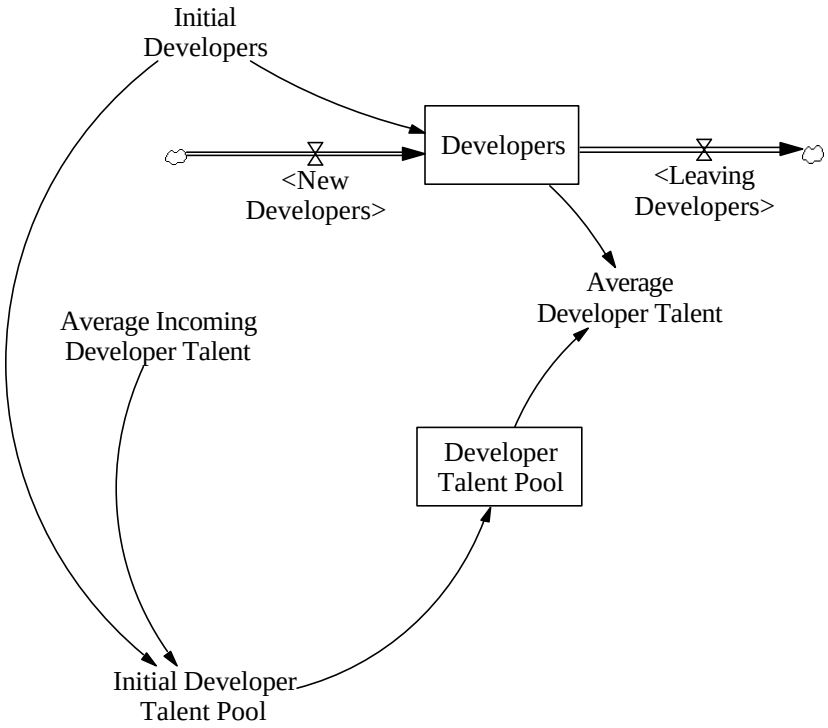


Figure 4.94. OSSD Model (Iteration IV) Developer Talent Sector

As developers join the community their relative talents are added to the developer talent pool through the developer talent gained inflow. It is assumed that each new developer has a relative talent level of 0.5 RTU, at the time of joining. A certain amount of developer talent is lost as developers leave the community. It is assumed that a leaving developer will take away an amount of talent that is equal to the average developer talent at the time of leaving. That is represented with the developer talent lost outflow. (See Figure 4.95.)

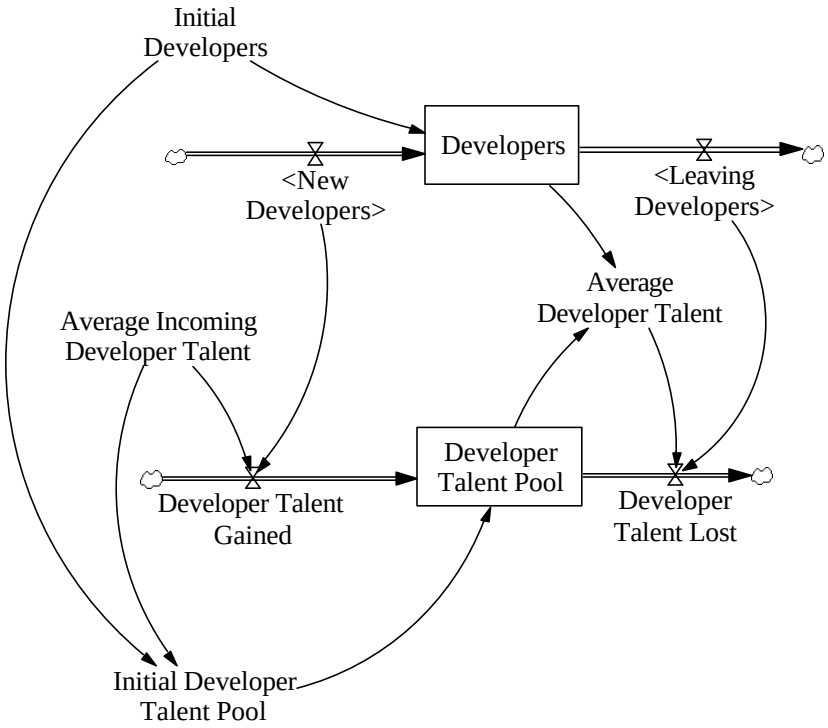


Figure 4.95. OSSD Model (Iteration IV) Developer Talent Sector

It is also possible to build developer talent by coaching developers, which is added to the developer talent pool through the developer talent built inflow. Coaching takes place as leaders train developers. As Figure 4.96 shows, the difference between the actual average developer talent and the maximum developer talent level indicates an average developer talent building opportunity. Here the maximum developer talent is assumed to be equal to average leader talent, which is 1 RTU. So in effect, average developer talent building opportunity is equal to the difference between the actual average developer talent and the average leader talent. Average developer talent building opportunity indicates a pressure for talent building. A higher developer talent building opportunity indicates a higher pressure for talent building, and that in turn indicates a certain number of coaching hours per developer, which is the basis for the total number of developer hours needed for coaching. (See Figure 4.96.)

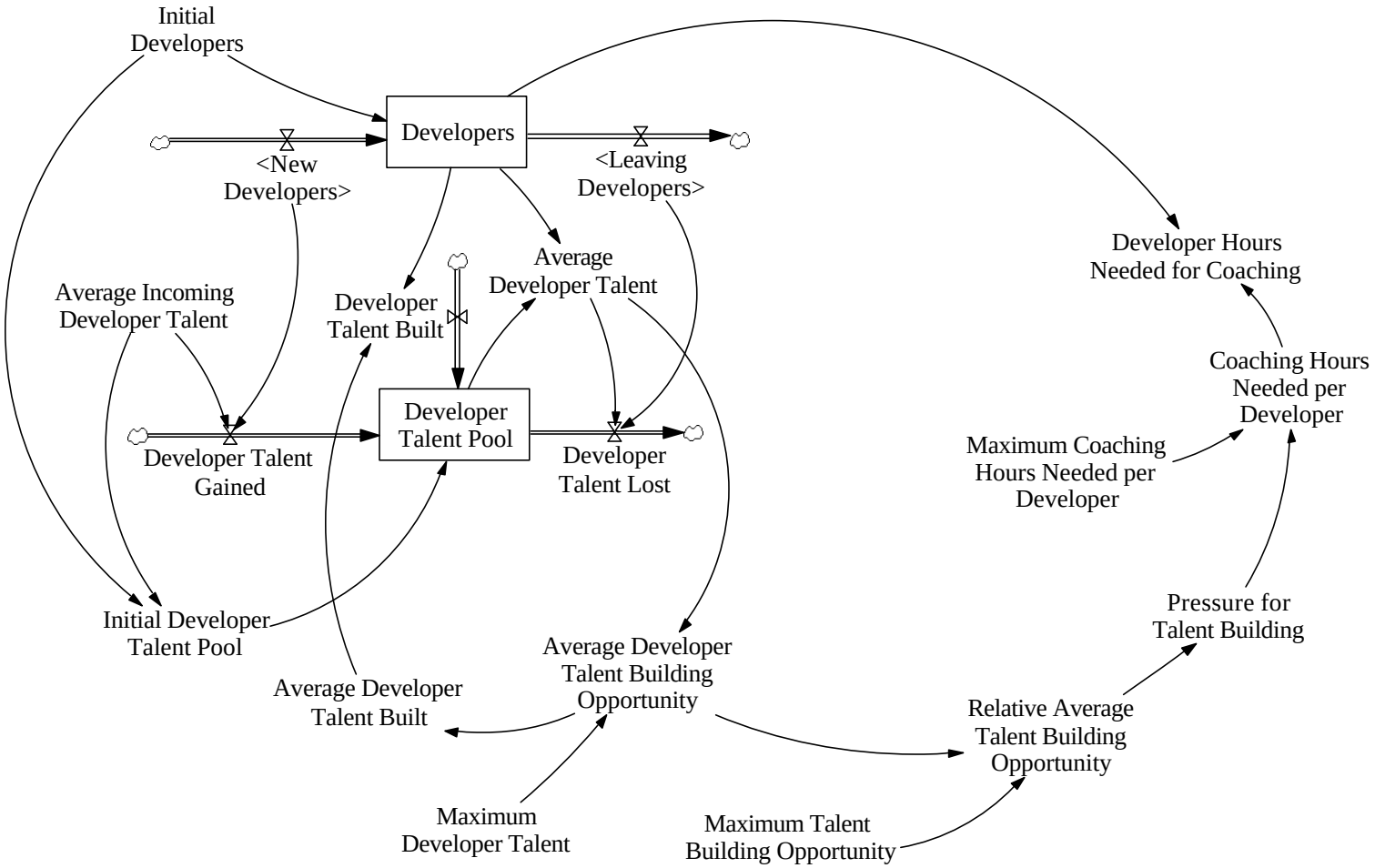


Figure 4.96. OSSD Model (Iteration IV) Developer Talent Sector

Leader hours allocated to coaching sets an upper limit for available coaching hours. Developer hours needed for coaching translates into developer hours planned for coaching as much as the coaching hours availability ratio permits. Developer hours allocation factor indicates what percentage of coaching hours planned is actually allocated for coaching. Allocated coaching hours per developer indicates the amount of talent built per developer, which when multiplied by the number of developers gives the total developer talent built in a given period. (See Figure 4.97.)

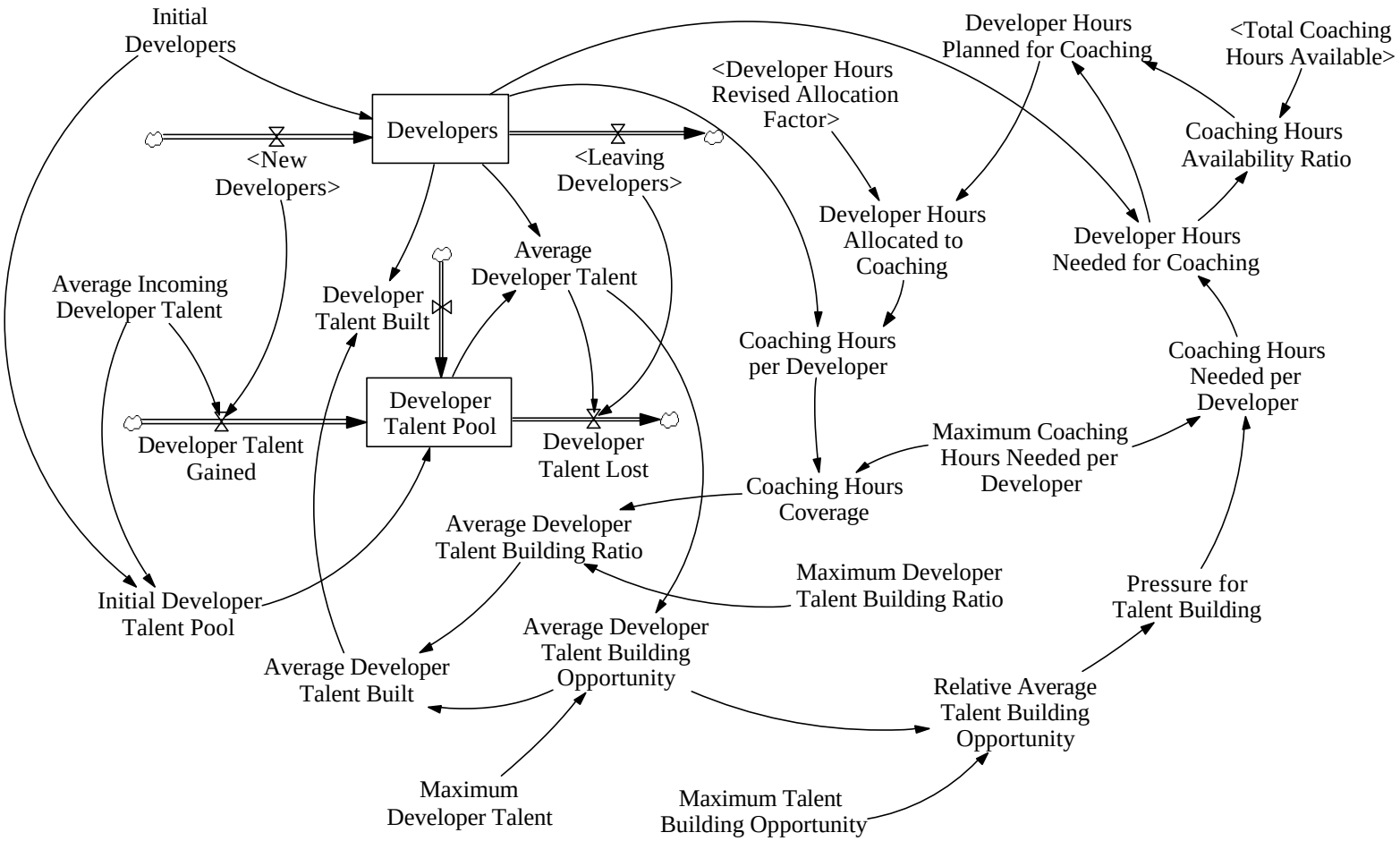


Figure 4.97. OSSD Model (Iteration IV) Developer Talent Sector

The behavior of the Iteration IV model under base case conditions is mostly similar to the behavior of the previous version. Once again, more explanatory power is added to the model with new structure, without losing plausible behavior. As Figures 4.98 through 4.104 demonstrate, the behaviors of the main indicators have stayed roughly the same from Iteration III to Iteration IV.

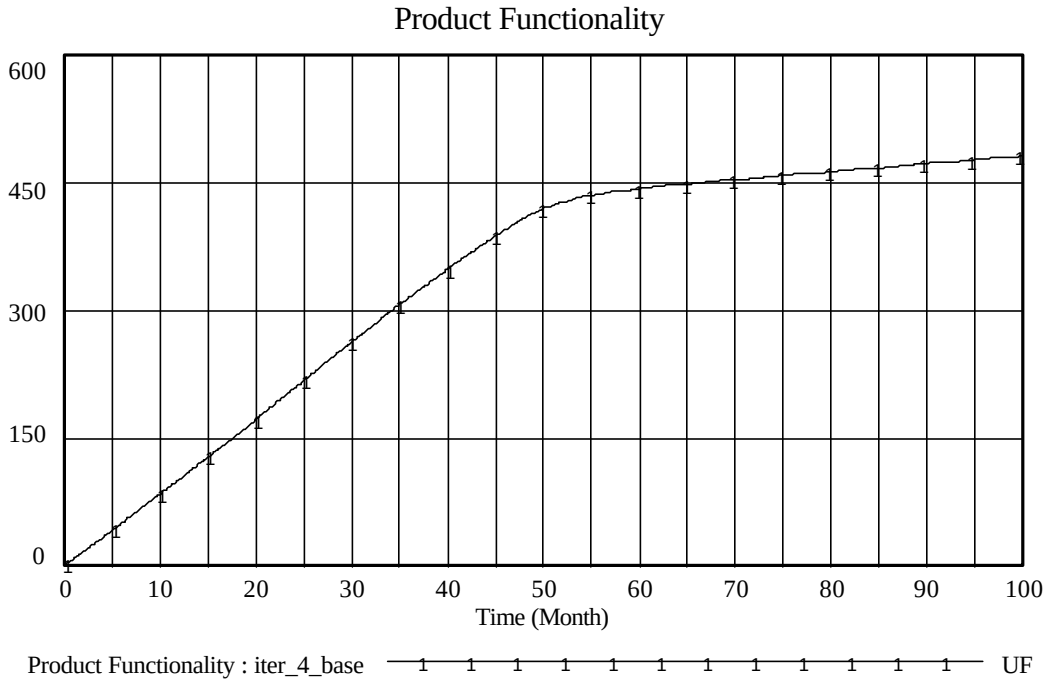


Figure 4.98. OSSD Model (Iteration IV) Base Run - Product Functionality

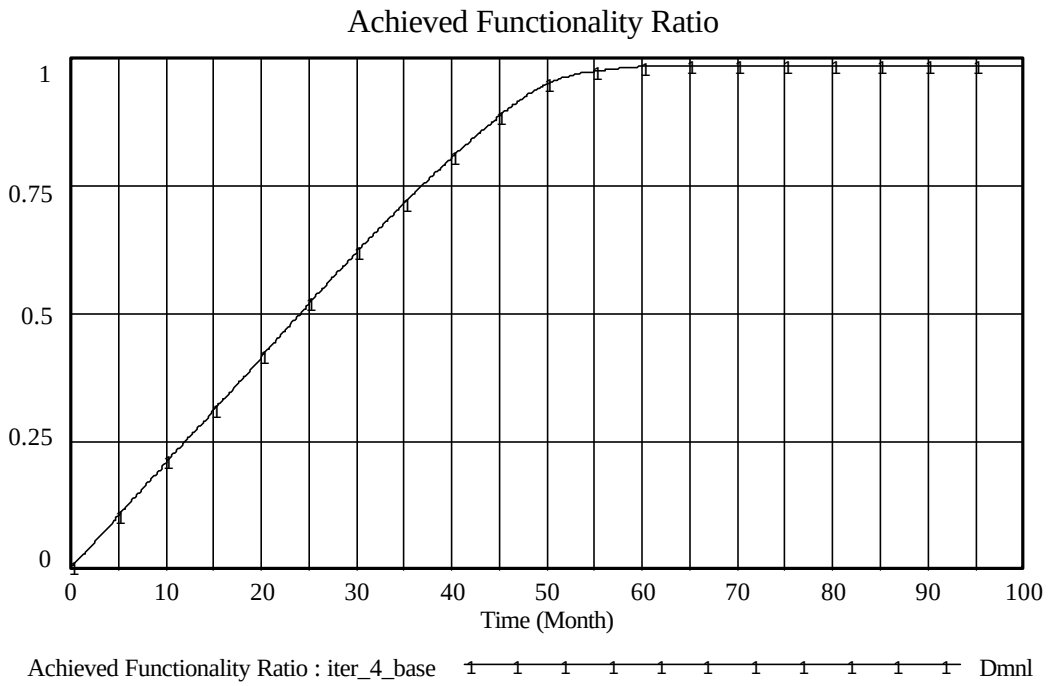


Figure 4.99. OSSD Model (Iteration IV) Base Run - Achieved Functionality Ratio

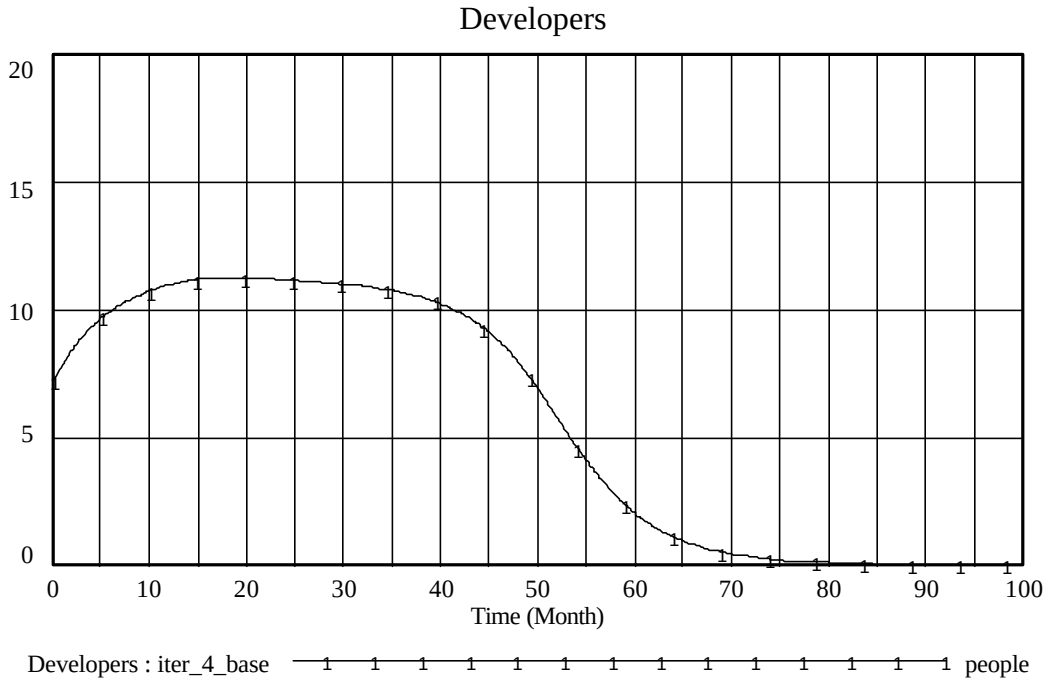


Figure 4.100. OSSD Model (Iteration IV) Base Run - Developers

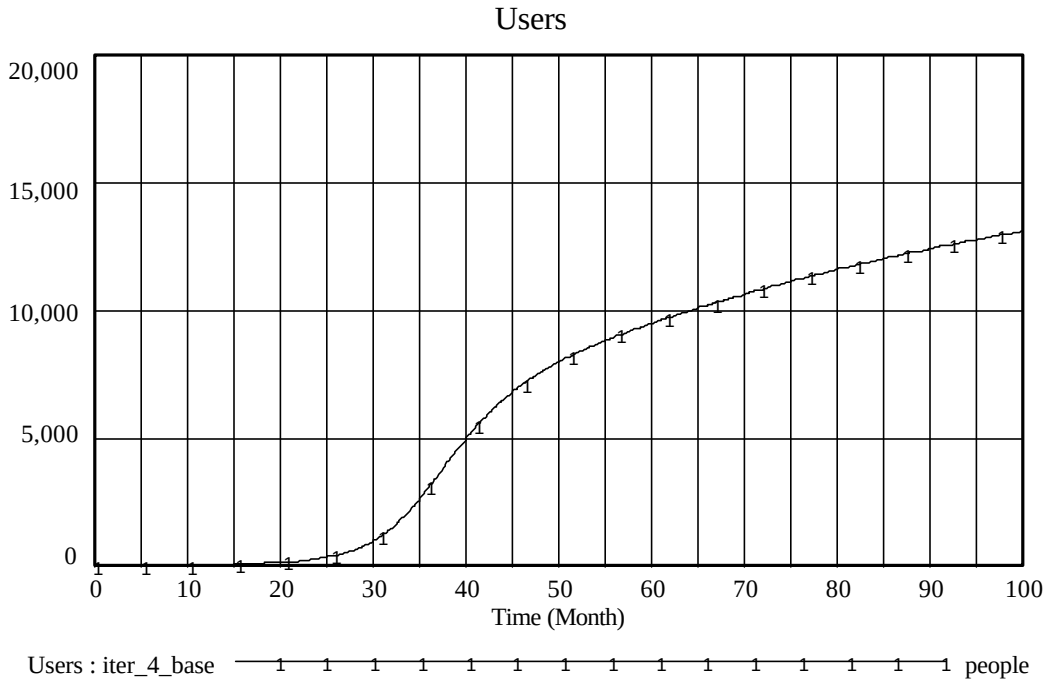


Figure 4.101. OSSD Model (Iteration IV) Base Run - Users

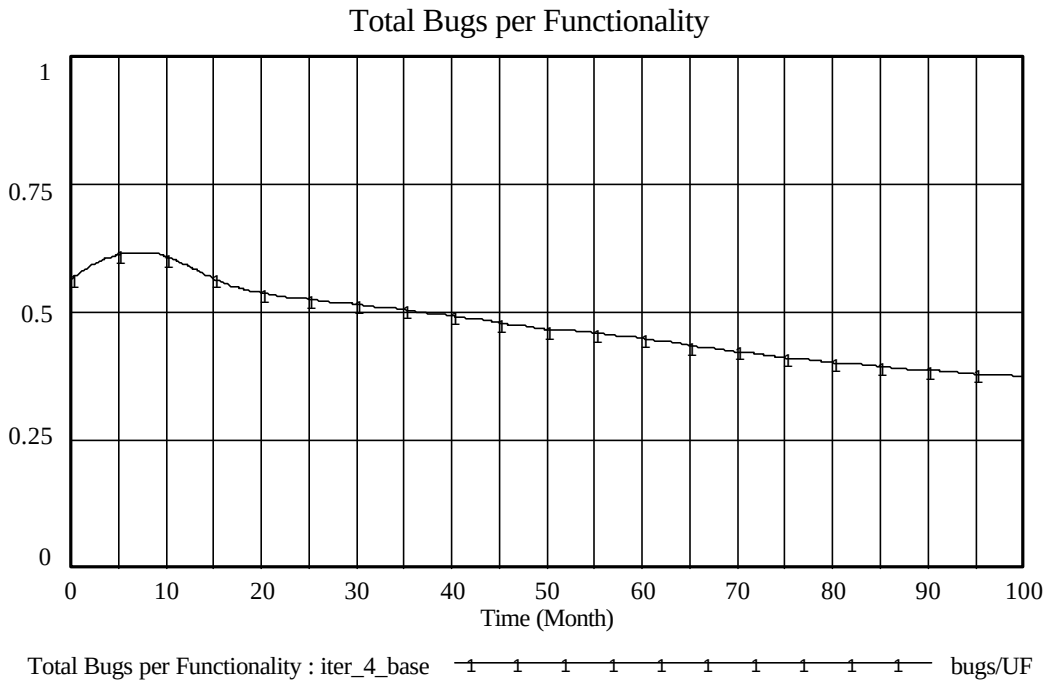


Figure 4.102. OSSD Model (Iteration IV) Base Run - Total Bugs per Functionality

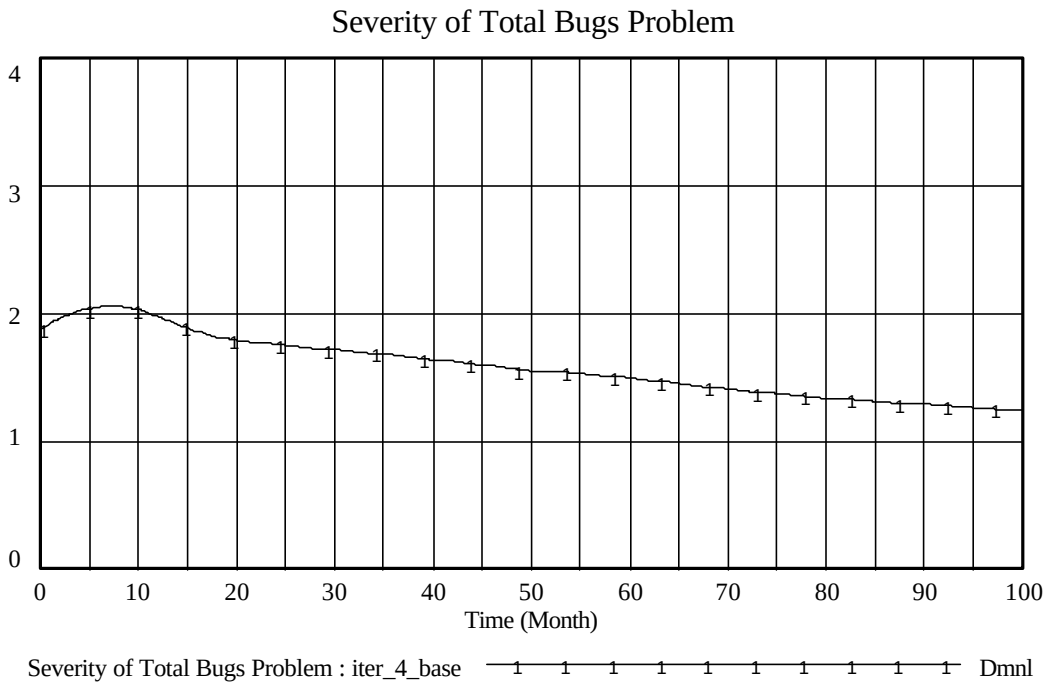


Figure 4.103. OSSD Model (Iteration IV) Base Run - Severity of Total Bugs Problem

Figure 4.104. OSSD Model (Iteration IV) Base Run - Perceived Product Quality

The behavior of the newly introduced variable Average Developer Talent is portrayed in Figure 4.105. The average talent of the developers starts at .5 RTU, since that is the default talent for incoming developers, and all the developers are considered newcomers at the beginning of the project. The average talent gradually increases throughout the project as the leaders coach developers thus adding new talent to the overall talent pool. Average Developer Talent reaches .75 RTU by the end of the simulation horizon of 100 months.

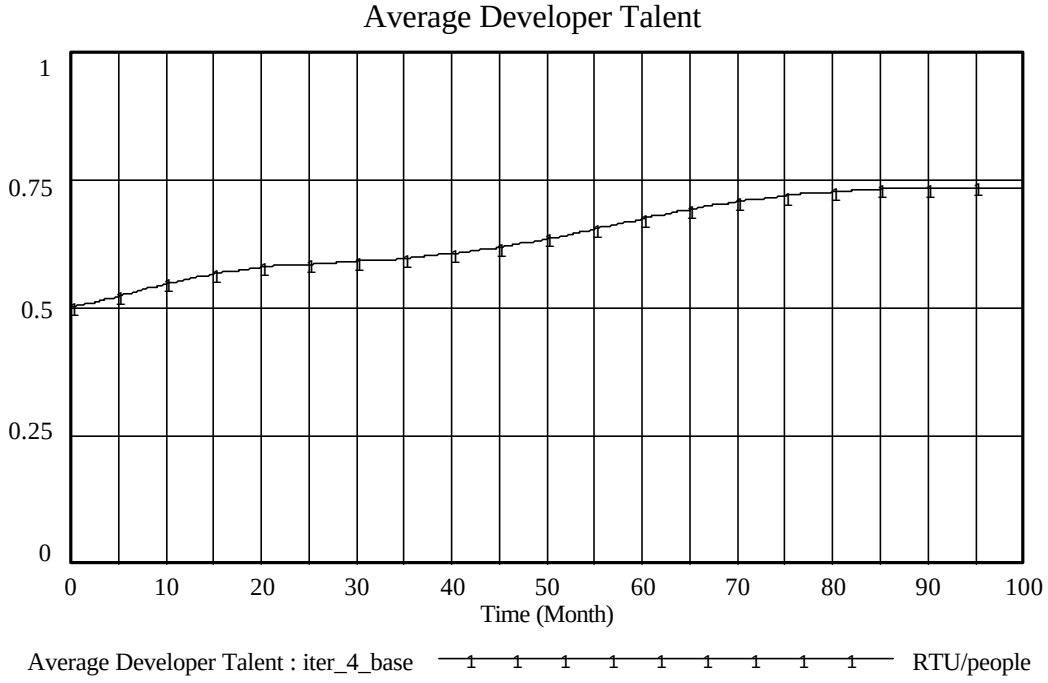


Figure 4.105. OSSD Model (Iteration IV) Base Run - Average Developer Talent

The behavior of the Iteration IV model is similar to that of the previous version under the two alternative conditions, high initial limit on product functionality, and low participation. Figures 4.106 through 4. 113 portray the model behavior under high initial limit on product functionality condition, while Figures 4.114 through 4.123 show the behavior of the model under low participation condition. The behavior of the newly

introduced variable Average Developer Talent is different under the two alternative conditions then that under the base condition. (See Figures 4.113 and 4.123.) In both cases, average developer talent remains almost flat throughout the simulation horizon of 100 months, but due to different reason in each case. In the high initial limit on product functionality case, the number of developers becomes too many for the available number of leaders for effective coaching. Therefore each developer gets an almost negligible amount of coaching, and that does not produce considerable improvement in developer talent. On the other hand, under low participation condition, the available developer hours are so low that they can only cover the basic development needs, leaving developers a negligible amount of time for coaching, which results in practically no improvement in the average developer talent.

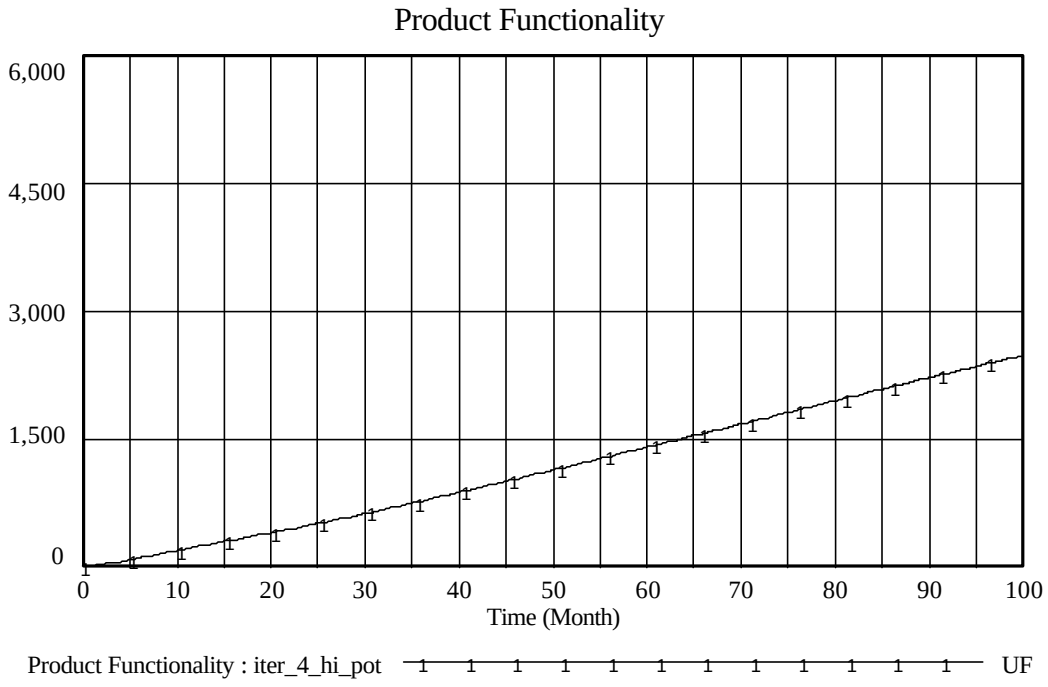


Figure 4.106. OSSD Model (Iteration IV) High Potential Run - Product Functionality

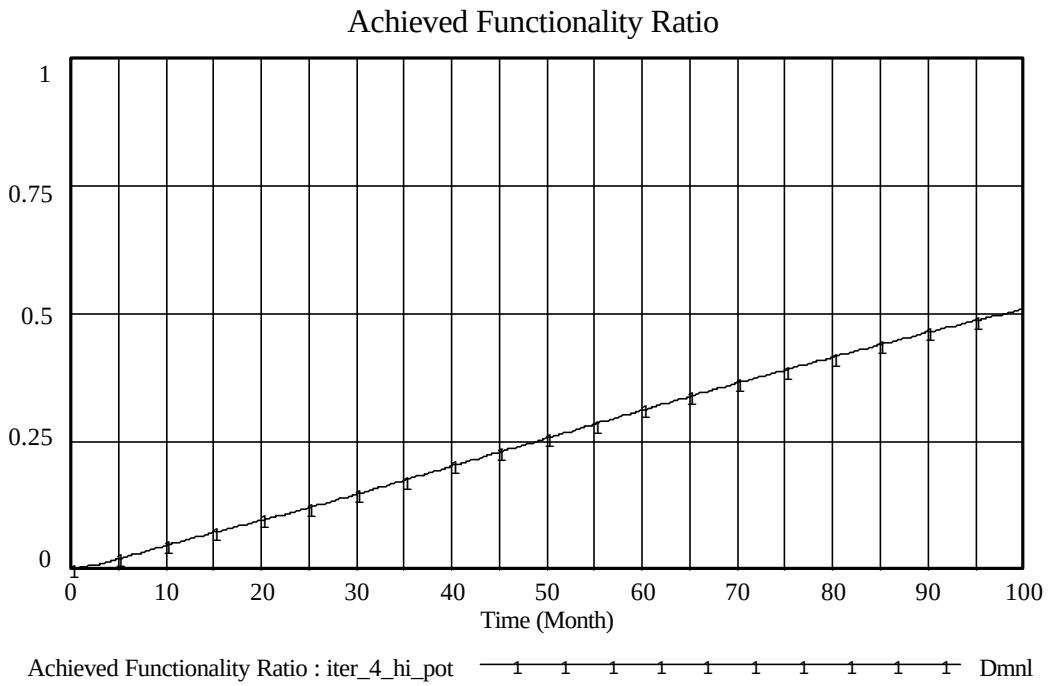


Figure 4.107. OSSD Model (Iteration IV) High Potential Run - Achieved Functionality Ratio

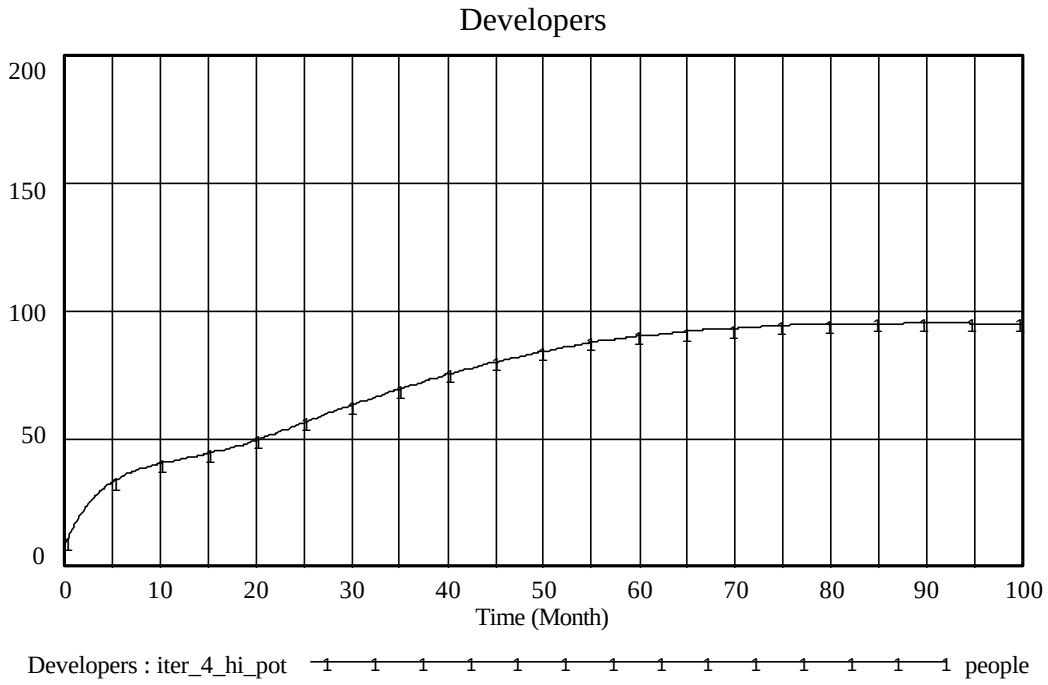


Figure 4.108. OSSD Model (Iteration IV) High Potential Run - Developers

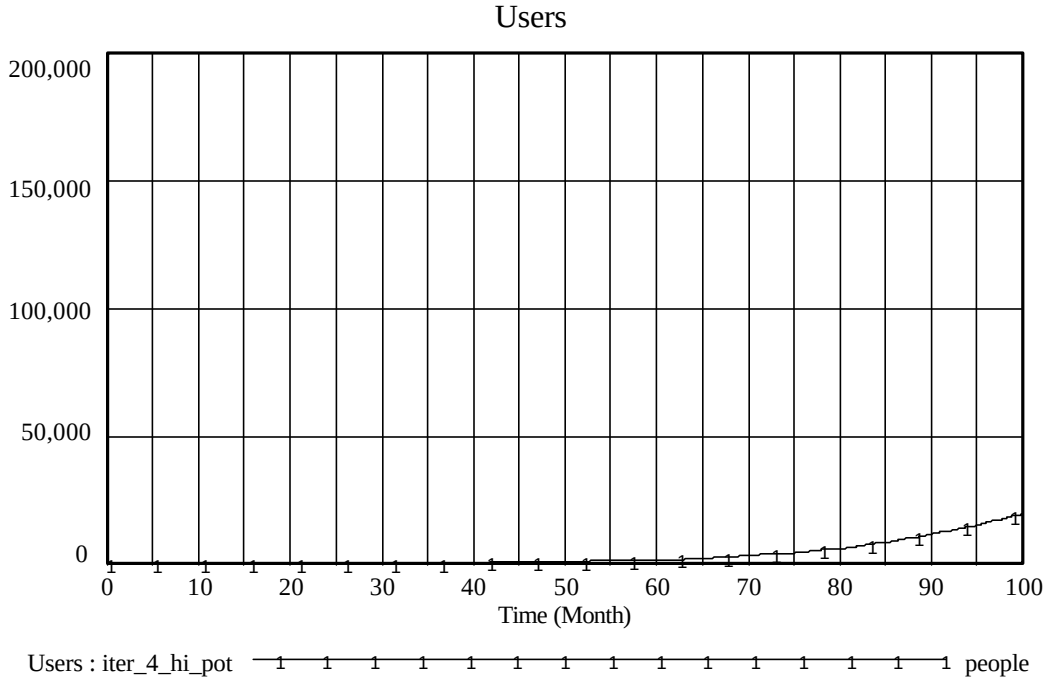


Figure 4.109. OSSD Model (Iteration IV) High Potential Run - Users

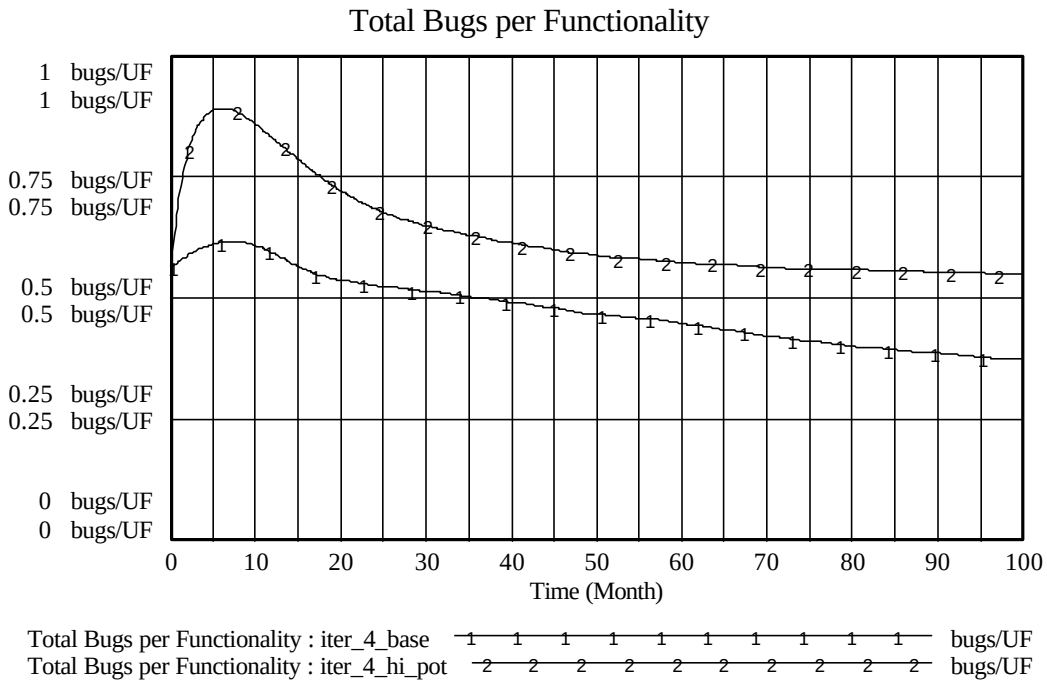


Figure 4.110. OSSD Model (Iteration IV) High Potential Run - Total Bugs per Functionality

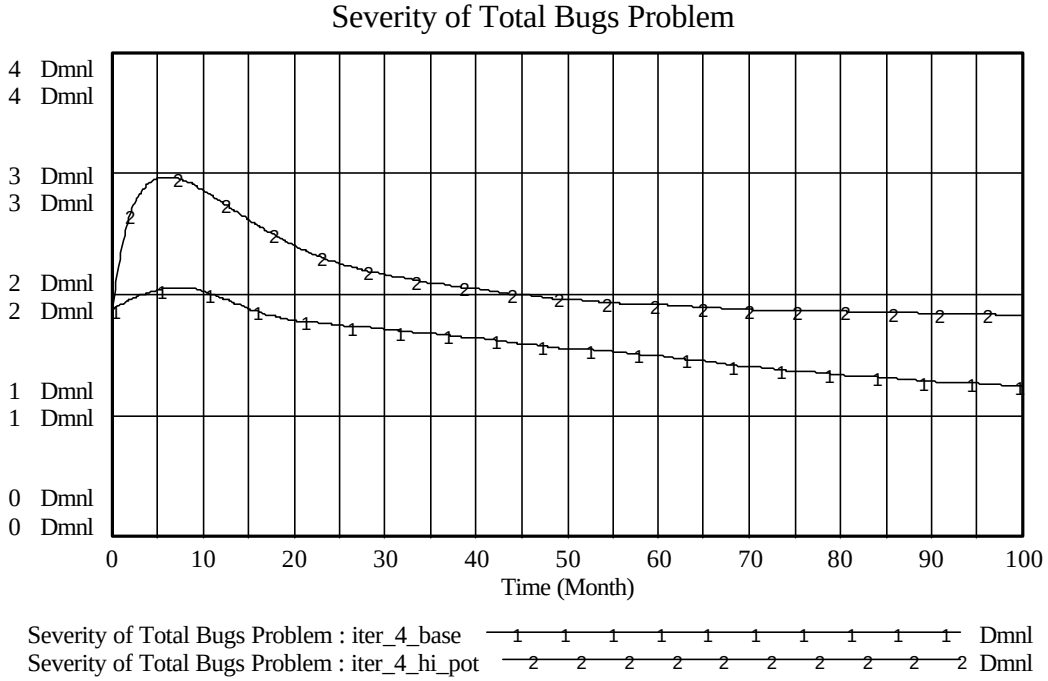


Figure 4.111. OSSD Model (Iteration IV) High Potential Run - Severity of Total Bugs Problem

Figure 4.112. OSSD Model (Iteration IV) High Potential Run - Perceived Product Quality

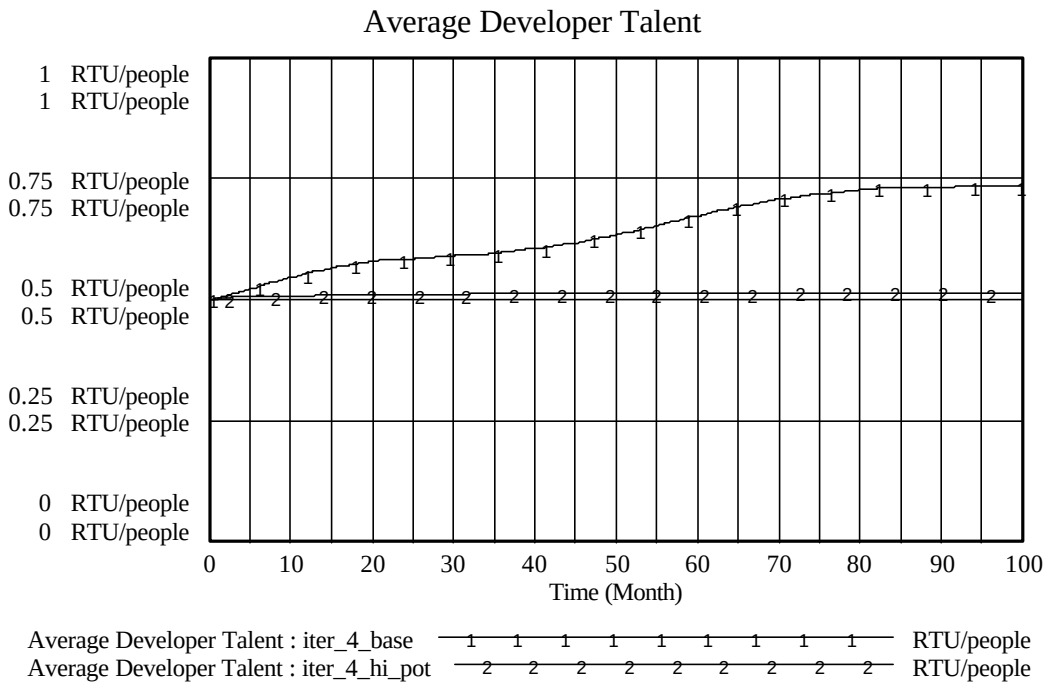


Figure 4.113. OSSD Model (Iteration IV) High Potential Run - Average Developer Talent

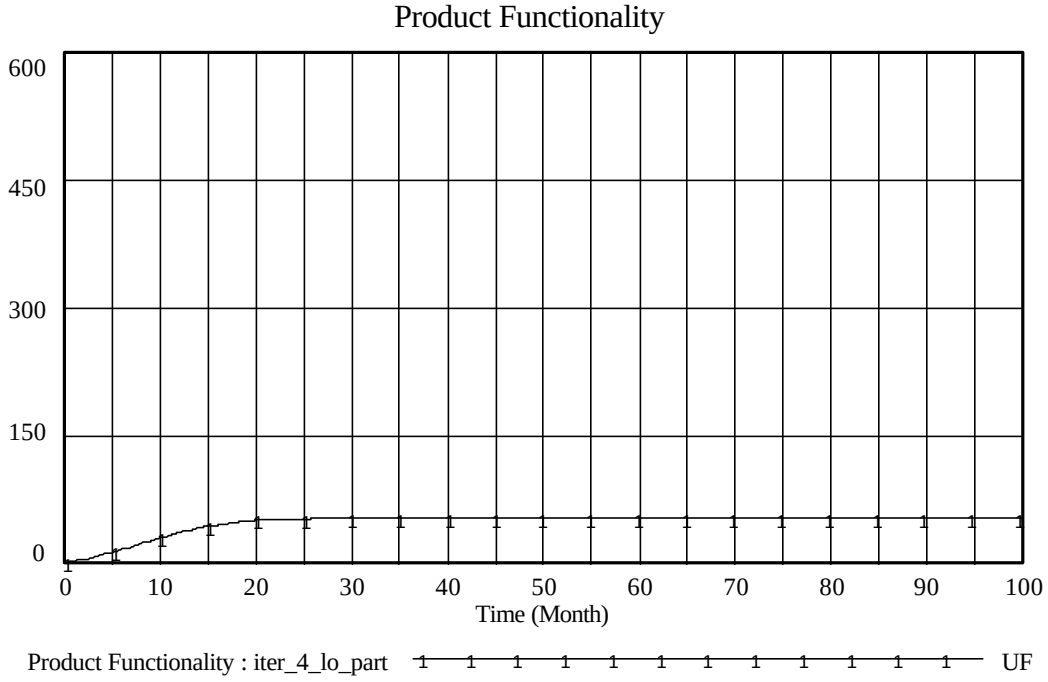


Figure 4.114. OSSD Model (Iteration IV) Low Participation Run - Product Functionality

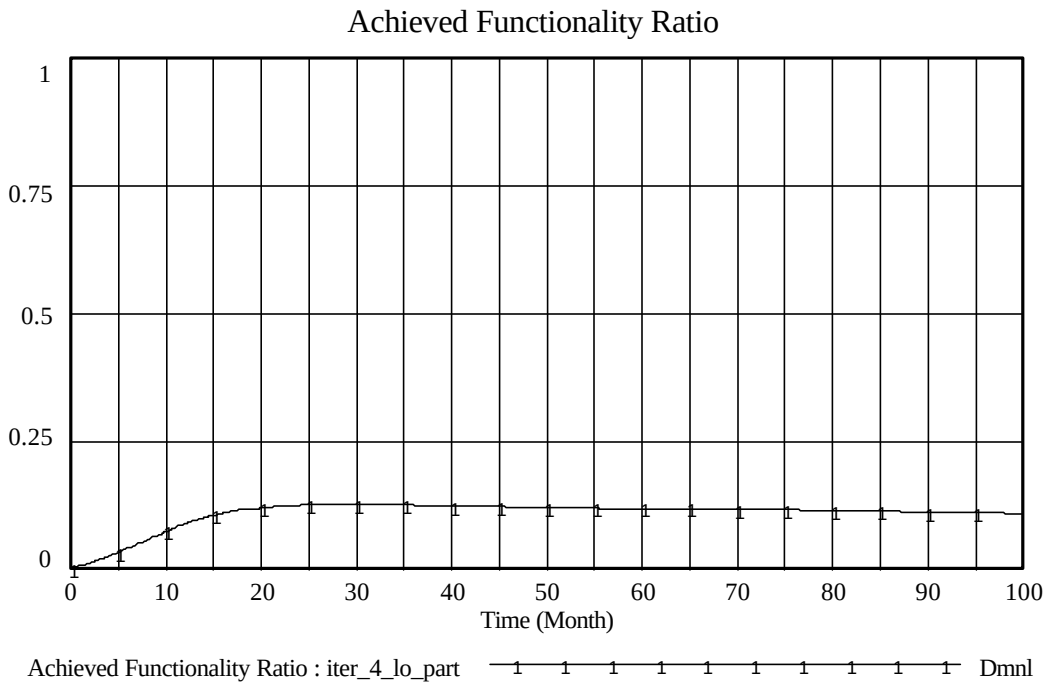


Figure 4.115. OSSD Model (Iteration IV) Low Participation Run - Achieved Functionality Ratio

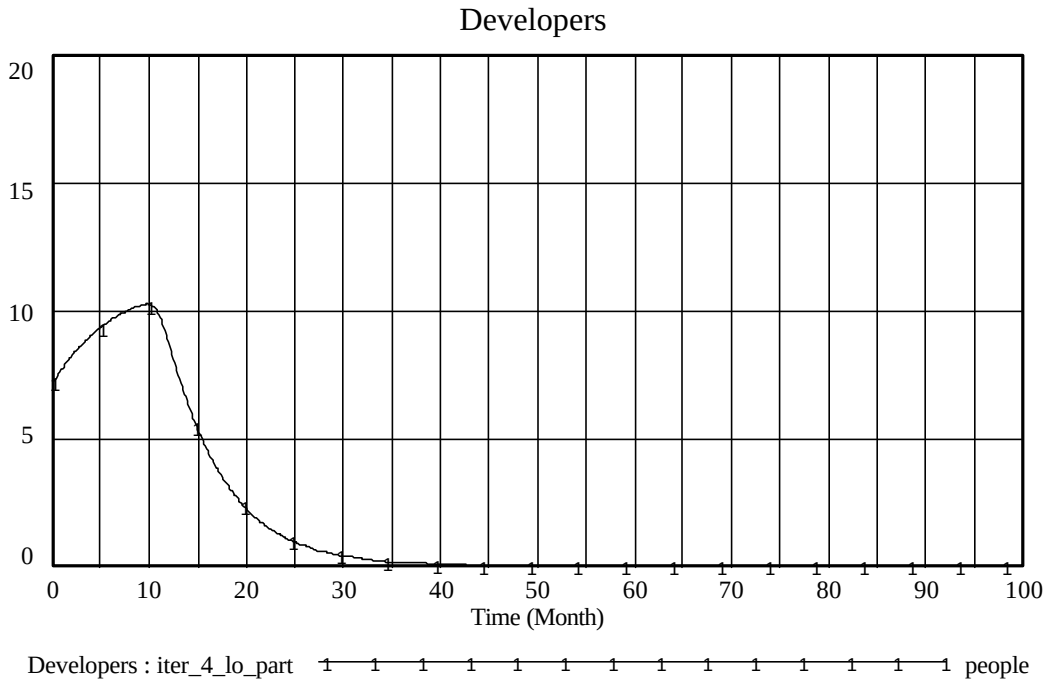


Figure 4.116. OSSD Model (Iteration IV) Low Participation Run - Developers

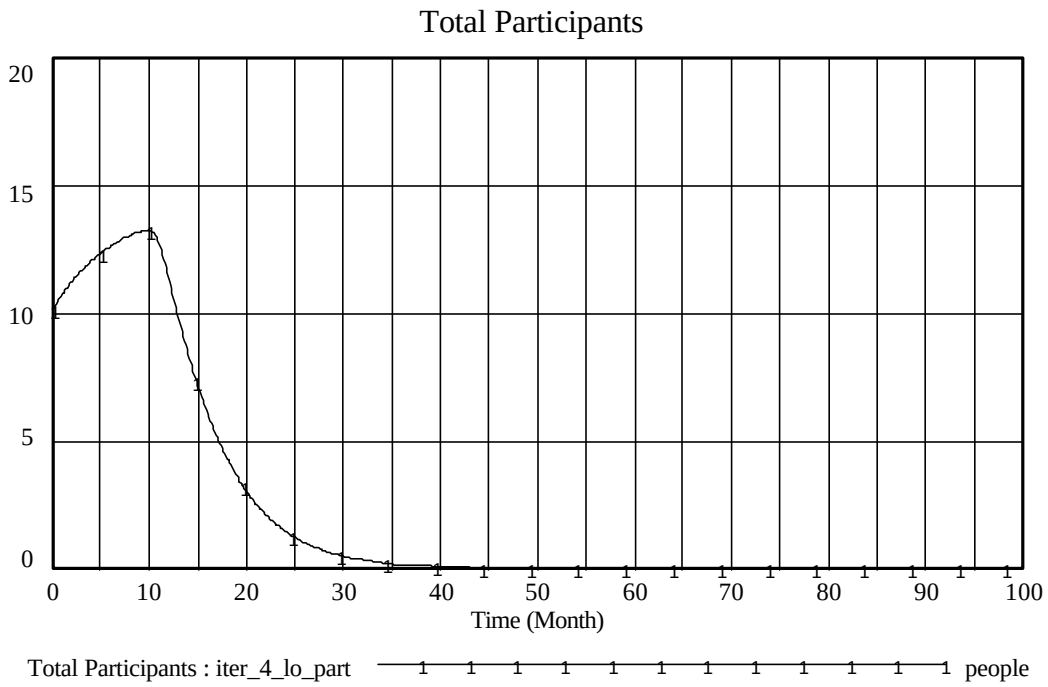


Figure 4.117. OSSD Model (Iteration IV) Low Participation Run - Total Participants

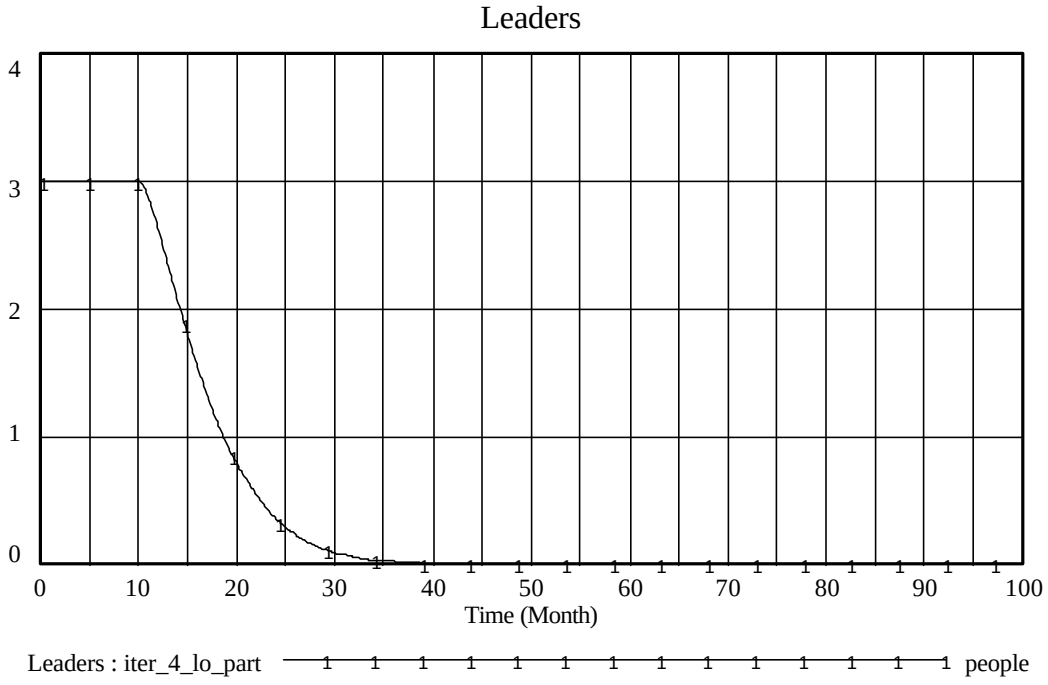


Figure 4.118. OSSD Model (Iteration IV) Low Participation Run - Leaders

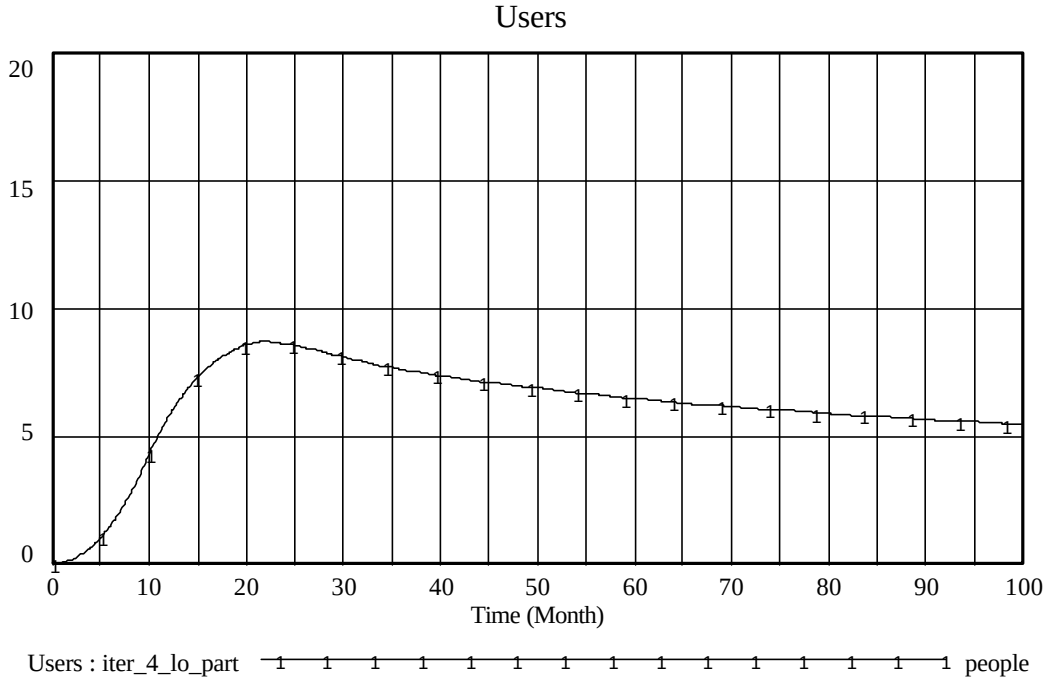


Figure 4.119. OSSD Model (Iteration IV) Low Participation Run - Users

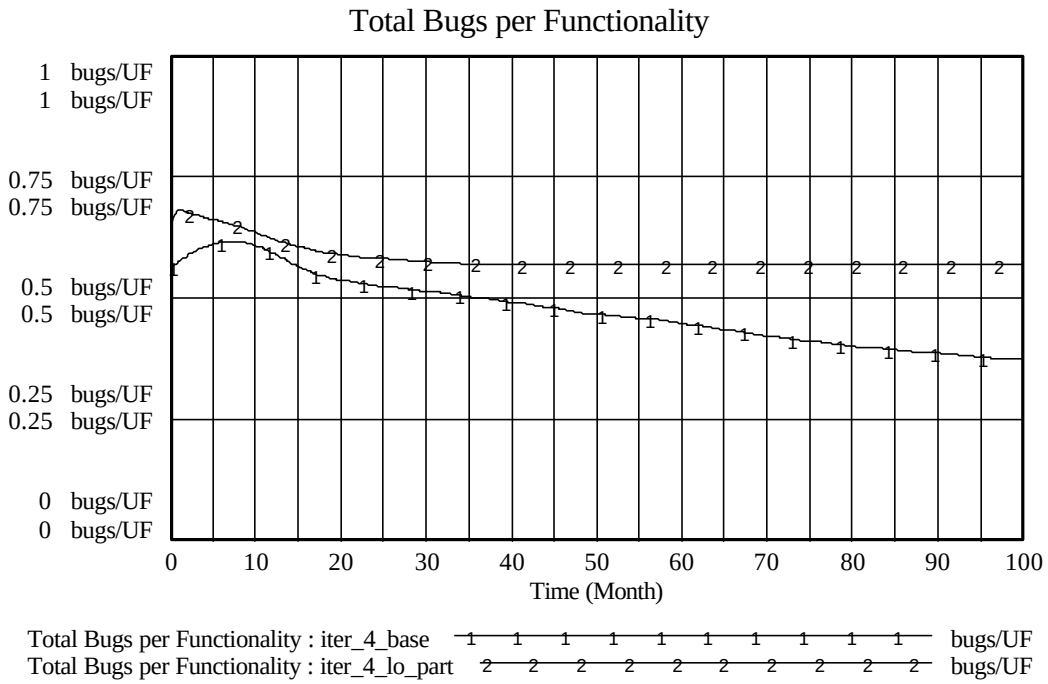


Figure 4.120. OSSD Model (Iteration IV) Low Participation Run - Total Bugs per Functionality

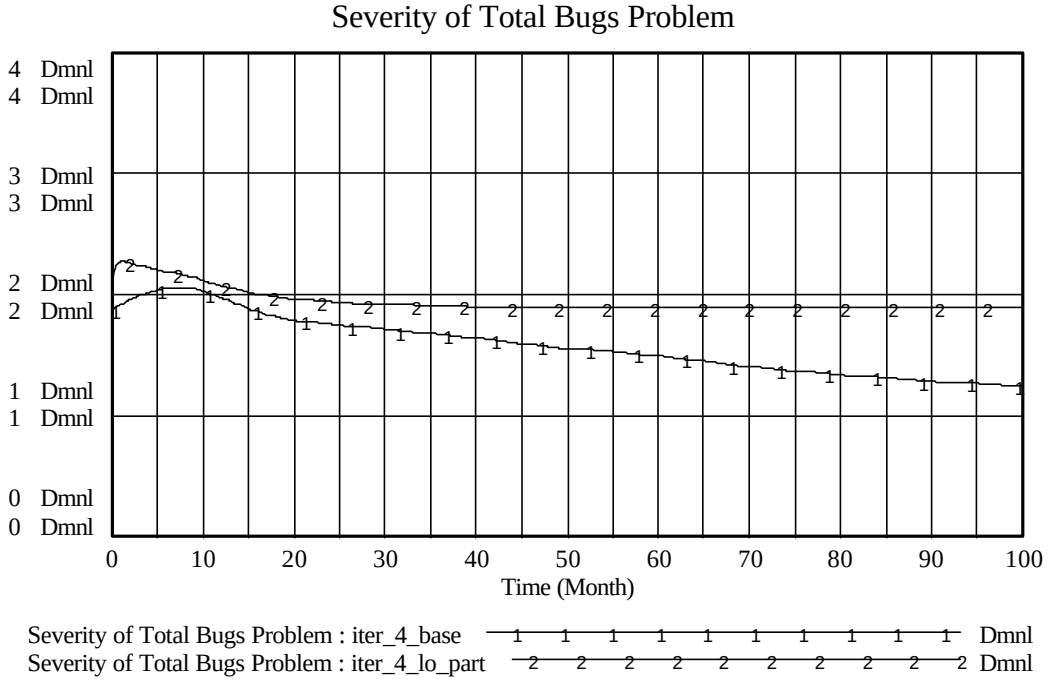


Figure 4.121. OSSD Model (Iteration IV) Low Participation Run - Severity of Total Bugs Problem

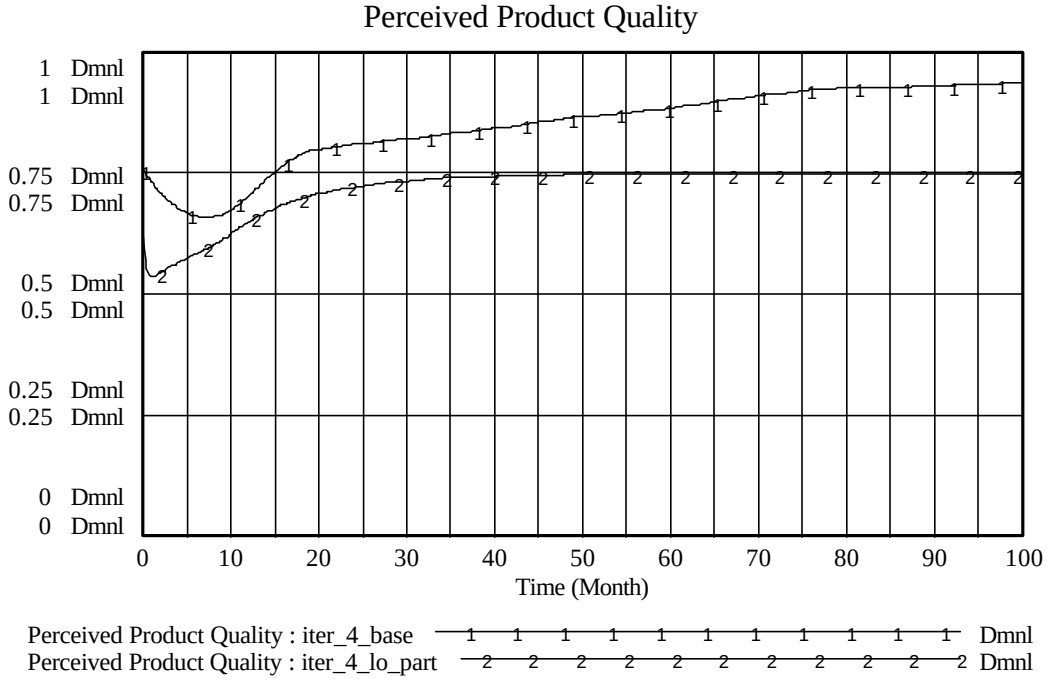


Figure 4.122. OSSD Model (Iteration IV) Low Participation Run - Perceived Product Quality

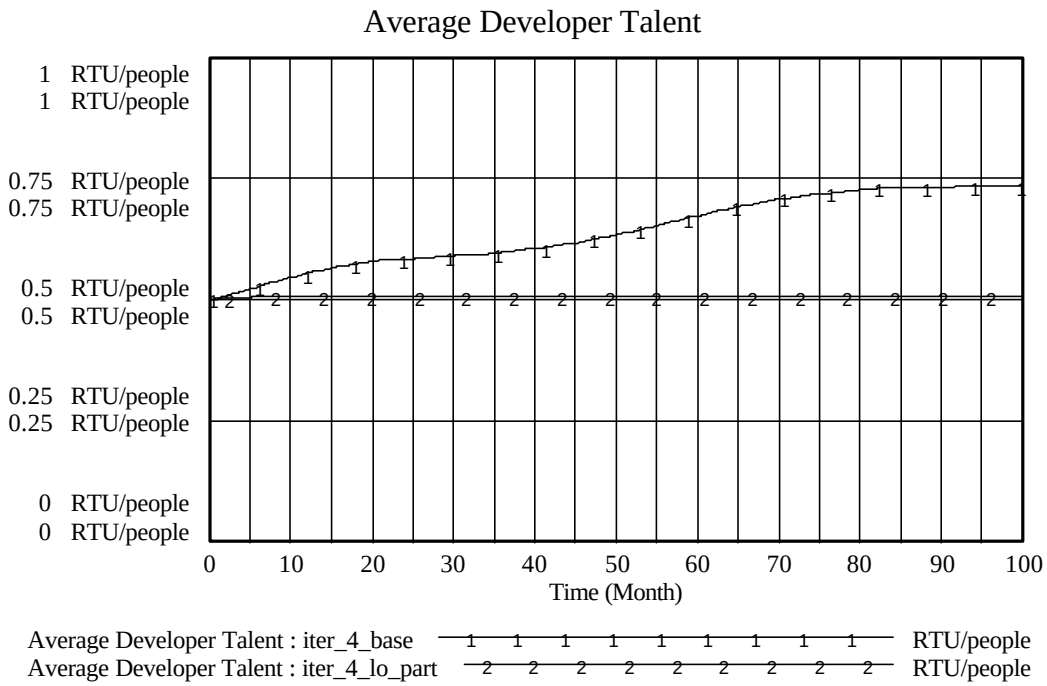


Figure 4.123. OSSD Model (Iteration IV) Low Participation Run - Average Developer Talent

4.6. Iteration V: Adding Barriers to Entry and Contribution

With the Iteration V version, barriers to entry to the community and barriers to making contributions are added to the model. Barriers to entry are realized through a process of selecting new developers that will join the community. (See Figure 4.124.) In this version of the model the incoming developers are not added directly to the developer pool. Instead they are selected from a pool of candidates. The selection is carried out with a selecting rate. A refusal ratio determines the percentage of the candidates that are denied entry to the community. The rest of the candidates are selected as new developers. Refusal ratio also determines the average talent of the incoming developers. A higher refusal ratio would mean a higher level of scrutiny while selecting new developers.

Consequently, as the refusal ratio increases, so does the average incoming developer talent.

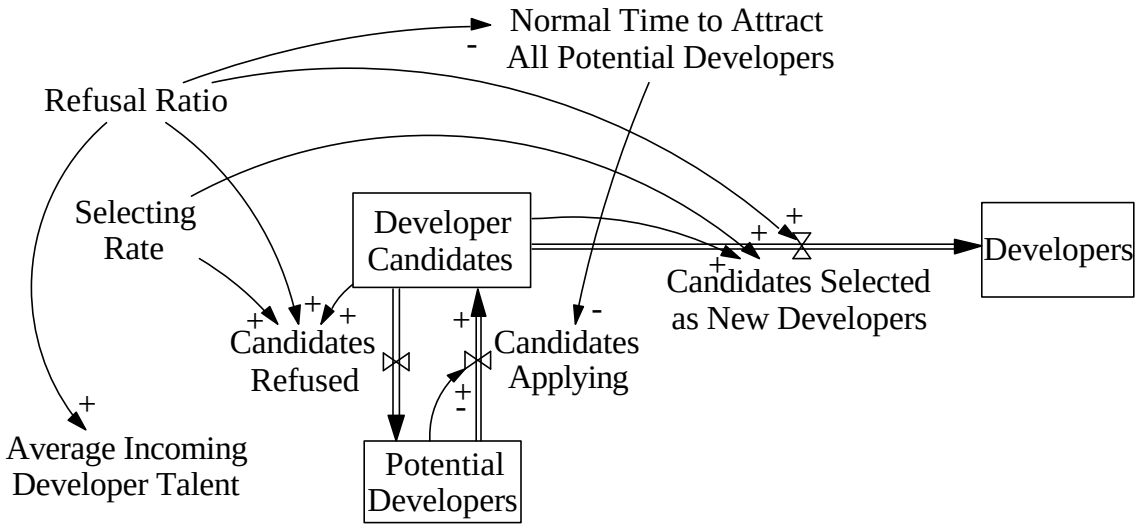


Figure 4.124. OSSD Model (Iteration V) Changes in the Developers Sector due to Adding Barriers to Entry to the Model

A filtering process provides the mechanism for barriers to making contributions. Production effort is divided between developers and leaders, as discussed earlier in this chapter. The leaders review the code produced by the developers. Reviewed code is accepted or rejected based on its quality. The quality of the code in this context means the number of bugs per unit functionality, as discussed earlier in this chapter. Code produced by developers is added to a backlog to be filtered, and the leaders filter the backlog with a filtering rate. Code is accepted or rejected based on a rejection ratio. The rejection ratio has a negative effect on average developer participation. As a greater portion of production is rejected, the developers would be less motivated to produce further code, thus their participation level would decrease. (See Figure 4.125.)

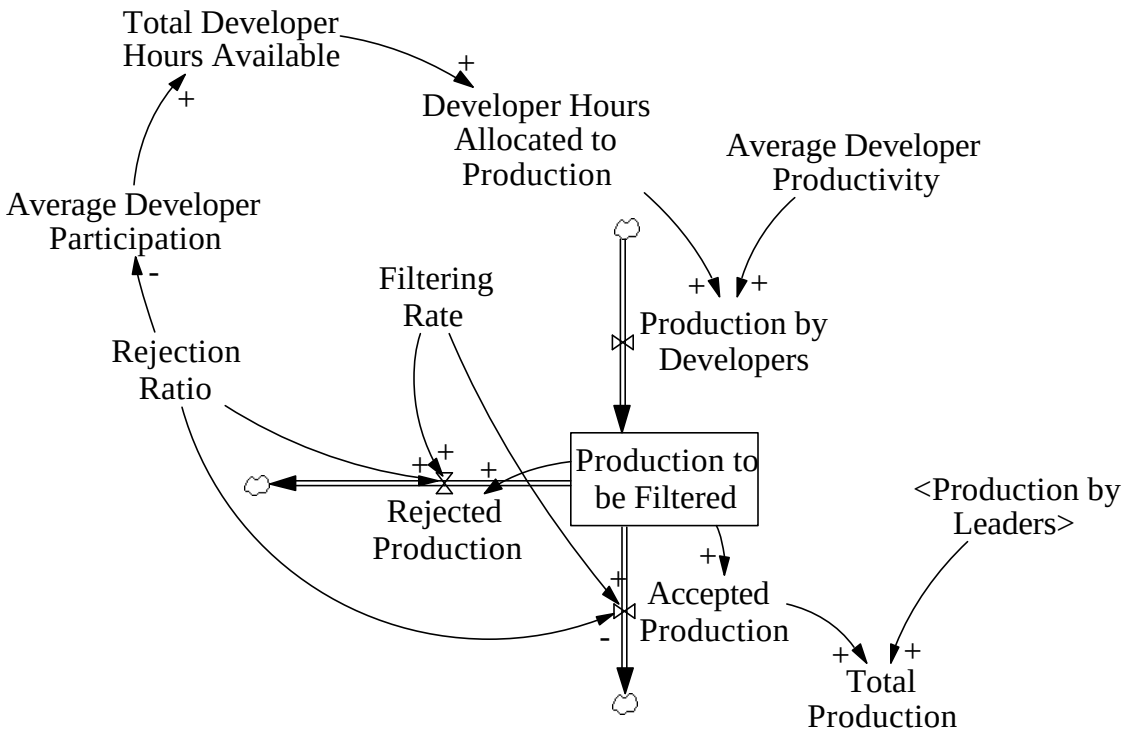


Figure 4.125. OSSD Model (Iteration V) Changes in the Developers Sector due to Adding Barriers to Contribution to the Model

A new Filtering Sector is also added to the model with the Iteration V version. As developers produce code they also generate bugs, and these bugs are added to a "backlog of bugs" just as production is added to the backlog of production. (See Figure 4.126.) When the backlog of production is reviewed and some of the code is accepted and added to the overall product, a certain number of the bugs are also introduced to the product. This is represented by the outflow Bugs in Accepted Code. Another group of bugs, represented by the outflow Bugs in Rejected Code, also flow out of the "backlog of bugs" with the rejected code. (See Figure 4.127.)

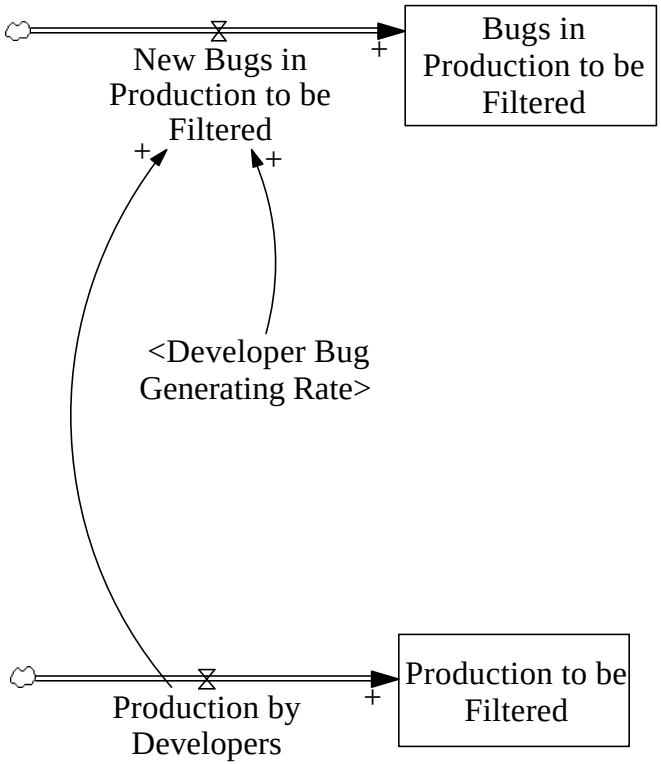


Figure 4.126. OSSD Model (Iteration V) Filtering Sector

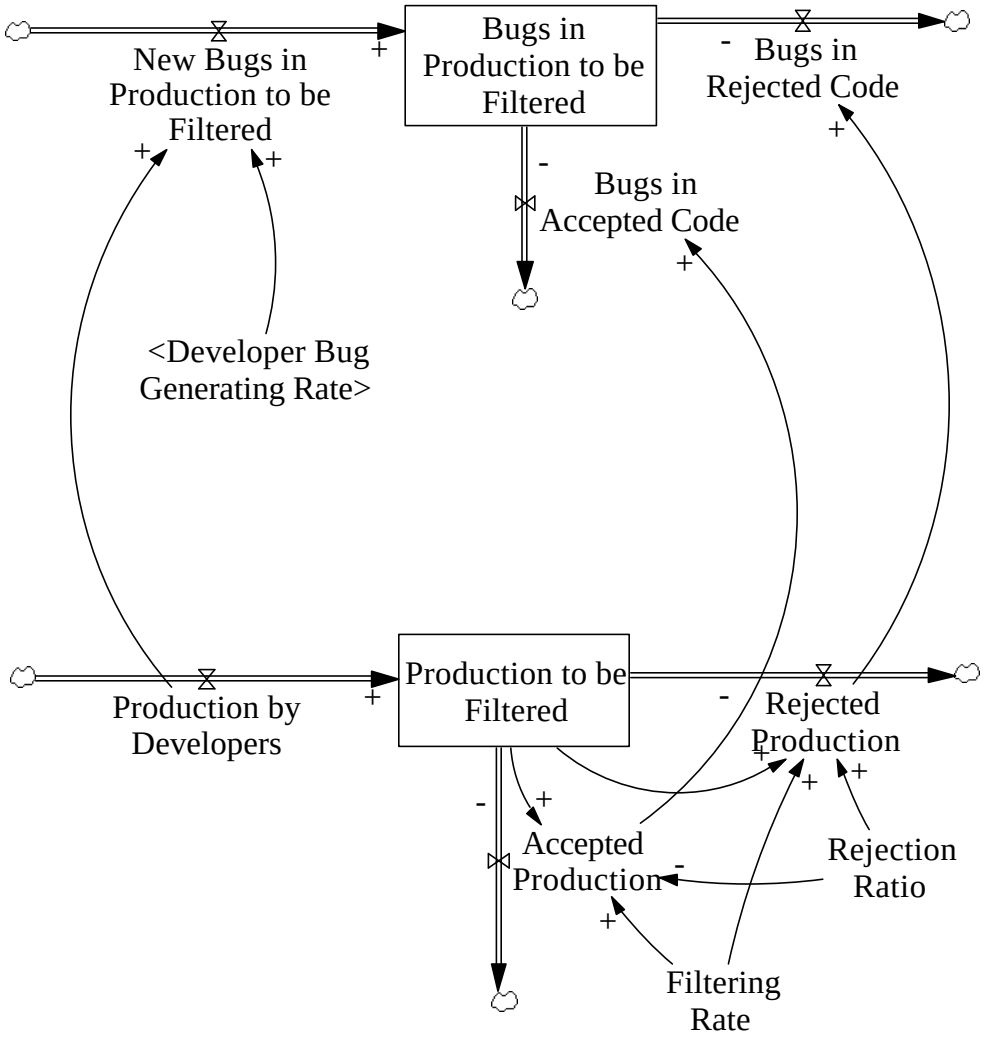


Figure 4.127. OSSD Model (Iteration V) Filtering Sector

The filtering process aims to decrease the number of bugs in new code that is added to the product. Thus, it is expected that the bug density of the accepted portion of the code will be less than that of the production backlog. The assumption here is that the worst case of filtering would yield the same number of bugs per functionality as the original production. Any case better than the worst case would bring a quality improvement, which will yield a lower bug density for the accepted code and a higher bug density for the rejected code. The level of quality improvement is determined by the quality of filtering. (See Figure 4.128.)

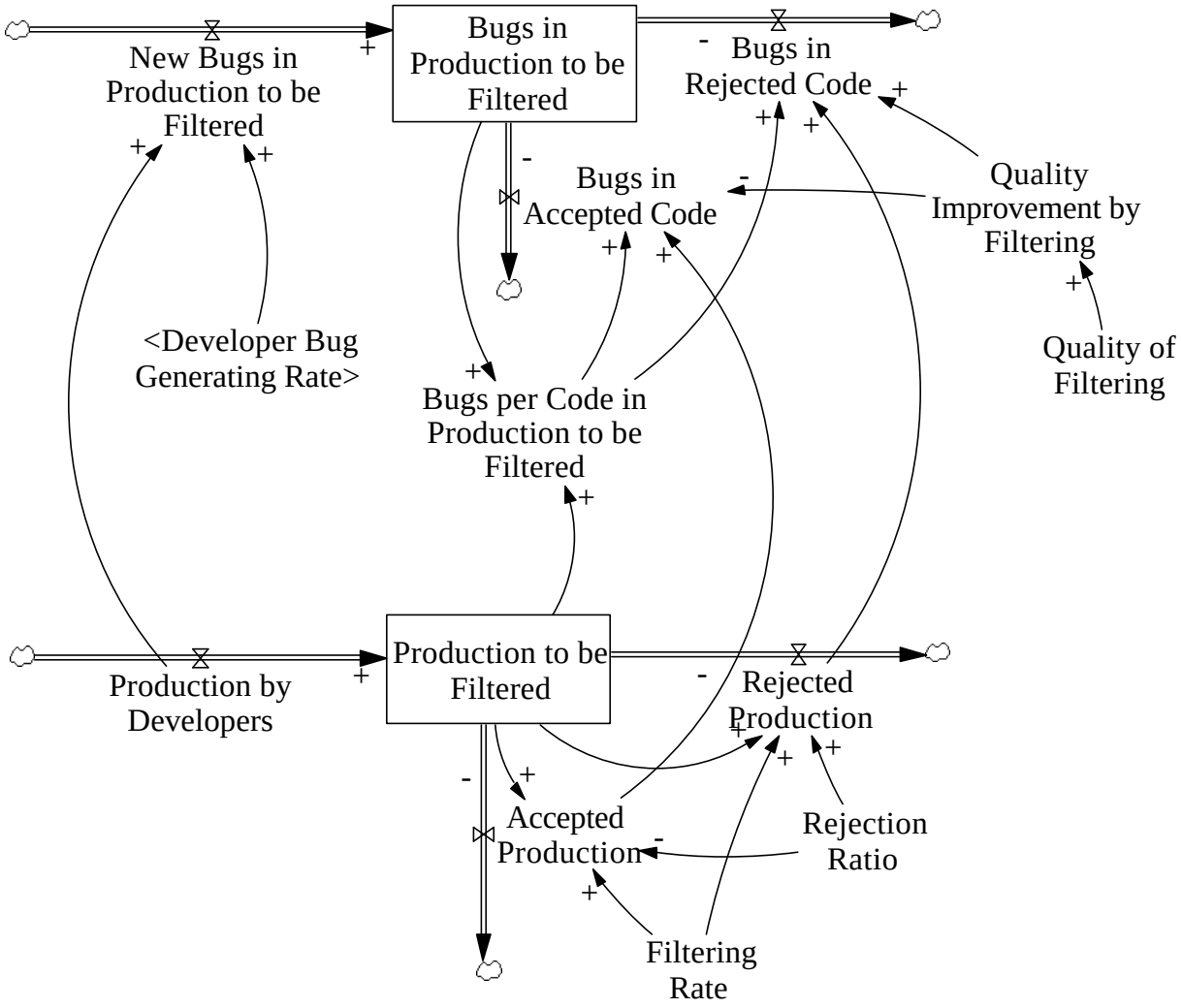


Figure 4.128. OSSD Model (Iteration V) Filtering Sector

The quality of filtering depends on the relative rate at which the leaders filter the production backlog. The model assumes a fixed filtering rate, which is set to .5 for the base case. This means that the leaders would filter .5 of the backlog at a given month, regardless of the size of the backlog. However, there would be an optimal filtering rate for a given amount of code filtered by a given number of leaders, and as the actual filtering rate goes above that optimal rate the quality of filtering decreases. The optimal filtering rate depends on the optimal filtering horizon, which is the amount of time the leaders can filter the existing backlog without compromising the quality of filtering. (See Figure 4.129.)

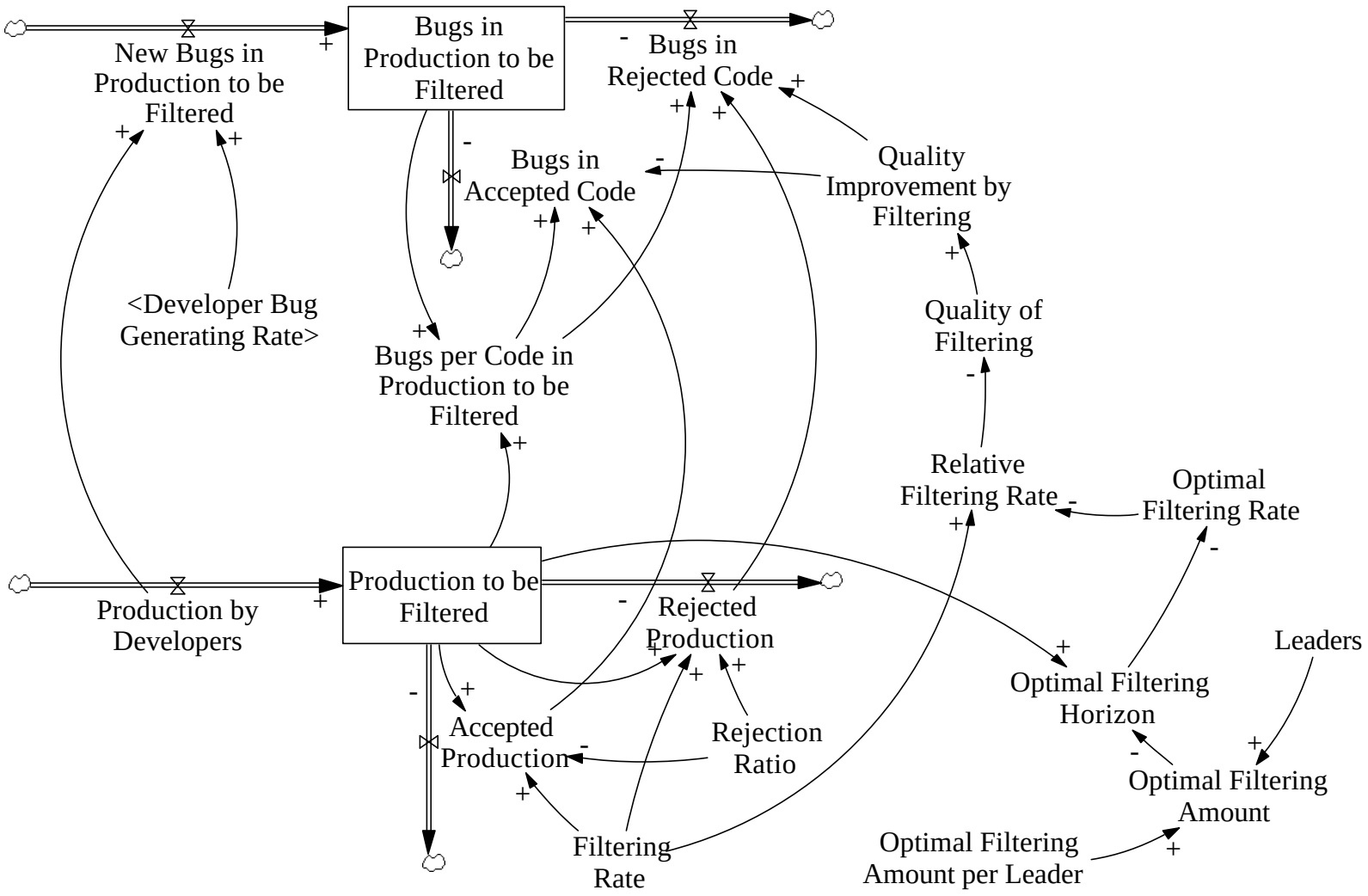


Figure 4.129. OSSD Model (Iteration V) Filtering Sector

Once again, the general behavior of the model is mostly preserved from the Iteration IV version to the Iteration V version, while adding new structure and consequently more explanatory power to the model. Figures 4.130 through 4.134 demonstrate the behaviors of the main indicators under base case conditions.

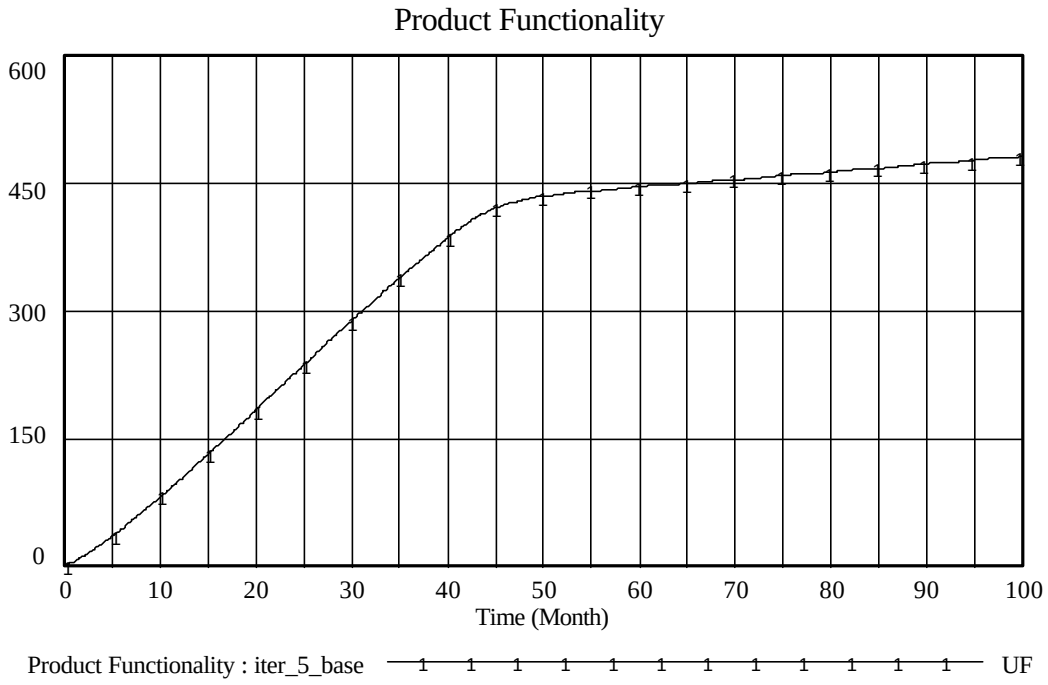


Figure 4.130. OSSD Model (Iteration V) Base Run - Product Functionality

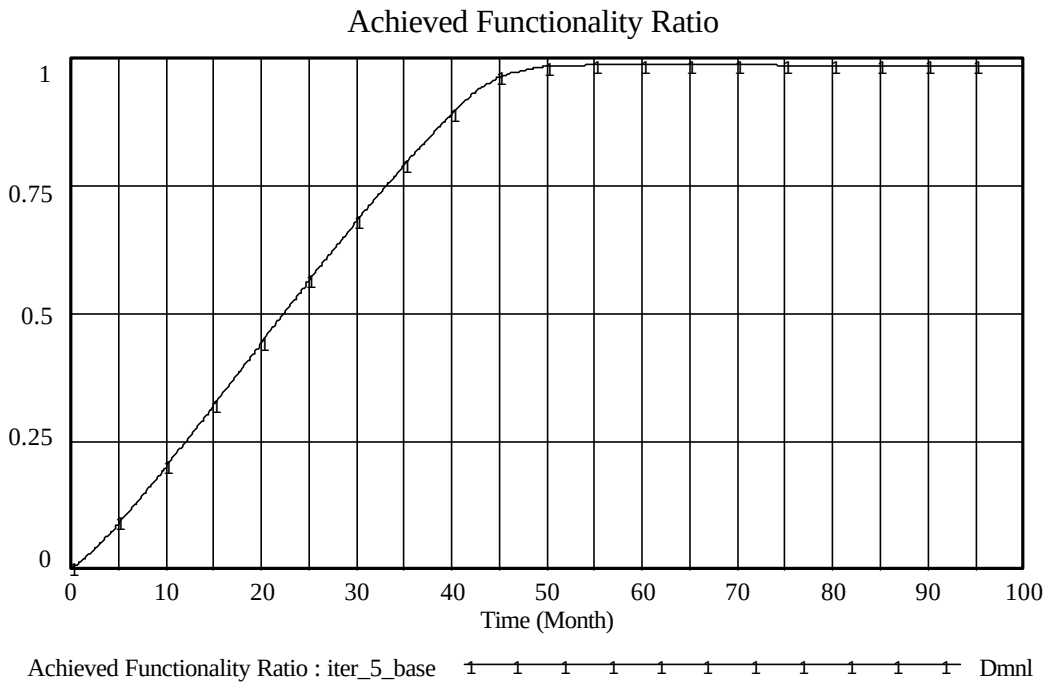


Figure 4.131. OSSD Model (Iteration V) Base Run - Achieved Functionality Ratio

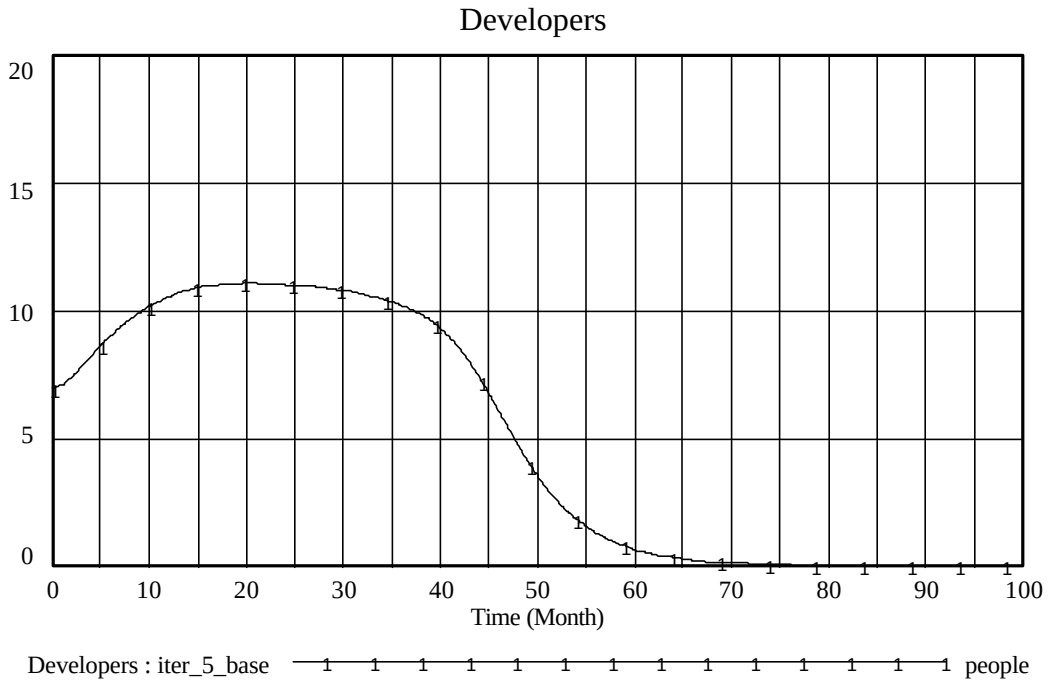


Figure 4.132. OSSD Model (Iteration V) Base Run - Developers

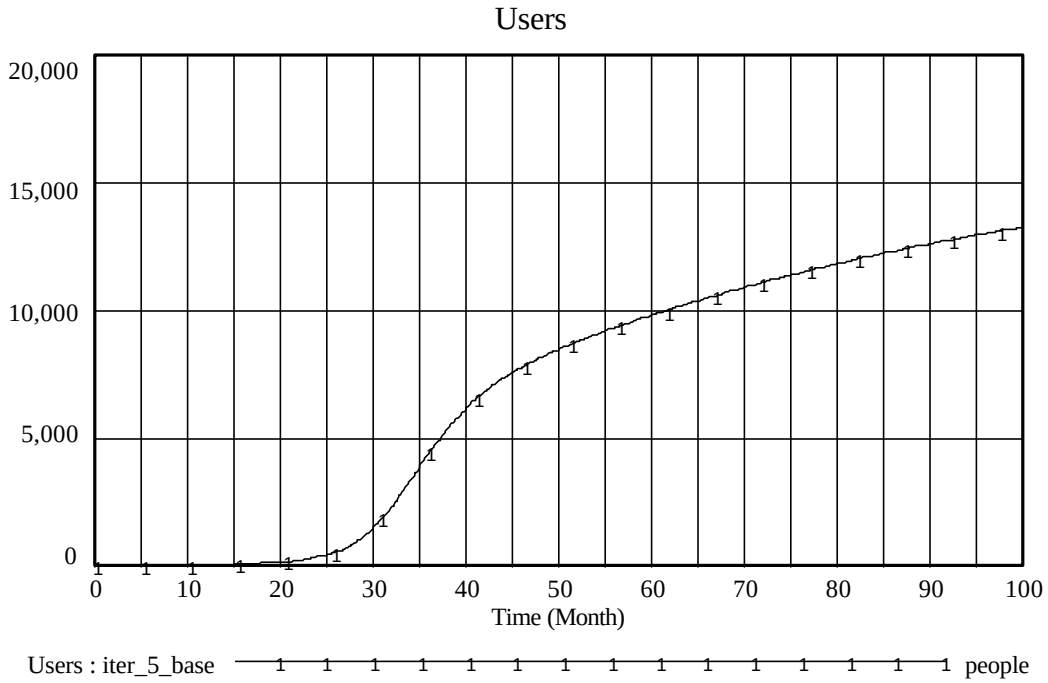


Figure 4.133. OSSD Model (Iteration V) Base Run - Users

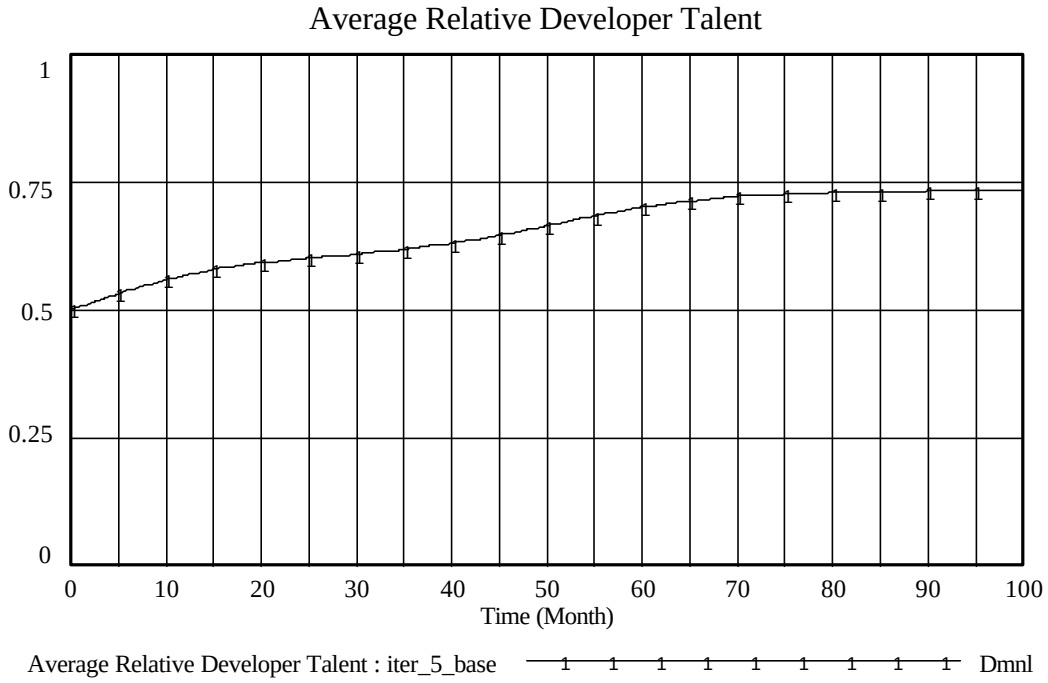


Figure 4.134. OSSD Model (Iteration V) Base Run - Average Relative Developer Talent

One important behavior difference between the Iteration V version and the previous versions is observed in the behavior of Total Bugs per Functionality and other variables that are affected by it, namely Severity of Total Bugs Problem and Perceived Product Quality. In the Iteration V version, Total Bugs per Functionality starts at a lower level than it does in the previous versions of the model. Also it does not reach as high a peak as in the previous versions. (See Figure 4.135.) Severity of Total Bugs Problem, too, starts lower, and reaches a lower peak than in the previous versions. (See Figure 4.136.) Consequently, Perceived Product Quality starts at a higher level, and does not reach as low a level as it does in the previous versions. (See Figure 4.137.)

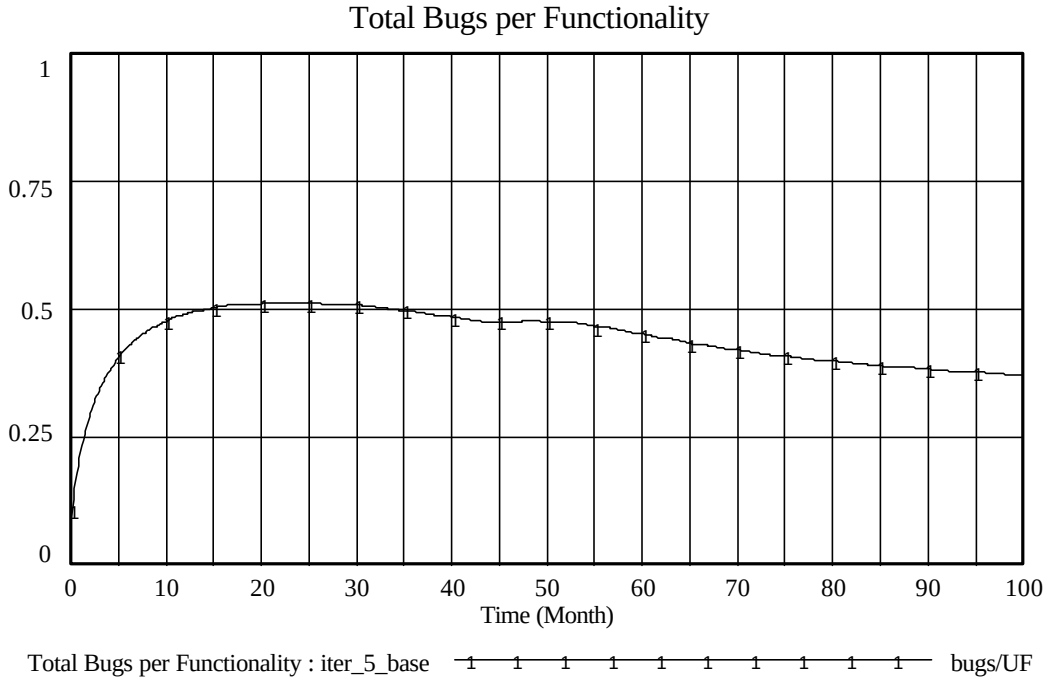


Figure 4.135. OSSD Model (Iteration V) Base Run - Total Bugs per Functionality

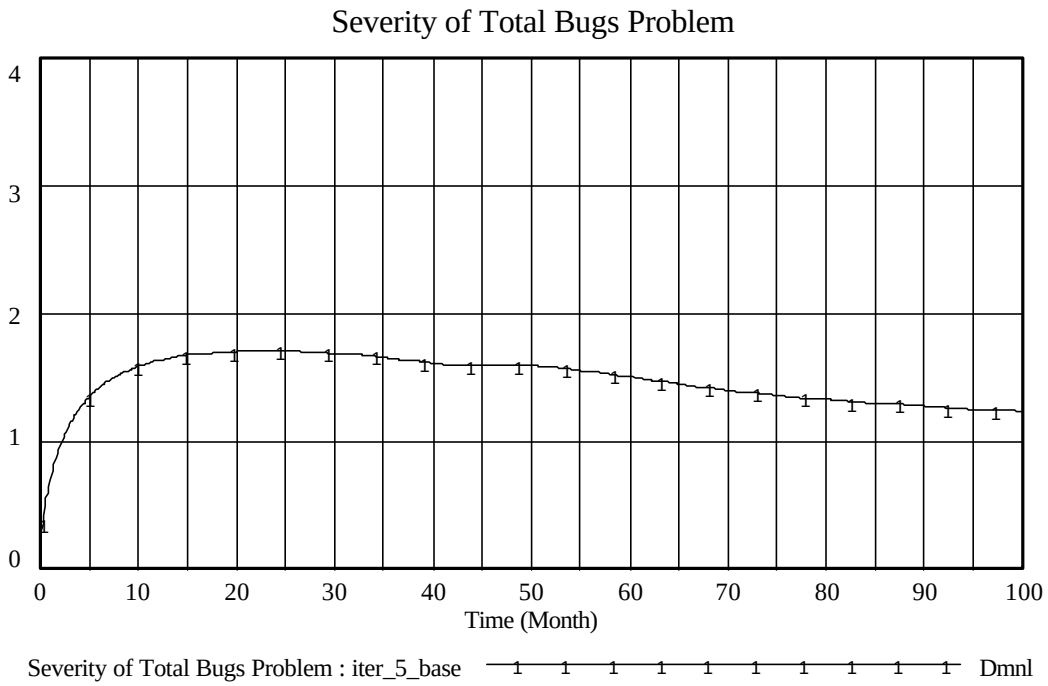


Figure 4.136. OSSD Model (Iteration V) Base Run - Severity of Total Bugs Problem

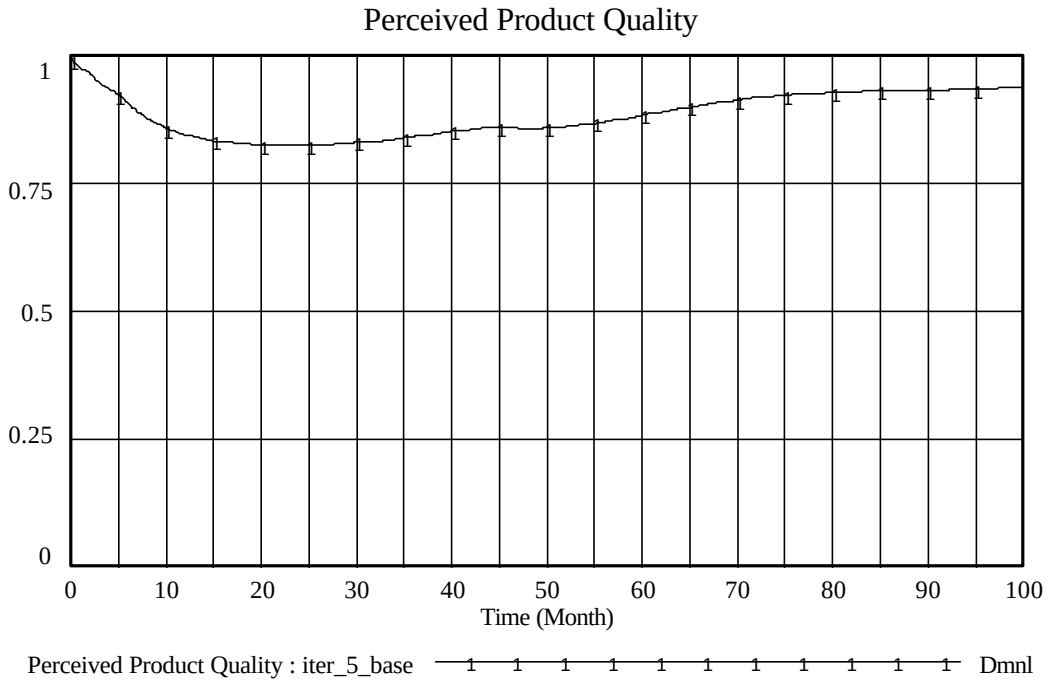


Figure 4.137. OSSD Model (Iteration V) Base Run - Perceived Product Quality

However, a closer comparison of the behaviors of these variables in the Iteration IV and Iteration V models show that they demonstrate almost the same behaviors after about month 25 in both versions. (See Figure 4.138 through Figure 4.140.)

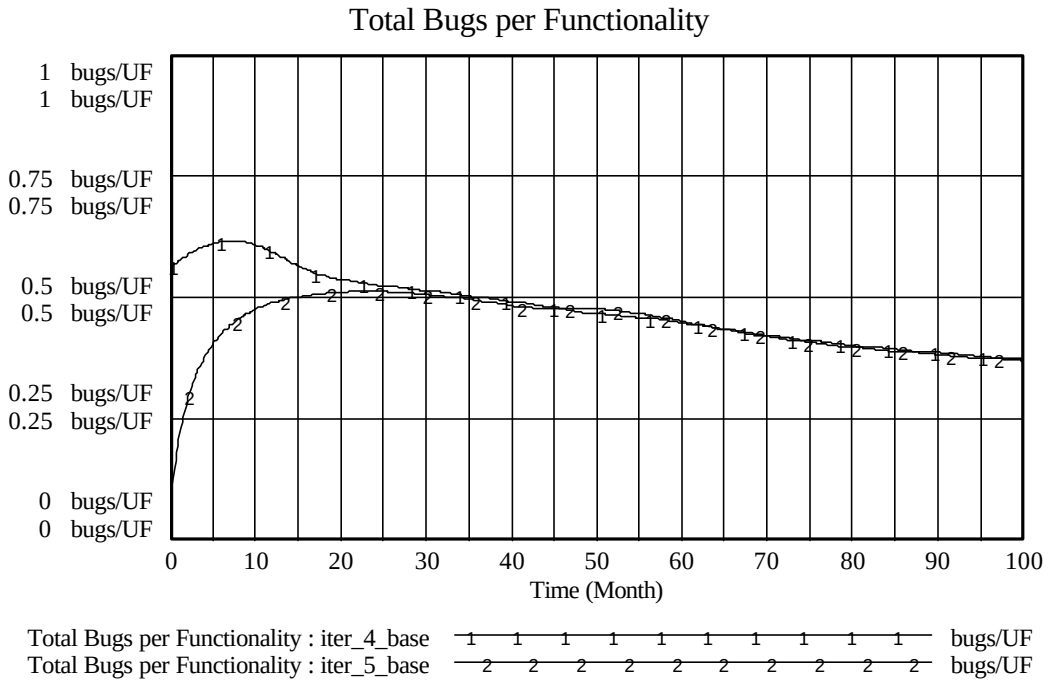


Figure 4.138. OSSD Model (Iteration V compared with Iteration IV) Base Run - Total Bugs per Functionality

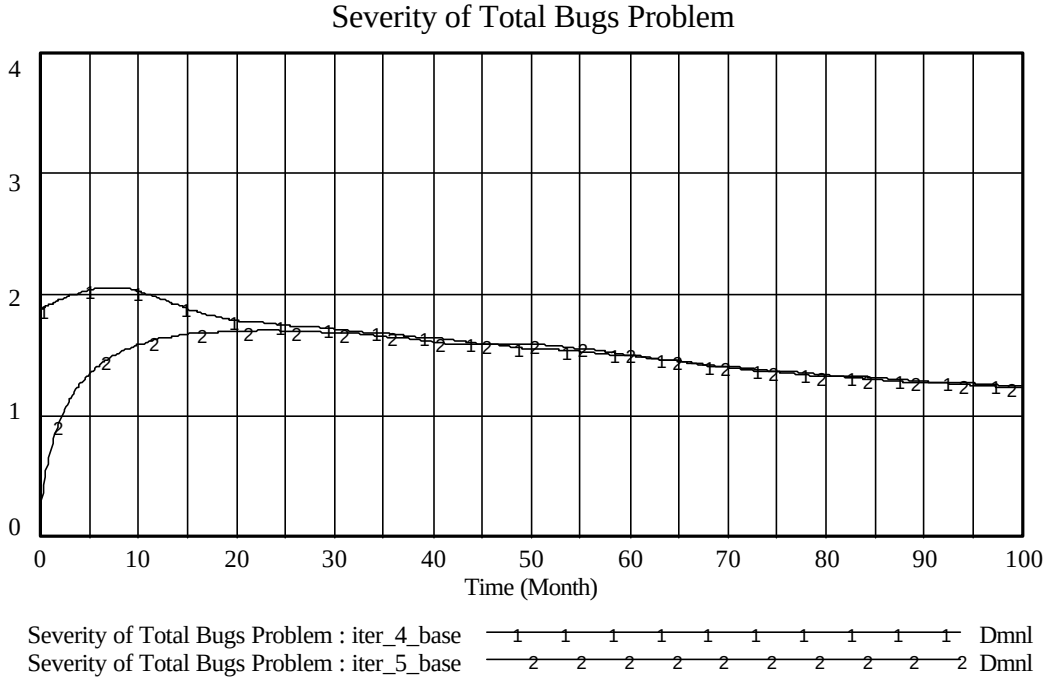


Figure 4.139. OSSD Model (Iteration V compared with Iteration IV) Base Run - Severity of Total Bugs Problem

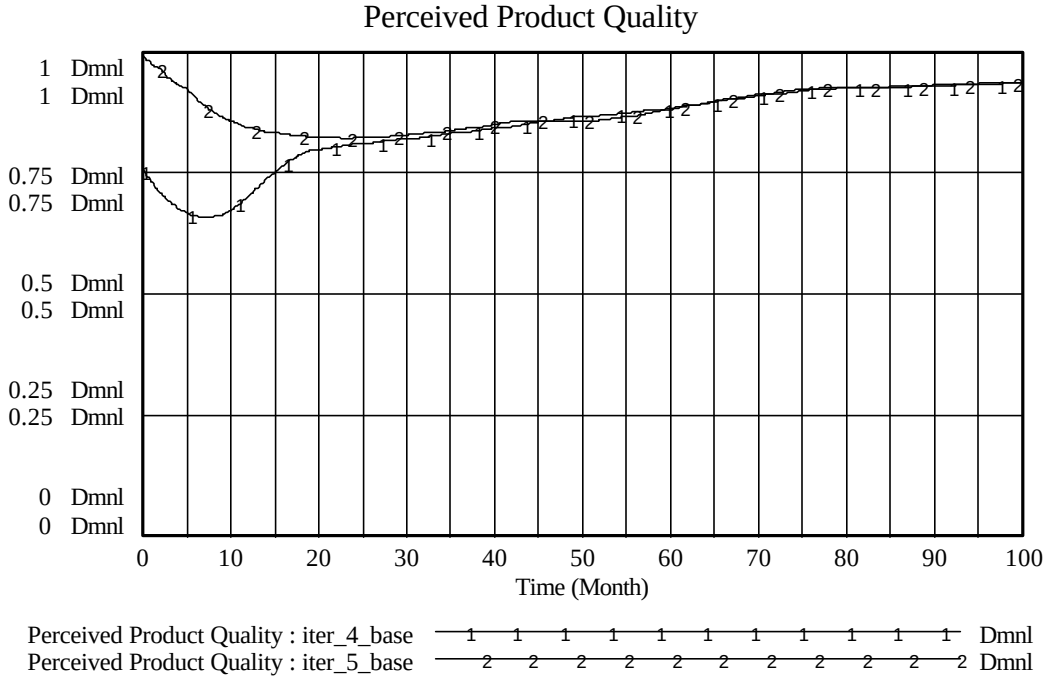


Figure 4.140. OSSD Model (Iteration V compared with Iteration IV) Base Run - Perceived Product Quality

The initial differences between the behaviors are attributable to the addition of the filtering process to the model. Production by developers involves a higher number of bugs per functionality compared to production by leaders. The filtering process delays the inclusion of production by developers in the overall product pool. So, in the Iteration V version most of the early production comes from the leaders, and thus has a lower bugs per functionality ratio. As more production by developers is added to the product the bugs per functionality ratio increases. Though the leaders eliminate a portion of the bugs through the filtering process, there are still bugs from production by developers that go into the product. The number of bugs in the later stages of the project depends on the rate of debugging rather than filtering, because debugging is driven by the assessment of the severity of the total bugs problem. As a consequence, the number of bugs does not decrease in the Iteration V version, more than it does in the Iteration IV version, because the pressure for debugging is the same in both versions. However, since there are fewer bugs to fix throughout the entire project in the Iteration V version, a certain amount of time is saved. That time is used for production and coaching, and consequently the achieved functionality ratio and average developer talent increase faster in the Iteration V version. (See Figure 4.141. and Figure 4.142.)

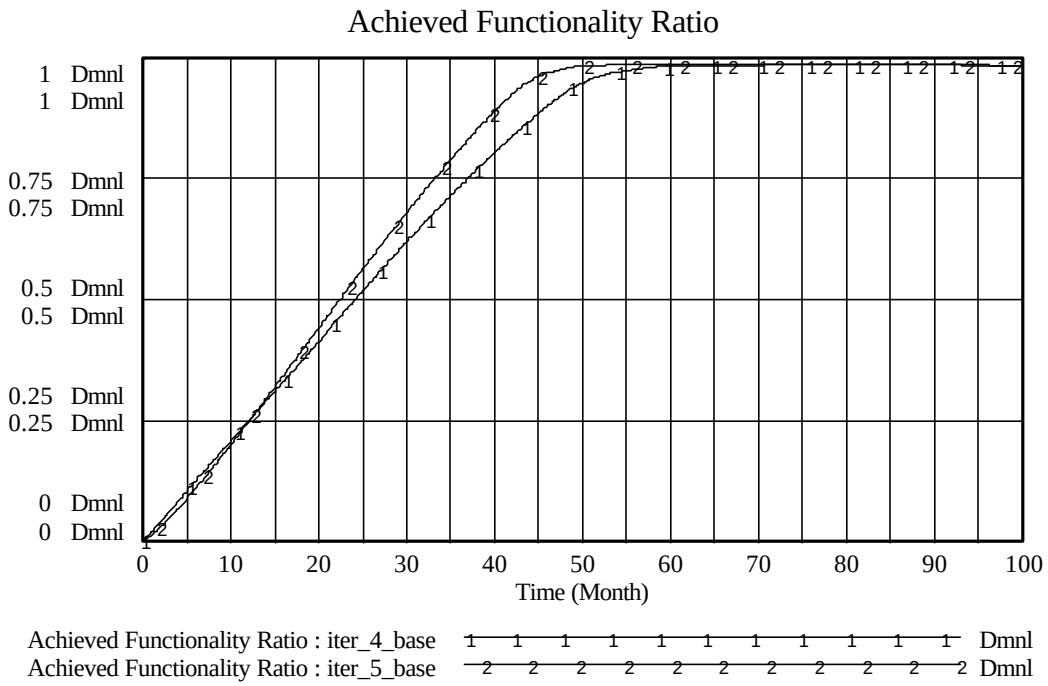


Figure 4.141. OSSD Model (Iteration V compared with Iteration IV) Base Run - Achieved Functionality Ratio

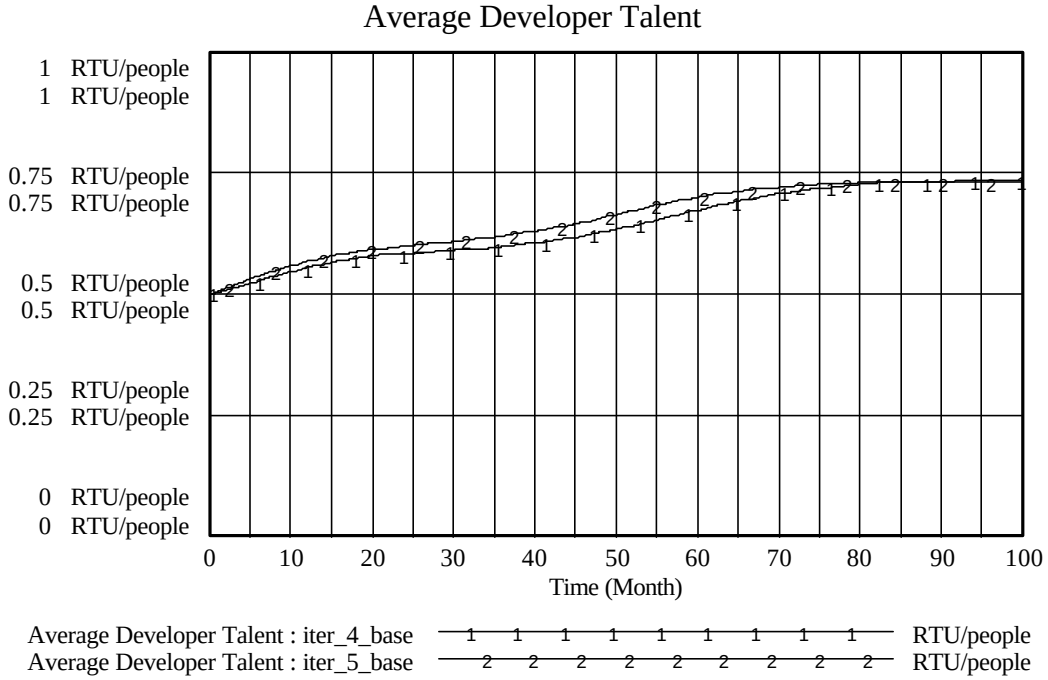


Figure 4.142. OSSD Model (Iteration V compared with Iteration IV) Base Run - Average Developer Talent

The behaviors of the Iteration V model under the two alternative conditions do not exhibit substantial differences than those found in previous versions with the exception of the behaviors of Total Bugs per Functionality and the variables affected by it. These differences are attributable to the inclusion of the filtering process as discussed above. Figures 4.143 through 4.150 portray the behavior of the Iteration V version under high initial limit on product functionality condition, while Figures 4.151 through 4.160 show the behavior of the model under low participation condition.

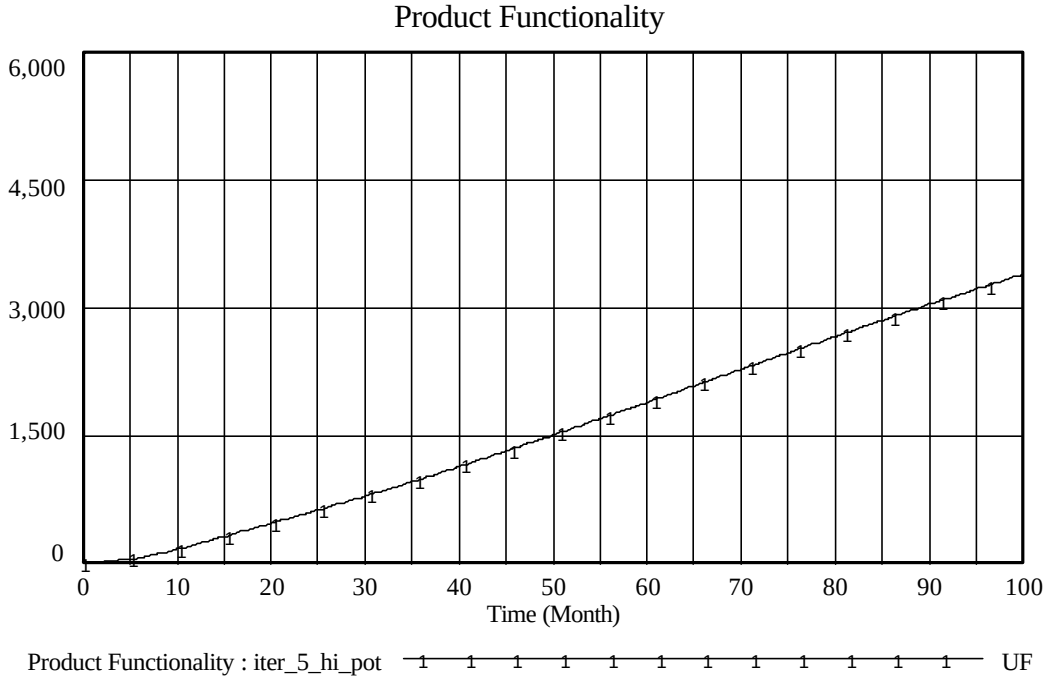


Figure 4.143. OSSD Model (Iteration V) High Potential Run - Product Functionality

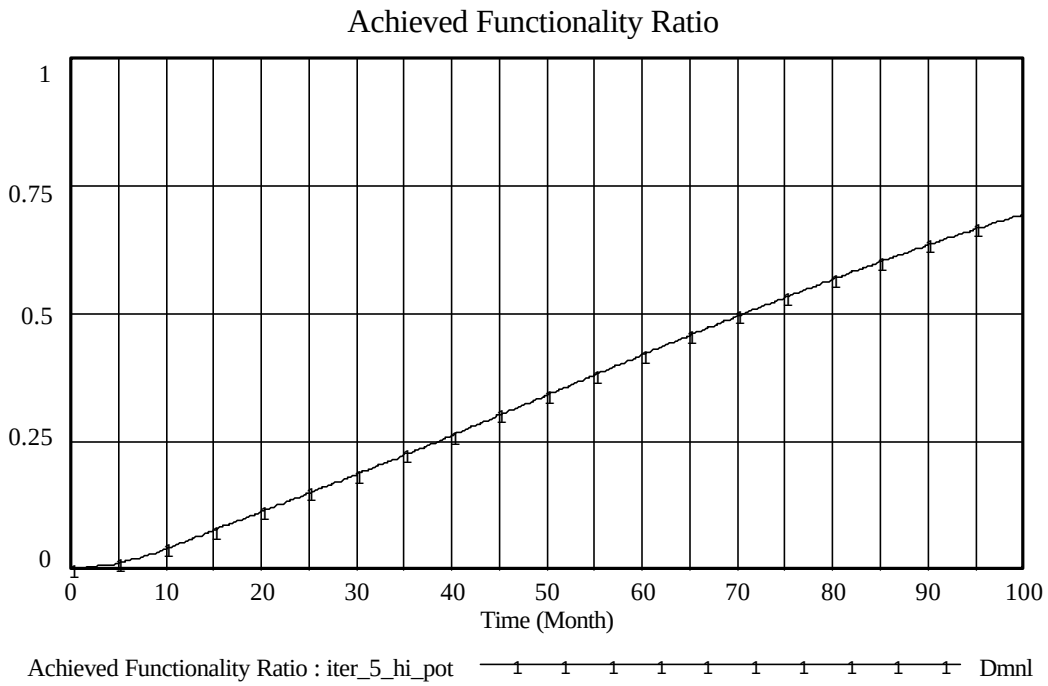


Figure 4.144. OSSD Model (Iteration V) High Potential Run - Achieved Functionality Ratio

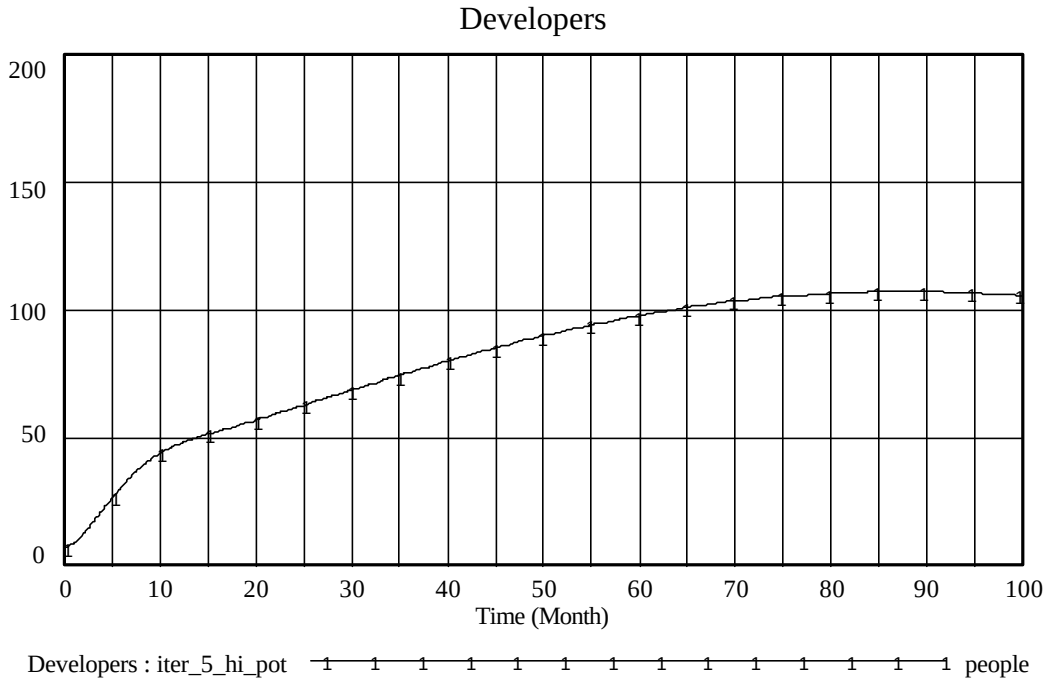


Figure 4.145. OSSD Model (Iteration V) High Potential Run - Developers

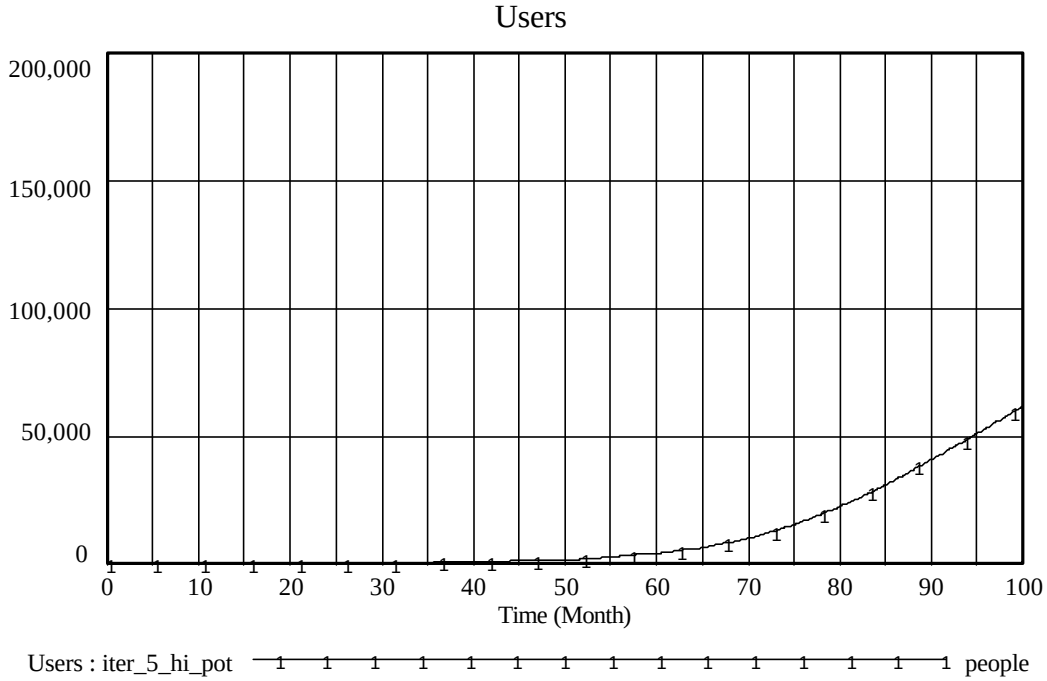


Figure 4.146. OSSD Model (Iteration V) High Potential Run - Users

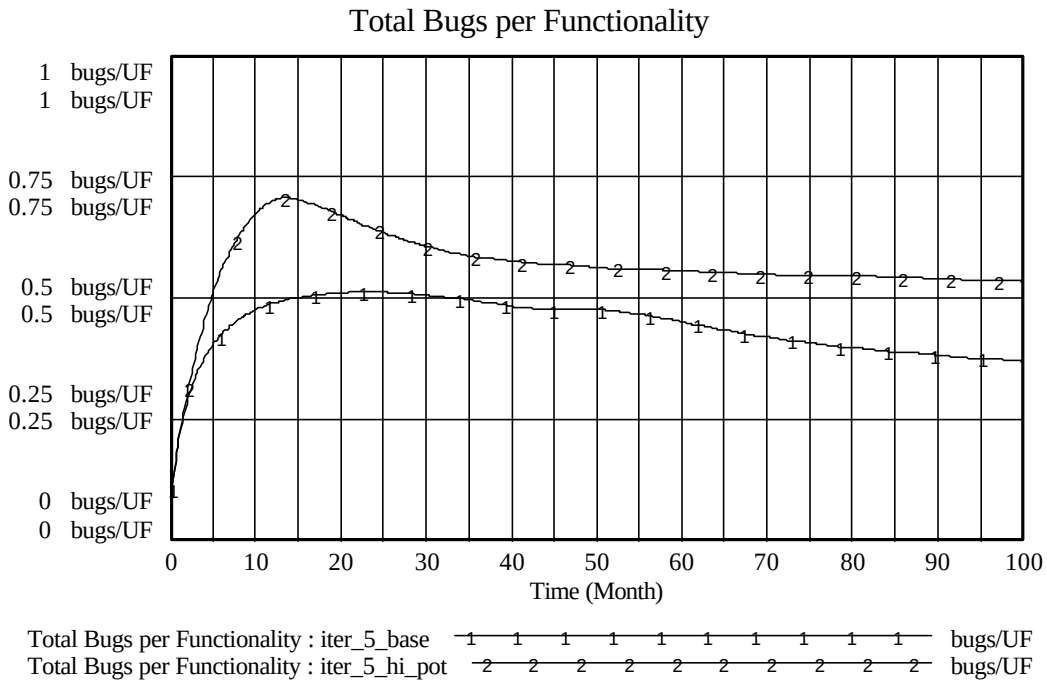


Figure 4.147. OSSD Model (Iteration V) High Potential Run - Total Bugs per Functionality

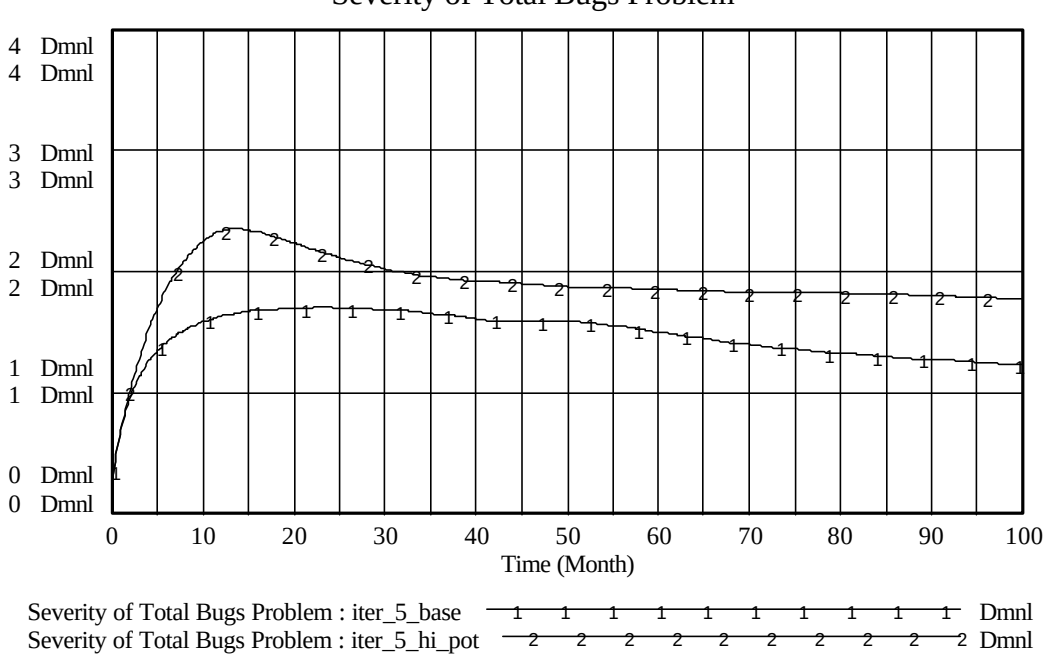


Figure 4.148. OSSD Model (Iteration V) High Potential Run - Severity of Total Bugs Problem

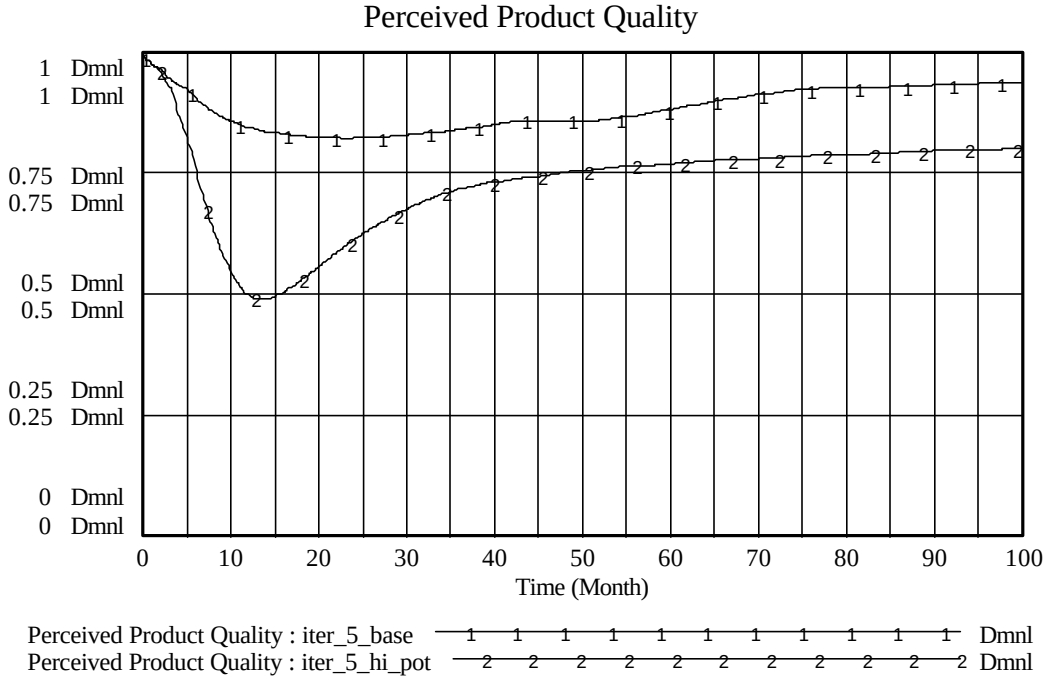


Figure 4.149. OSSD Model (Iteration V) High Potential Run - Perceived Product Quality

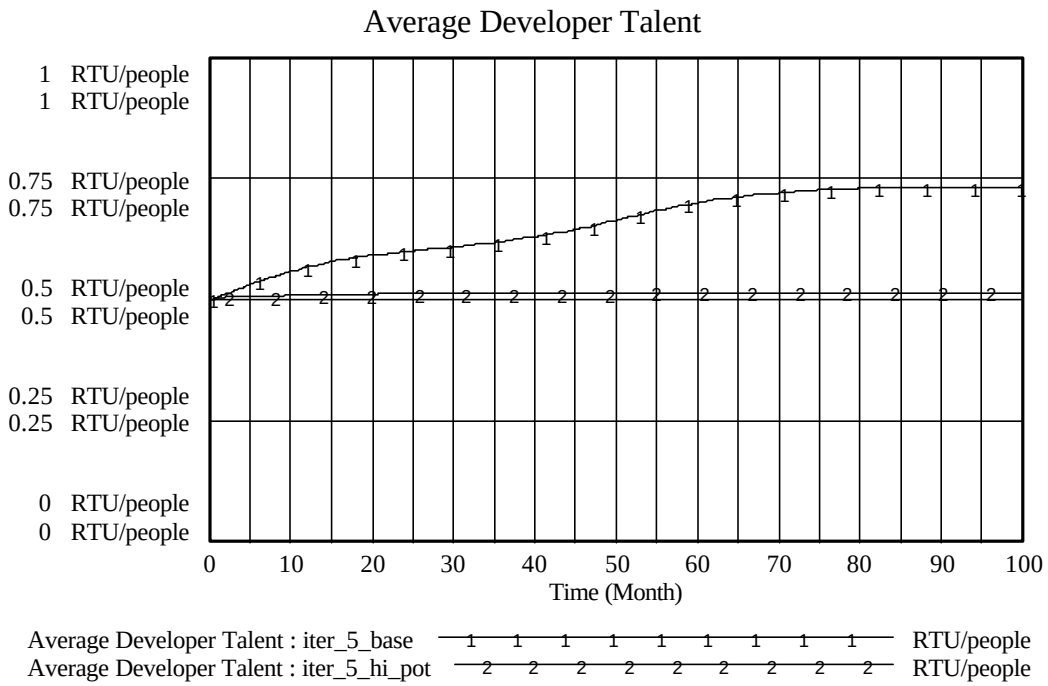


Figure 4.150. OSSD Model (Iteration V) High Potential Run - Average Developer Talent

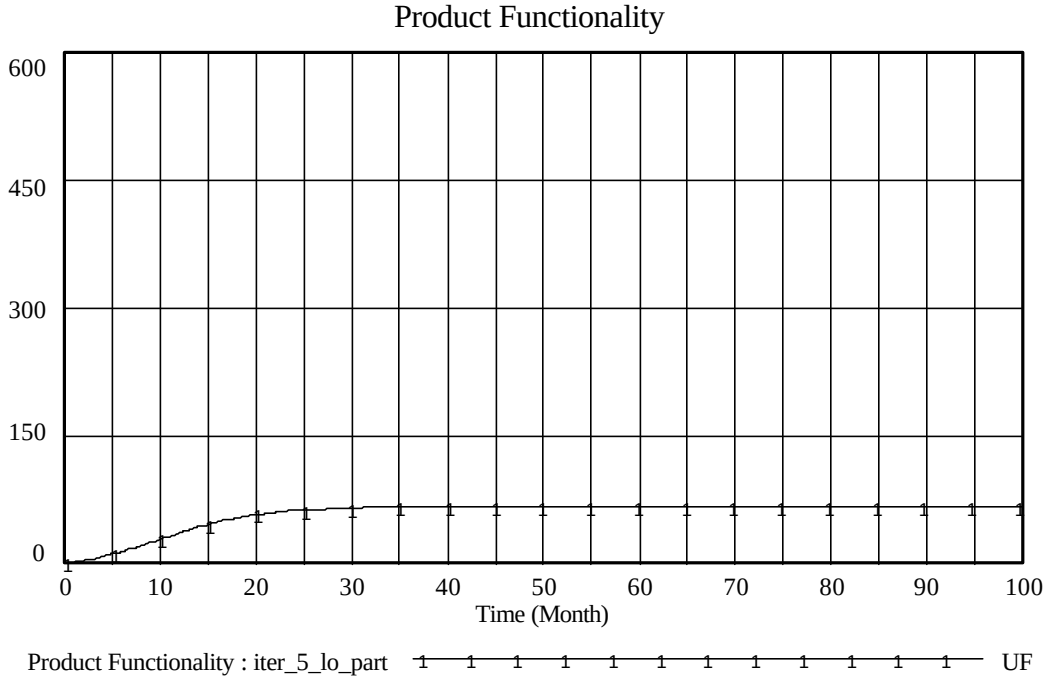


Figure 4.151. OSSD Model (Iteration V) Low Participation Run - Product Functionality

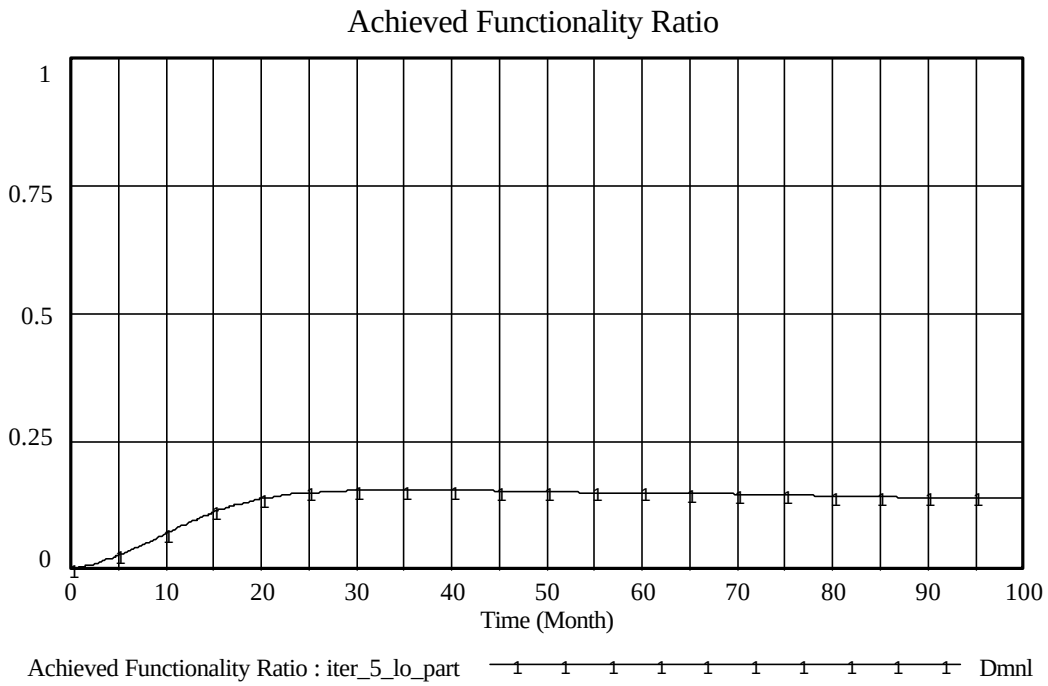


Figure 4.152. OSSD Model (Iteration V) Low Participation Run - Achieved Functionality Ratio

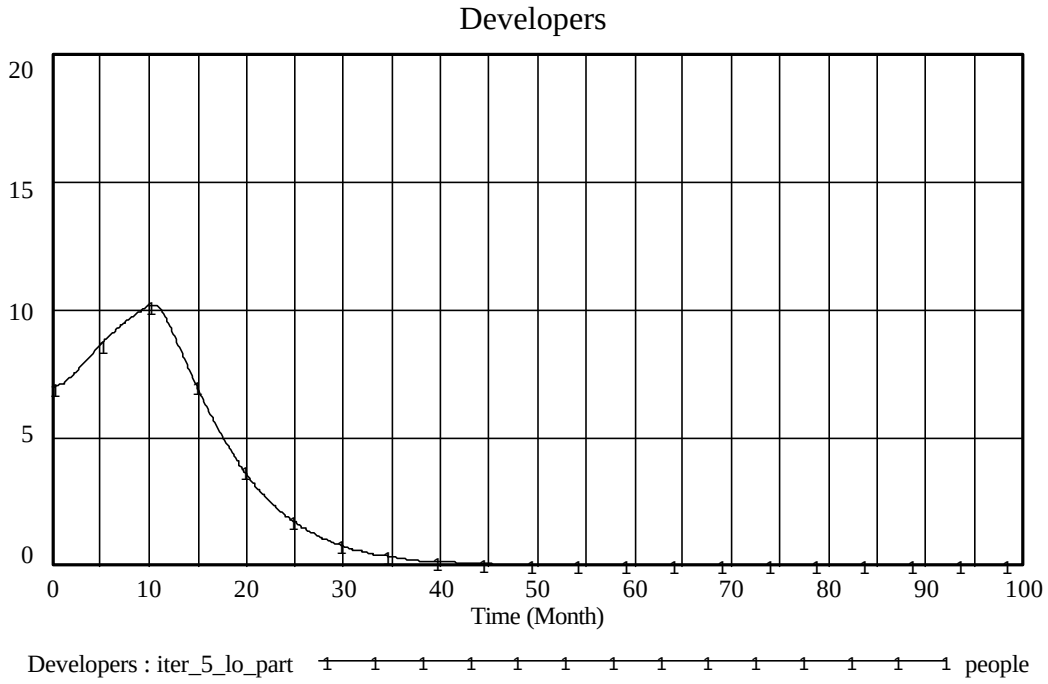


Figure 4.153. OSSD Model (Iteration V) Low Participation Run - Developers

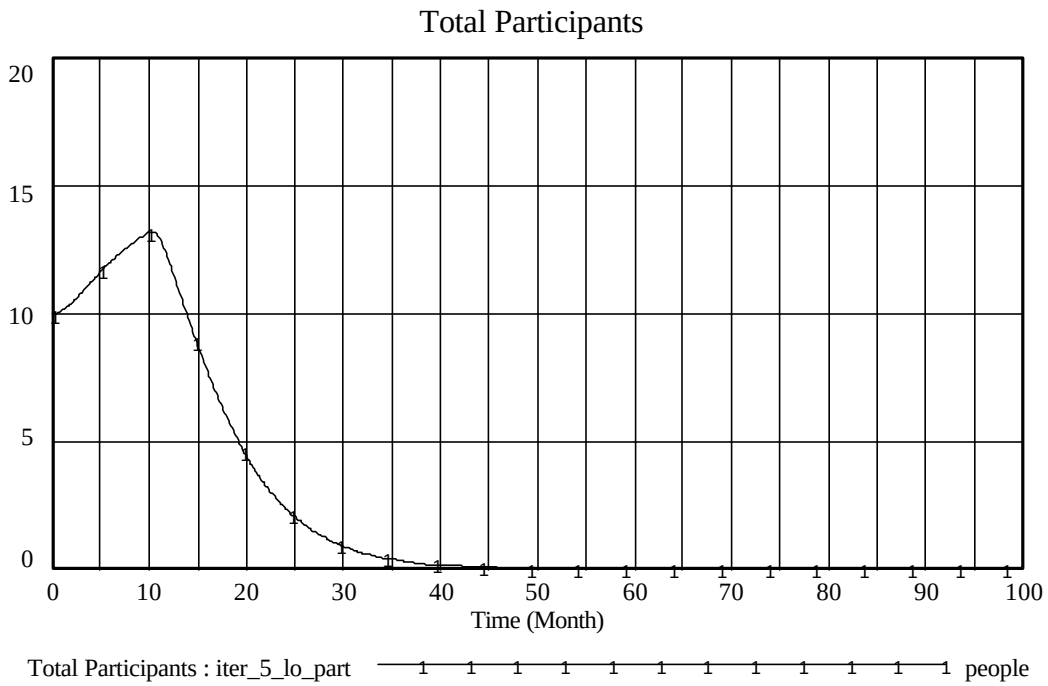


Figure 4.154. OSSD Model (Iteration V) Low Participation Run - Total Participants

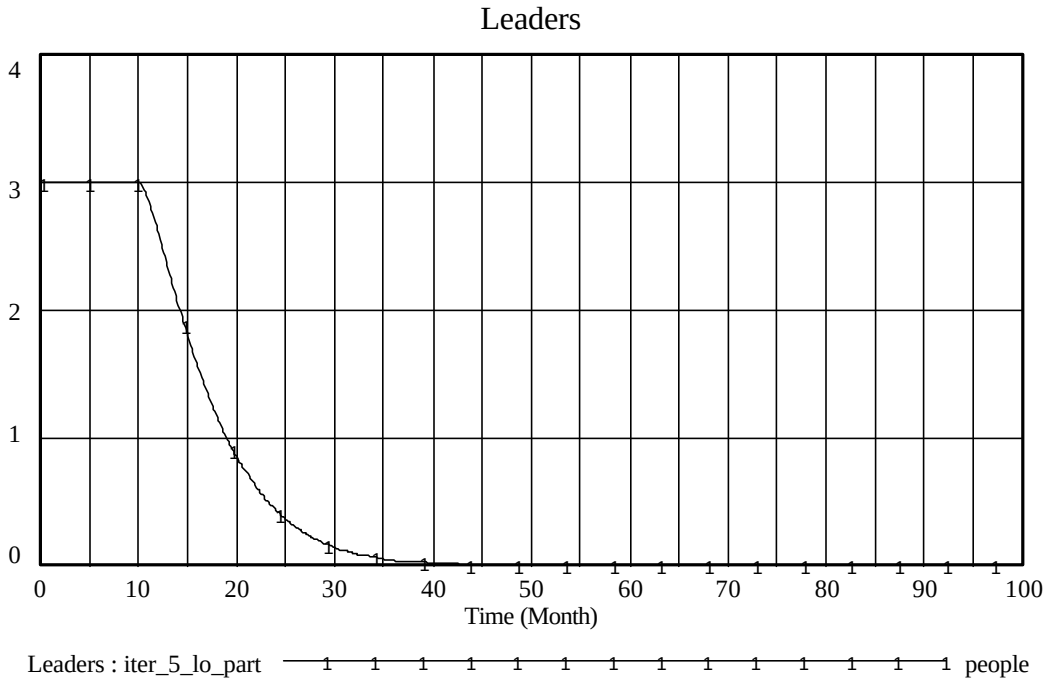


Figure 4.155. OSSD Model (Iteration V) Low Participation Run - Leaders

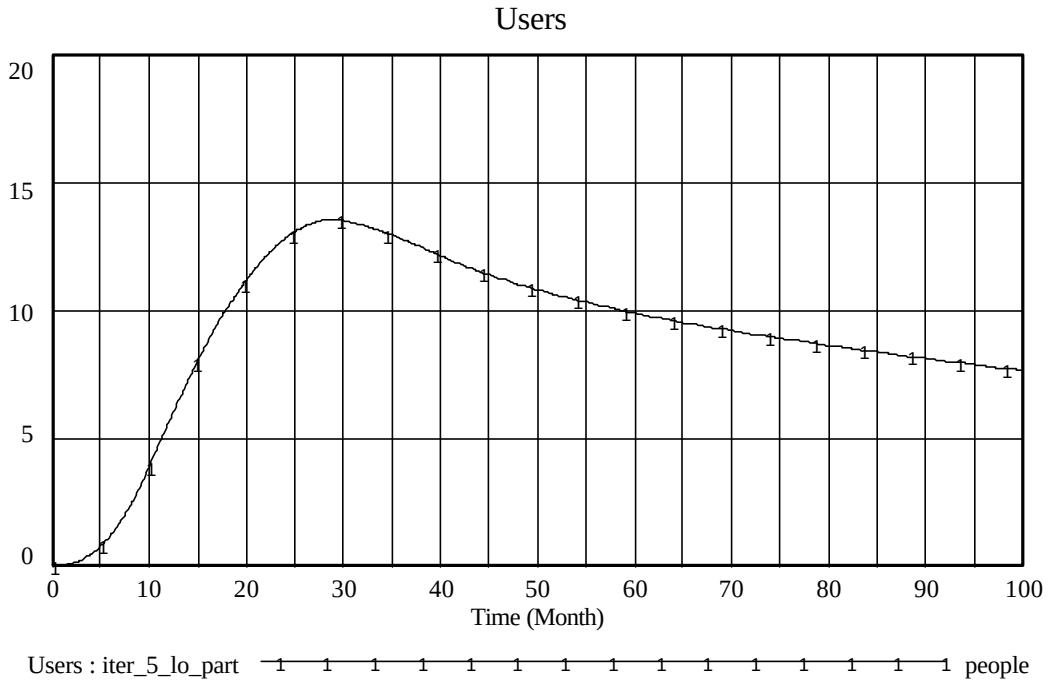


Figure 4.156. OSSD Model (Iteration V) Low Participation Run - Users

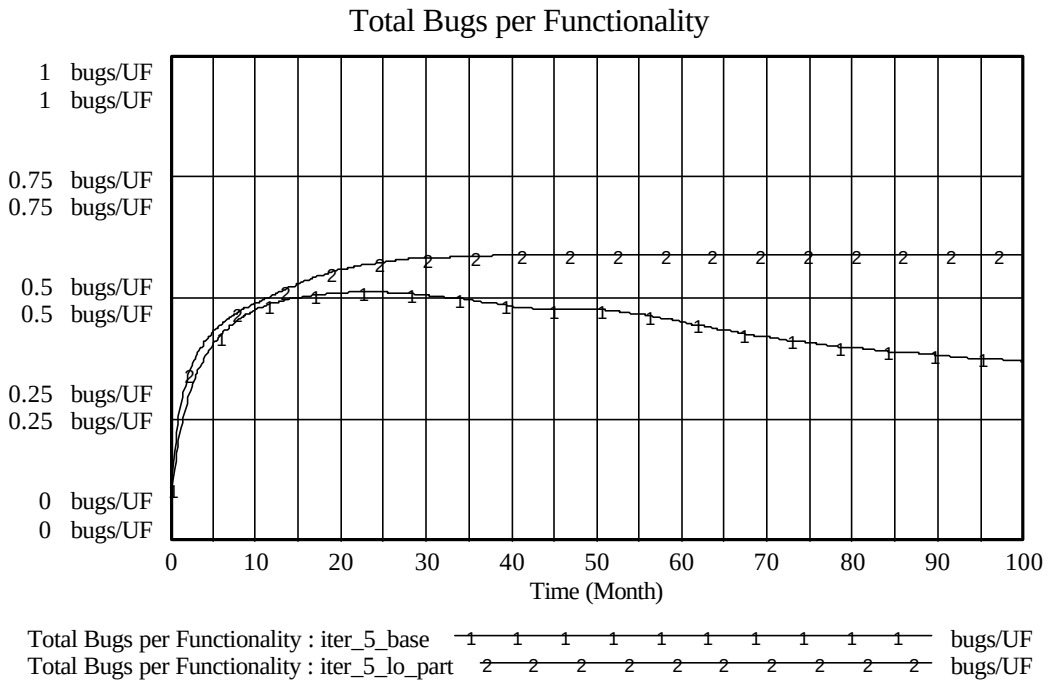


Figure 4.157. OSSD Model (Iteration V) Low Participation Run - Total Bugs per Functionality

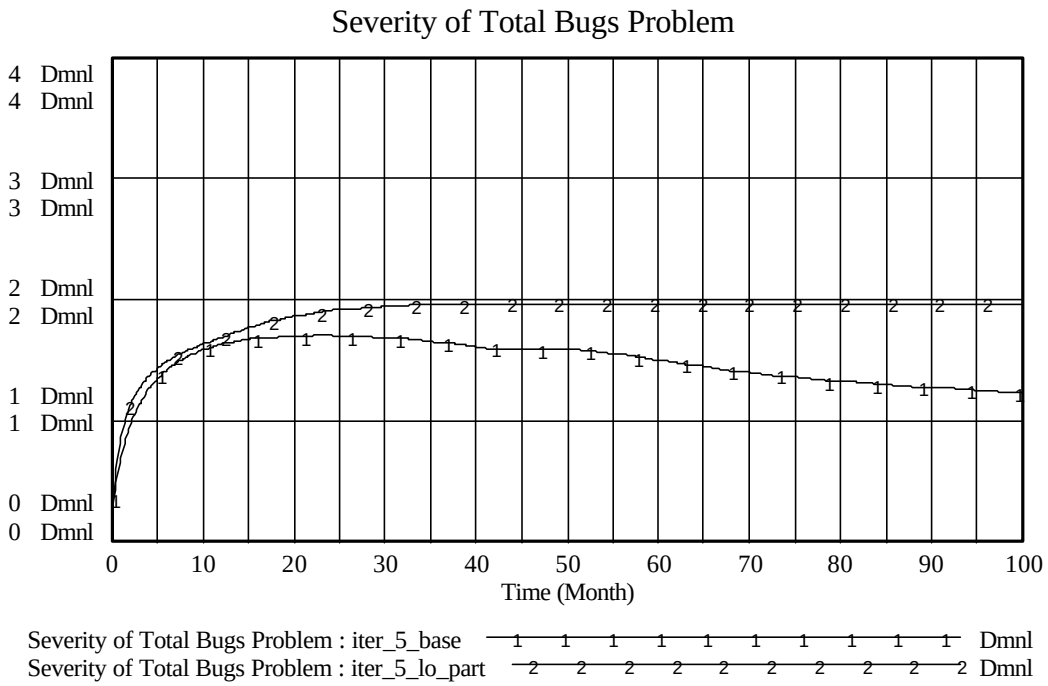


Figure 4.158. OSSD Model (Iteration V) Low Participation Run - Severity of Total Bugs Problem

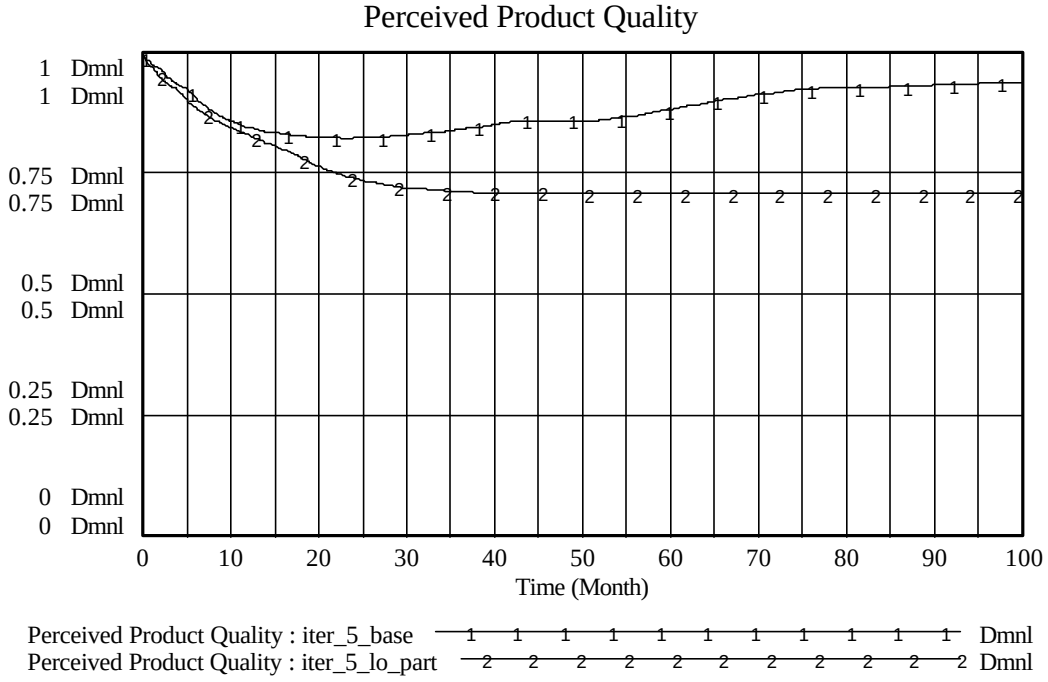


Figure 4.159. OSSD Model (Iteration V) Low Participation Run - Percevied Product Quality

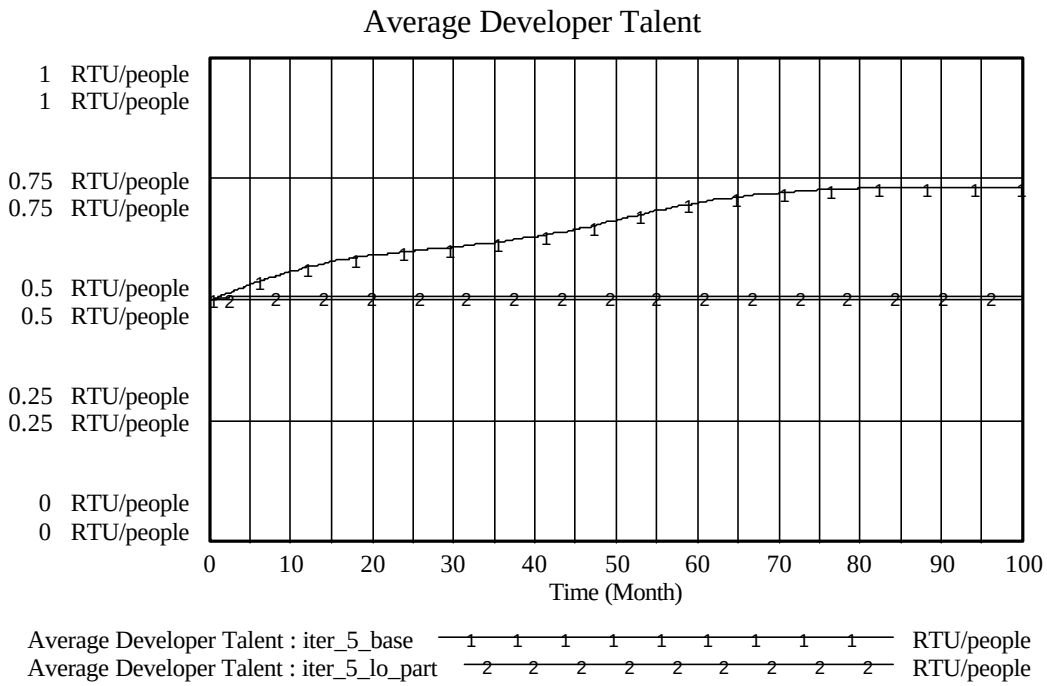


Figure 4.160. OSSD Model (Iteration V) Low Participation Run - Avergae Developer Talent

With the Iteration V version, the OSSD model reached a maturity level that provides adequate explanatory power for the purposes of this study. The model replicated product functionality accumulation, and growth of developer and user populations in successful, as well as unsuccessful open source software development communities. The model also replicated the effects of time pressure and quality on community growth. With the Iteration V version, four policy leverage points, namely debugging, coaching, barriers to entry and barriers to contribution were integrated into the model.

A small number of simulation runs under a limited variety of conditions were done during the model development stage. These runs showed that the model exhibited plausible and consistent behavior under normal conditions at each iteration. However, a more comprehensive model testing and analysis phase was needed to build confidence in

the model and to explore its behavior under different conditions and policy options before reaching a substantial set of both theoretical and practical implications. Chapter 5 summarizes the findings of the model testing and analysis phase, which built confidence in the model and provided critical implications about the model.

CHAPTER 5 -- MODEL TESTING AND ANALYSIS

5.1. Model Testing and Analysis Overview

The open source software development (OSSD) model reached its final stage of evolution within the scope of this study with the Iteration V version. Each iteration involved aspects of model evaluation and testing as well as the adding of new structure. As discussed in the methodology chapter, the process of testing a system dynamics model is generally referred to as "confidence building," rather than "validation." The rationale is that a model cannot be identified as either "valid" or "invalid," but rather there is a level of validity, or better yet, a confidence level for a given model. Also, "validation" is a static activity in its plain "pass or fail" mode. However, "confidence building" implies an iterative process of improving the model based on the model analysis findings.

Several authors suggested slightly different sets of tests for analyzing system dynamics models (Richardson and Pugh 1981 pp.313-318, Barlas 1989, Forrester and Senge 1996 pp.414-434, Sterman 2000). Some of the tests are common to all the suggested sets. A "complete" set of confidence building tests consists of many types of tests. Forrester and Senge (1996 pp.414-434) alone suggest 17 types of tests for analyzing a system dynamics model. Some of these tests involve comparing the behavior of the model to real data generated by the actual system to test whether the model replicates the real world behavior of the system it represents. Not all of the suggested tests were performed on the OSSD model. The rationale for that was that the OSSD model was not envisioned as an end product of this study. The model was used as a tool to integrate the implications of relevant literature with the observations and mental models of the members of an actual open online collaboration community in order to reach a dynamic

feedback framework, which serves as a theoretical basis for future research on the topic. Thus, applying an exhaustive set of tests to the model in an effort to refine it beyond a certain point was not considered relevant within the scope of this study. However, a future study that focuses mostly on the OSSD model itself should include a more exhaustive set of confidence building tests. This chapter summarizes the findings of three common tests applied to the model: extreme condition tests, sensitivity analysis tests, and policy analysis tests. The empirical component of this study, which involved interviews with system dynamics K through 12 instructional material development community members, can also be viewed as a test for building confidence in the model and improving it. The findings of the interviews are discussed in Chapter 6.

5.2. Base-Case Run

The base-case run of the model involves the simulation of the model with the default, or most likely parameter values. The base-case run mainly serves two purposes. Its first purpose is to test whether the model generates plausible behavior under default conditions. The base-case also serves as a reference, against which the non-default runs such as extreme condition and policy analysis runs can be compared.

The base case of the open source software development (OSSD) model was a run that represents a project for a software product with an initial limit of 400 units of functionality (UF.) The initial number of developers on the project is seven, and the number of leaders is three, creating 10 total participants in the projects. The initial number of users of the software is zero.

After the project starts, a number of developers join the project, increasing the total number of developers up to 14 people at around month 17 of the project. (See Figure

5.1.) The number of developers stays almost the same until around month 30. After that the number of developers starts to decline due to decreasing opportunities for contribution. Note that at around month 30 product functionality reaches almost 70% of the limit on product functionality. (See Figure 5.2.) The decrease in the number of developers continues until month 80, when all the developers have left the project.

The number of users starts to increase visibly after month 15, when the achieved functionality ratio reaches 0.3. The increase happens in an exponential fashion until about month 35, when the achieved functionality ratio reaches 0.75, and continues in an asymptotic fashion after that point (See Figure 5.1.)

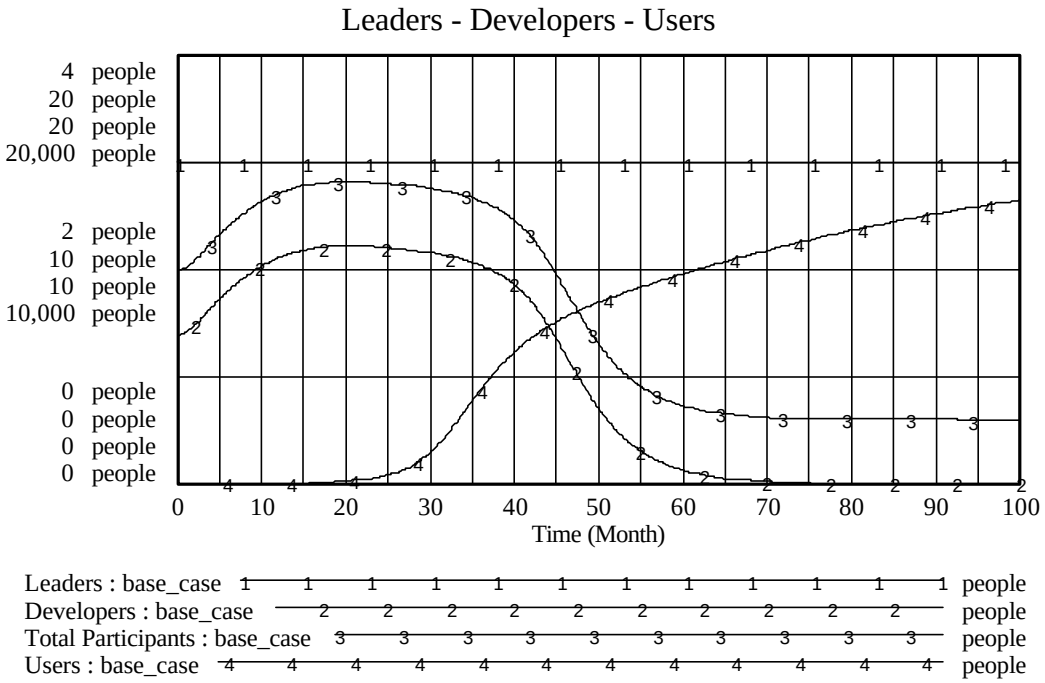


Figure 5.1. Leaders, Developers and Users under Base Case Conditions

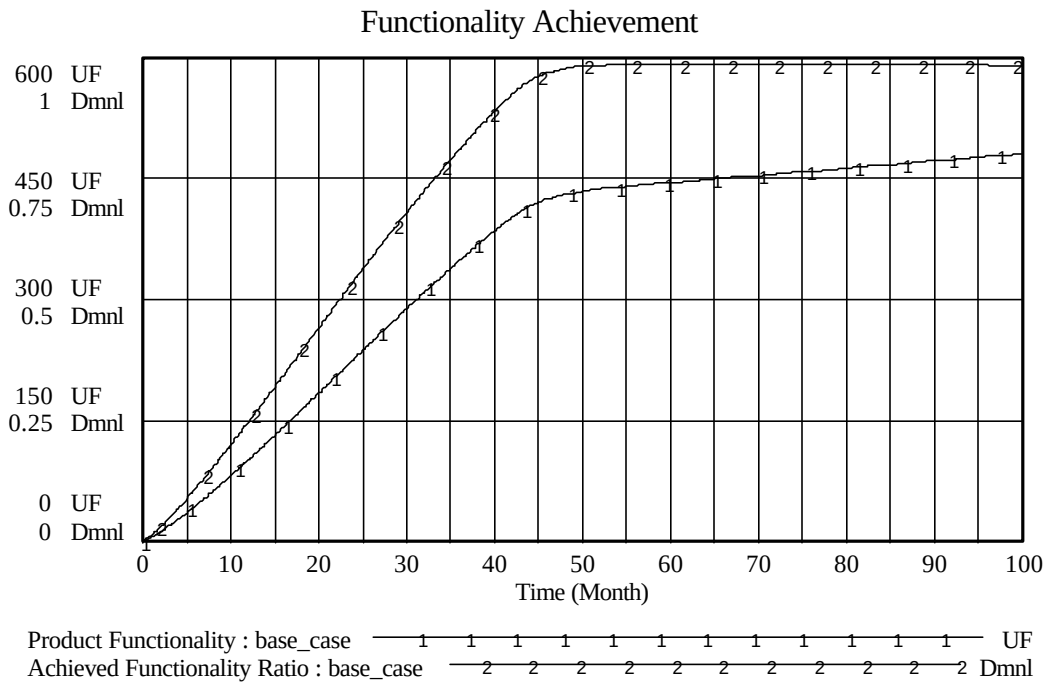


Figure 5.2. Functionality Achievement under Base Case Conditions

In the first quarter of the base case run, the product quality exhibits a decline, due to an increase in the number of bugs per unit of functionality. (See Figure 5.3.) The number of bugs per functionality increases until about month 25 as new bugs are introduced by production. During that period the participants (leaders and developers) focus mostly on adding functionality to the product, rather than maintaining its quality. As the severity of the total bugs problem increases, the participants feel an increased pressure for bug detection and bug fixing. After month 25 debugging efforts reach a point where the number of bugs per functionality starts to decrease, thus improving the perceived product quality. (See Figure 5.3.)

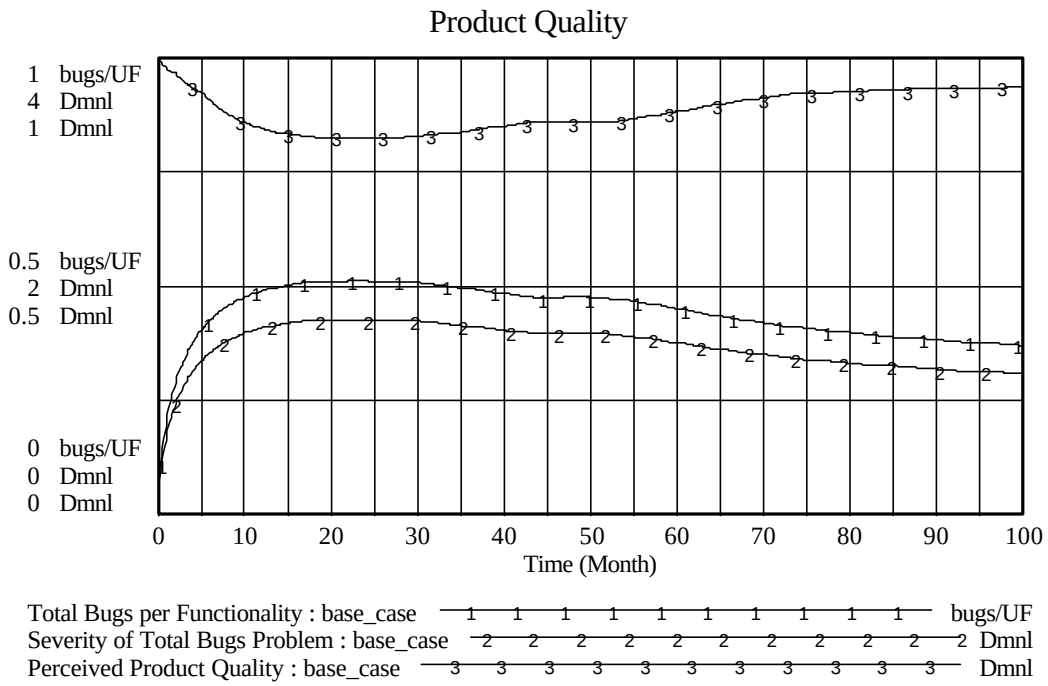


Figure 5.3. Product Quality under Base Case Conditions

Average Developer Talent steadily increases until around month 80, as new developer talent is built through coaching. (See Figure 5.4.) The increase stops after month 80, since almost all the developers have left the project as it reached a functionality saturation point, and there is no more coaching taking place within the community. (See Figure 5.4.)

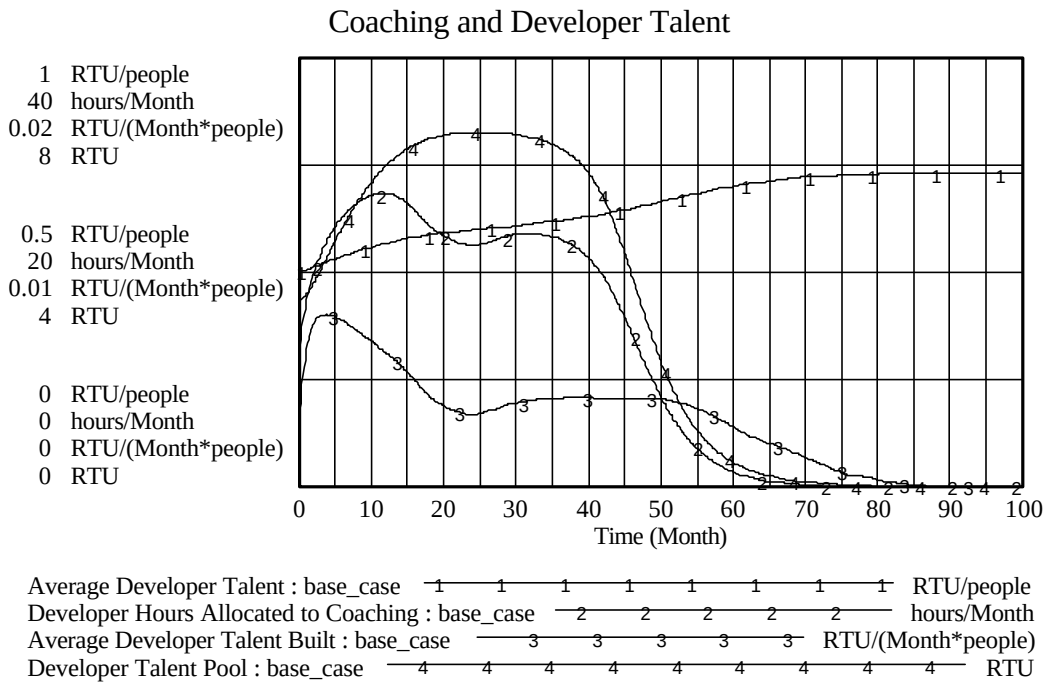


Figure 5.4. Coaching and Developer Talent under Base Case Conditions

5.3. Extreme Condition Runs

Extreme condition runs test whether the model behaves as expected under conditions that deviate extremely from normal conditions. The idea that lies behind the extreme condition tests is that model behavior under extreme conditions is far more predictable that under normal conditions. As a trivial example, the behavior of any given human body under extreme temperature conditions such as below the freezing point or above the boiling point is far more predictable than its behavior under normal conditions, i.e. between 60° to 80° Fahrenheit. The model of a human body may exhibit a "shivering," "sweating" or "total comfort" behavior between 60° to 80° Fahrenheit; and all of these behaviors can be argued to be plausible for *some* actual human bodies. Therefore, it may not be possible to refute the model based on its behavior under such conditions. However, the model should exhibit a distinctive "dying" behavior under

freezing, or boiling conditions. If the model does not exhibit that distinctive behavior, it should be refuted in its current state. Some of the most insightful extreme condition runs performed on the OSSD model are discussed below.

5.3.1. No Developers

The model was run under the condition of no developers throughout the project lifetime. The initial number of developers was set to zero. Also, the refusal ratio was set to 1 to ensure no incoming developers. The run yielded expected results under the given condition. The number of developers stayed at zero throughout the project. (See Figure 5.5.) Due to lack of developers, only leaders built product functionality under this extreme condition run, and consequently the achieved functionality ratio could not reach a point that could sustain the community. (See Figure 5.6.) The very limited amount of achieved functionality attracts an extremely small number of users, and the number of users increases until month 72. However, after that point even that small number of users starts to decline, as the relative functionality of the product decreases. (See Figure 5.5.) The failure to achieve a viable amount of product functionality caused leaders to leave the community starting at around month 13. By month 70 all the leaders had left the community. (See Figure 5.5.)

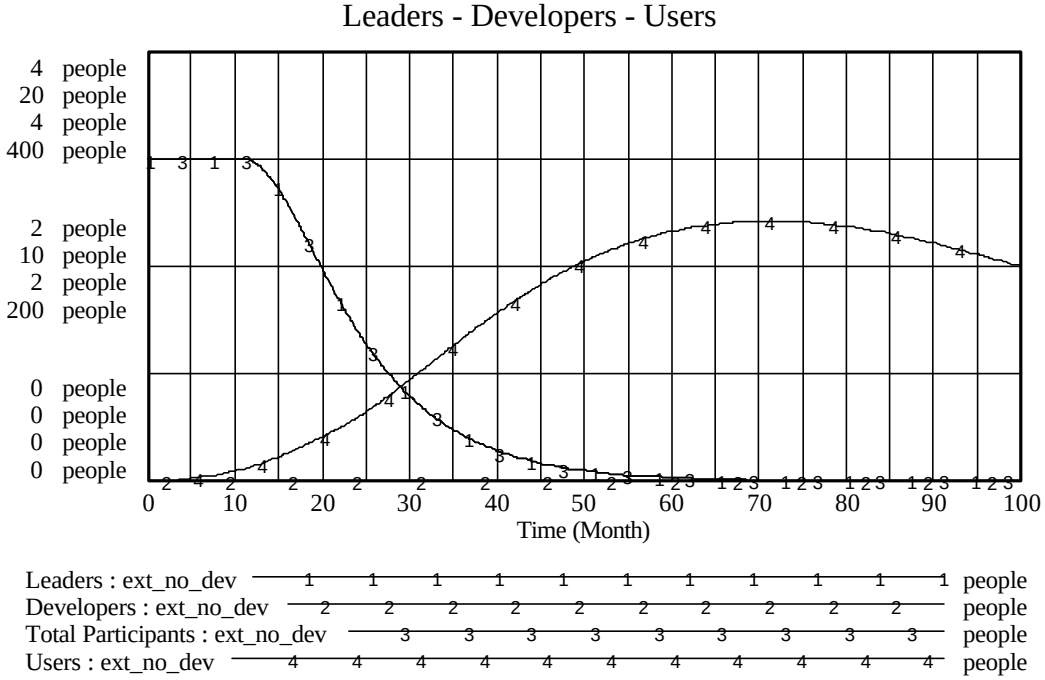


Figure 5.5. Leaders, Developers and Users under "No Developers" Extreme Case

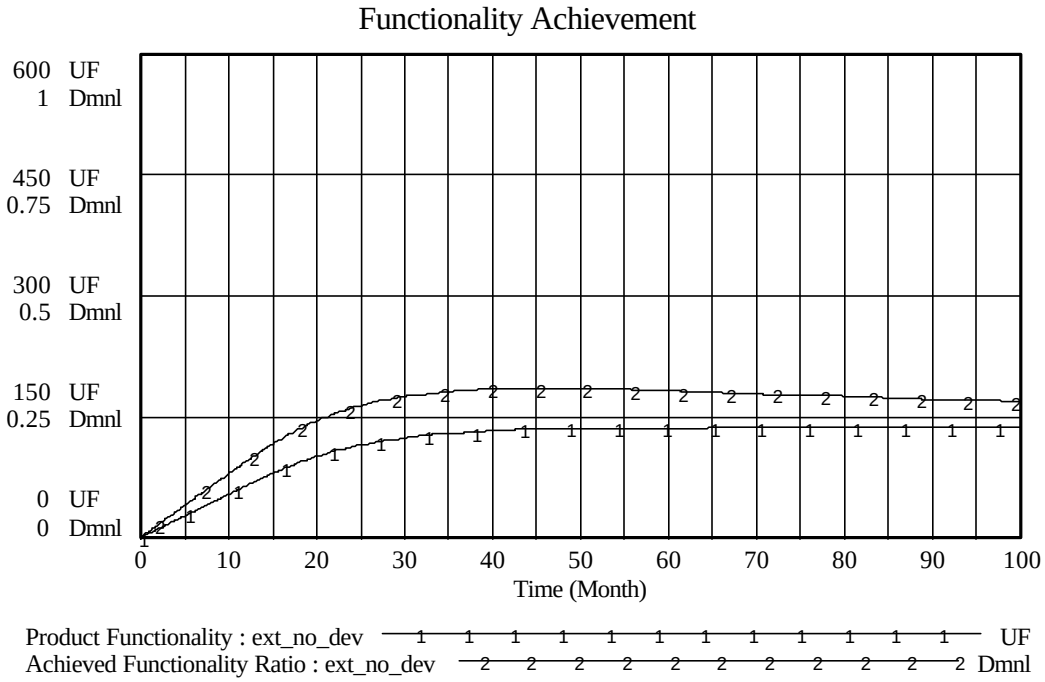


Figure 5.6. Functionality Achievement under "No Developers" Extreme Case

The OSSD model assumes that production by leaders introduces a much smaller number of bugs compared to production by developers, as discussed in the model description in Chapter 5. Since no production by developers took place under this extreme condition, the product quality stayed very high throughout the simulation run. The small number of bugs introduced by leaders could be held under control through a limited debugging effort. (See Figure 5.7.)

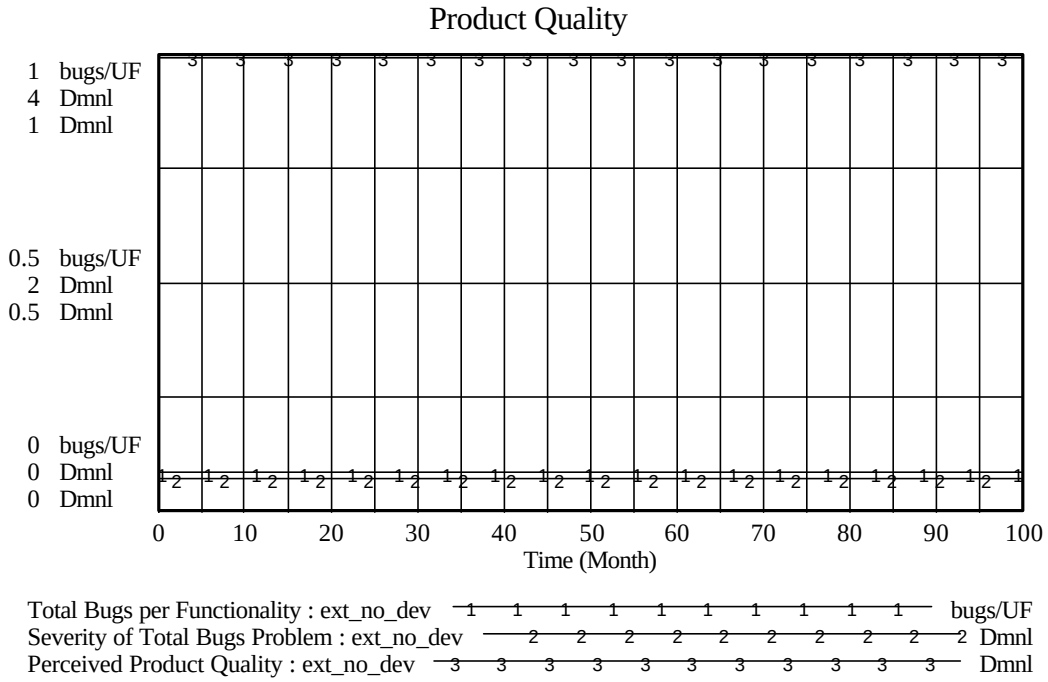


Figure 5.7. Product Quality under “No Developers” Extreme Case

Since there were no developers in the community, the overall developer talent pool and average developer talent stayed at zero throughout this run. Also, no coaching took place in this run, as expected. (See Figure 5.8.)

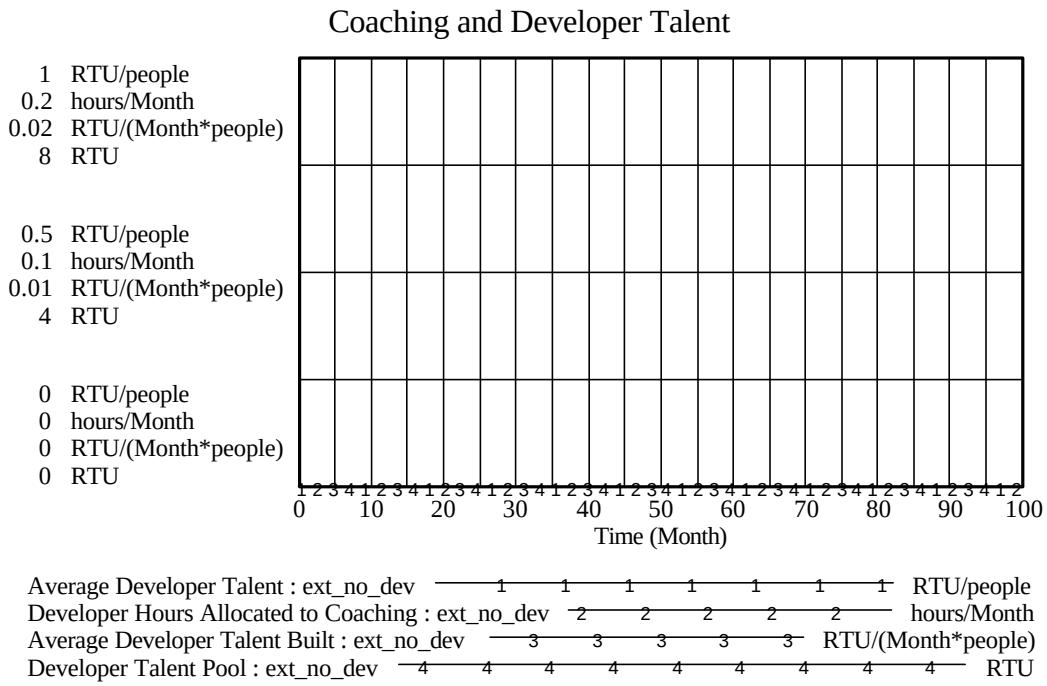


Figure 5.8. Coaching and Developer Talent under "No Developers" Extreme Case

5.3.2. No Leaders

Another extreme condition applied to the model was the case with no leaders in the community. The number of developers increased slightly at the beginning, but started to decline rapidly after month 10, dissolving the community within the first 25 months under this condition (See Figure 5.9.)

The product could attract an extremely small number of 13 users by month 18, which started to decrease after that point. (See Figure 5.9.) This was due to the very limited level of functionality achievement, which was caused by the lack of development by leaders. (See Figure 5.10.)

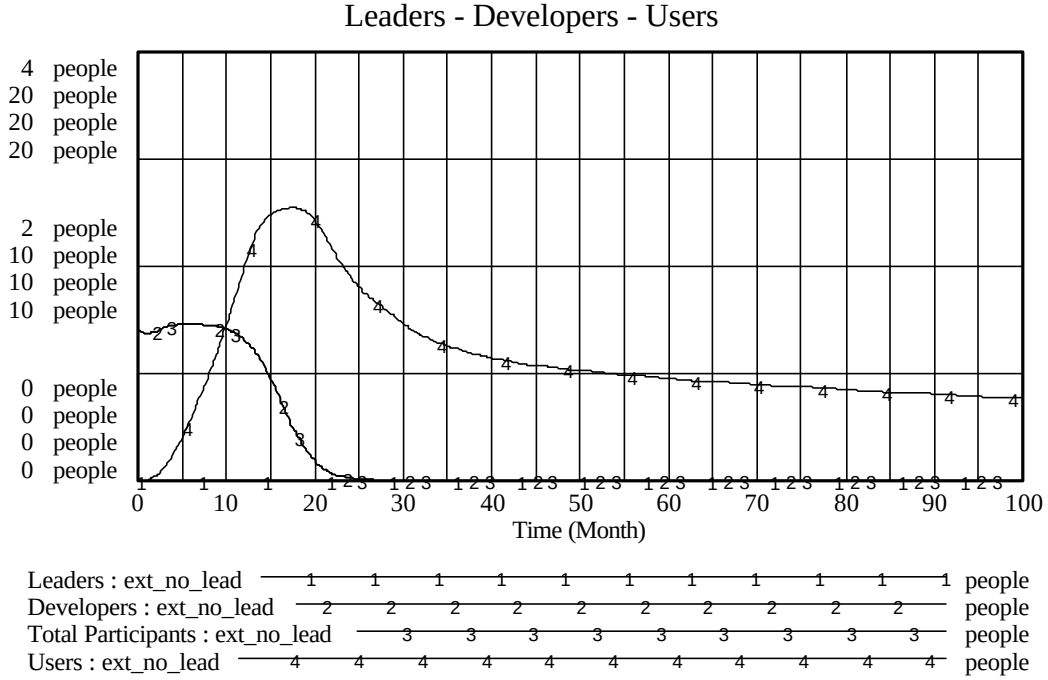


Figure 5.9. Leaders, Developers and Users under “No Leaders” Extreme Case

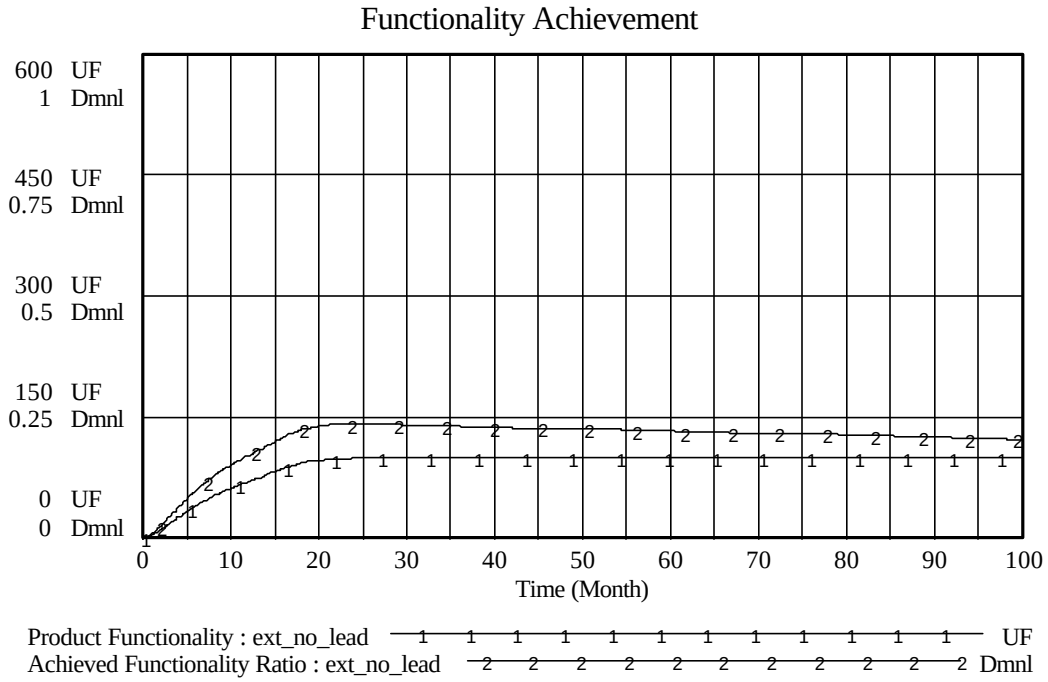


Figure 5.10. Functionality Achievement under “No Leaders” Extreme Case

Product quality started at a considerably low level and decreased even further at the beginning of the project. That was due to the lower average quality of the production, which was done solely by the developers. The OSSD model assumes that the bug detection and bug fixing skills of developers are lower than those of leaders. The already bad bugs-per-functionality problem was worsened by the lack of effective debugging by leaders. (Figure 5.11.)

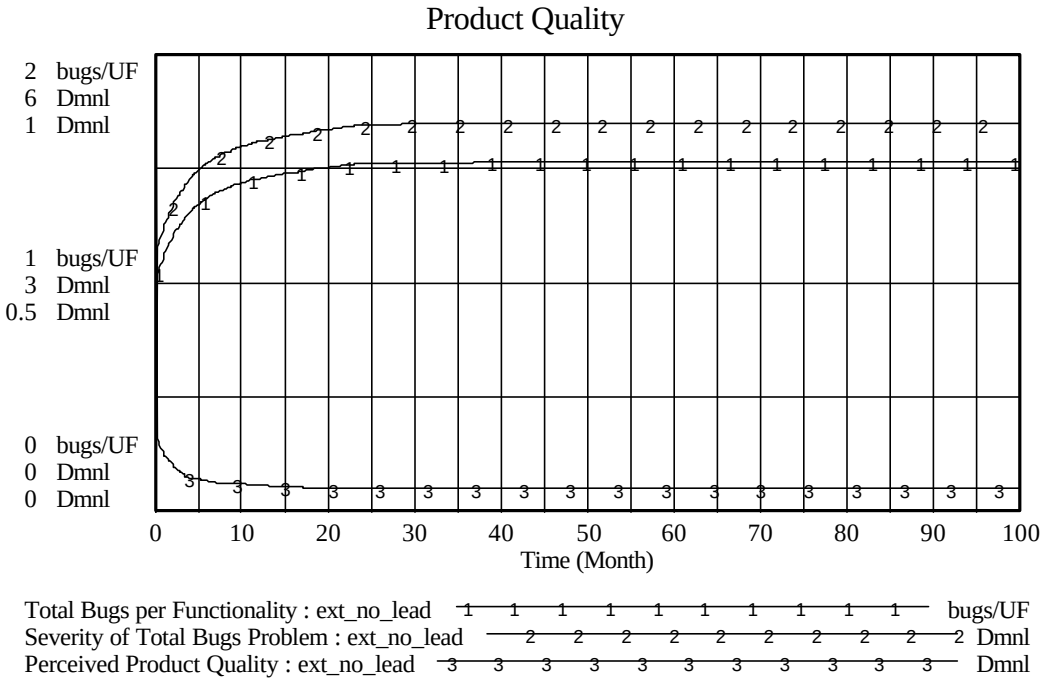


Figure 5.11. Product Quality under “No Leaders” Extreme Case

Relatively low and stagnant developer talent was another factor that worsened the quality problem in this run. The average developer talent started lower, due to the lack of a selecting process, which is normally carried out by leaders. Also, the average developer talent did not increase at all, since there were no leaders to coach the developers. (Figure 5.12.)

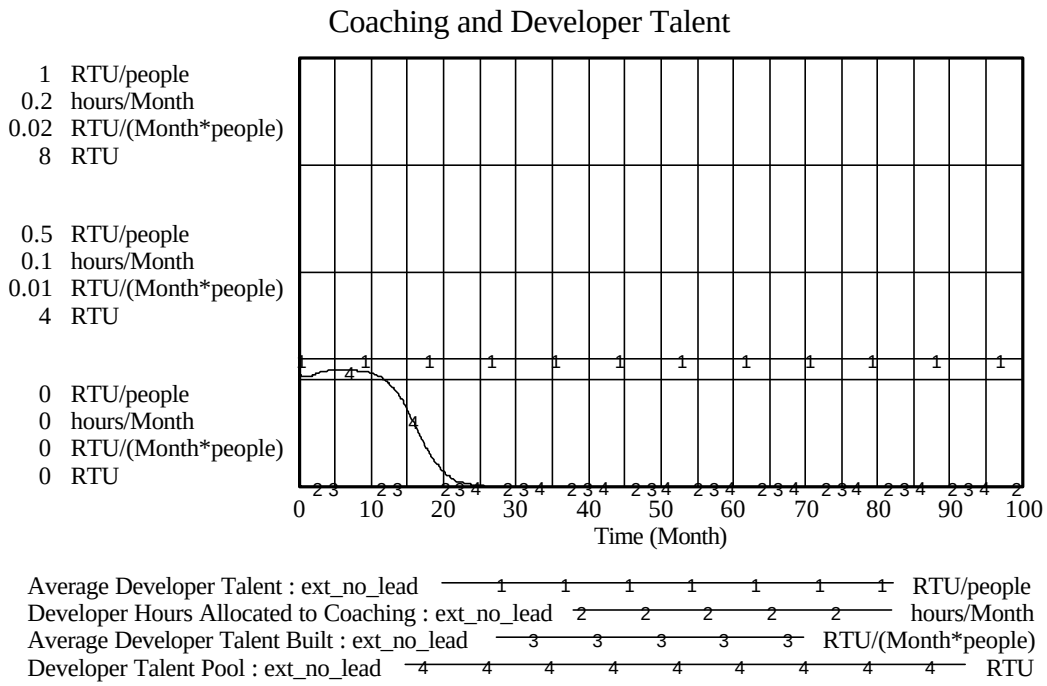


Figure 5.12. Coaching and Developer Talent under "No Leaders" Extreme Case

5.3.3. No Participants

The "no developers" and "no leaders" extreme cases were combined in another extreme condition run. This time, the community started with no participants at all, no developers *and* no leaders. Also the incoming developers flow was set to zero. All the population stayed at zero throughout the run (See Figure 5.13.) As expected, no production, no debugging and no coaching took place. Functionality stayed at zero. (See Figure 5.14.) The number of users stayed at zero, too, since there could be no users for a non-existent product (See Figure 5.13.) Only the product quality stayed at 1, since there were no bugs, and consequently no bugs problem. (See Figure 5.15.) Average developer talent, too, stayed at zero, since there were no developers. (See Figure 5.16.)

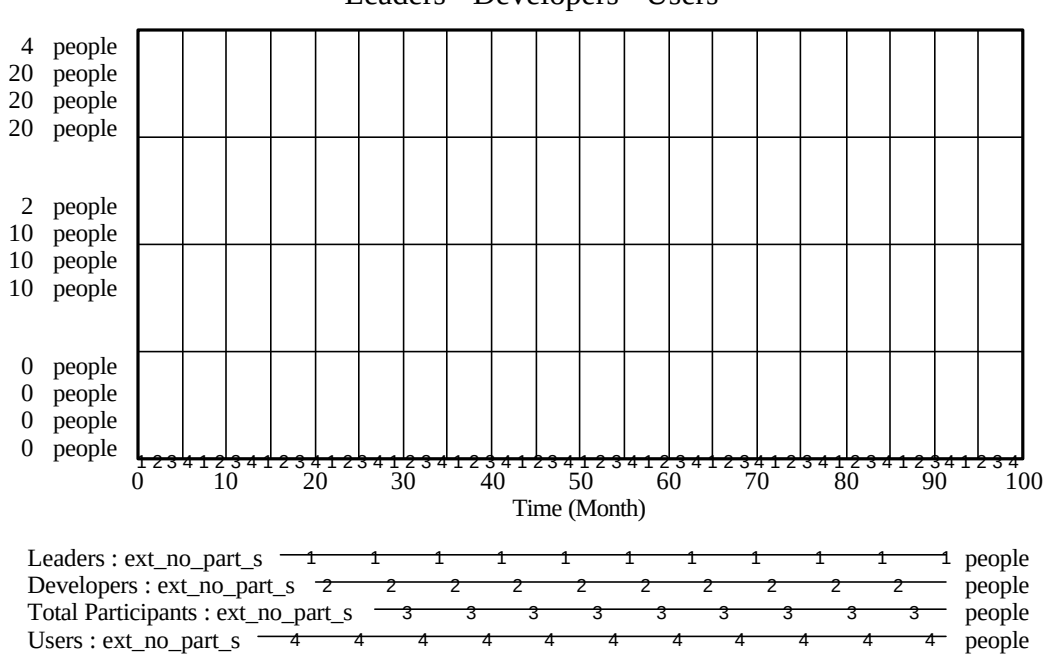


Figure 5.13. Leaders, Developers and Users under "No Participants" Extreme Case

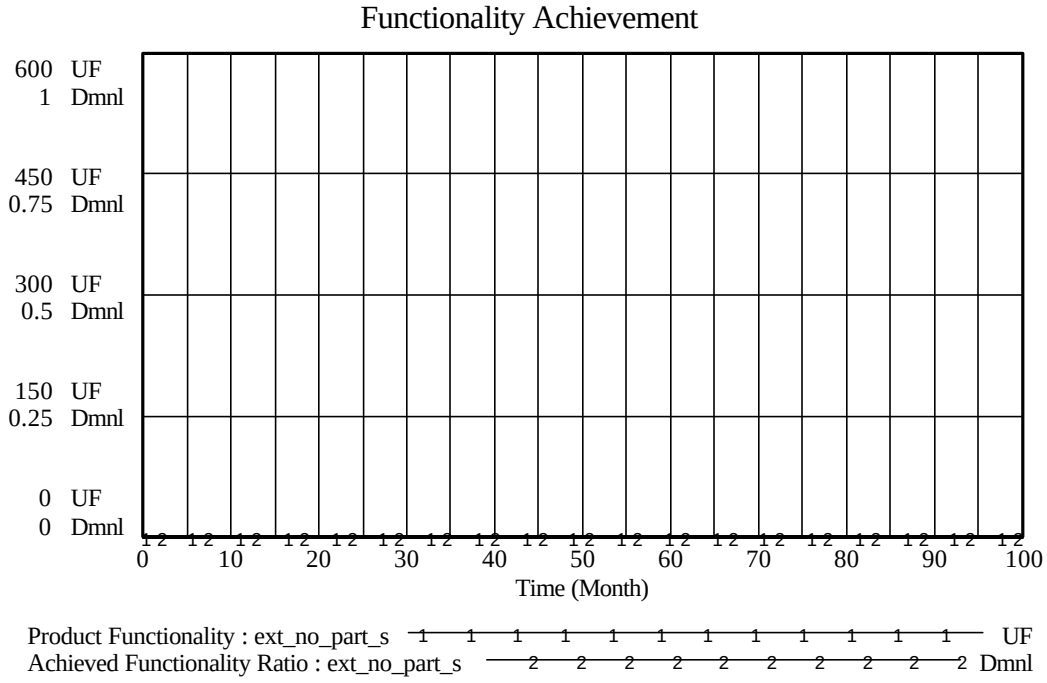


Figure 5.14. Functionality Achievement under "No Participants" Extreme Case

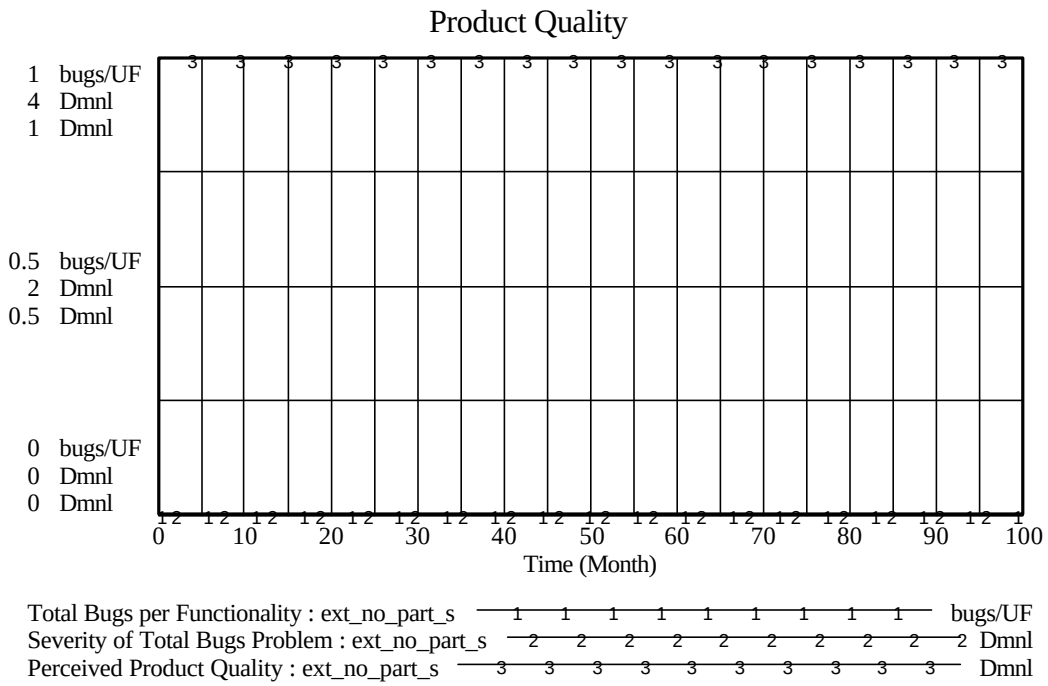


Figure 5.15. Product Quality under "No Participants" Extreme Case

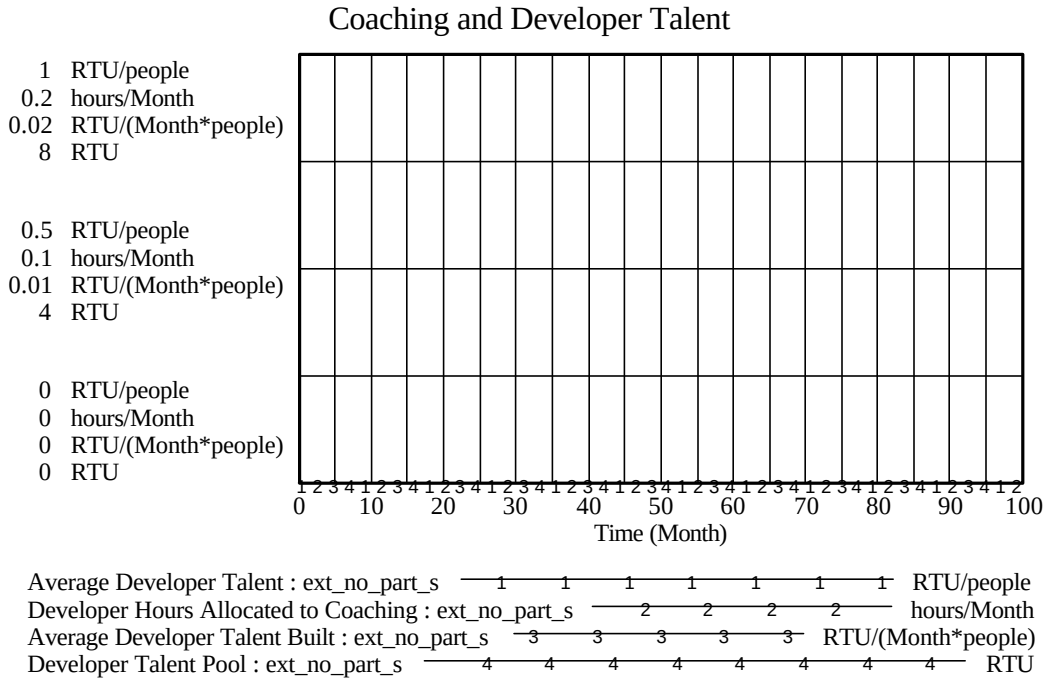


Figure 5.16. Coaching and Developer Talent under "No Participants" Extreme Case

5.3.4. No Developer Participation

"No developer participation" was a slightly different variant of the "no developers" extreme case. Here the community has an initial body of developers, and continues to recruit developers, but the developers do not participate in any activities within the community. The results of this run were very close to the results of the "no developers" run, with the exception of the behaviors of the number of developers, the overall developer talent pool and the average developer talent. (Compare Figures 5.5 - 5.8 and Figures 5.17 - 5.20.) The number of developers continued to increase until the community starts to dissolve at around month 13, and started to decrease after that until it reached zero at around month 80. (See Figure 5.17.) The average developer talent did increase, since the developers did not participate in coaching. (See Figure 5.20.)

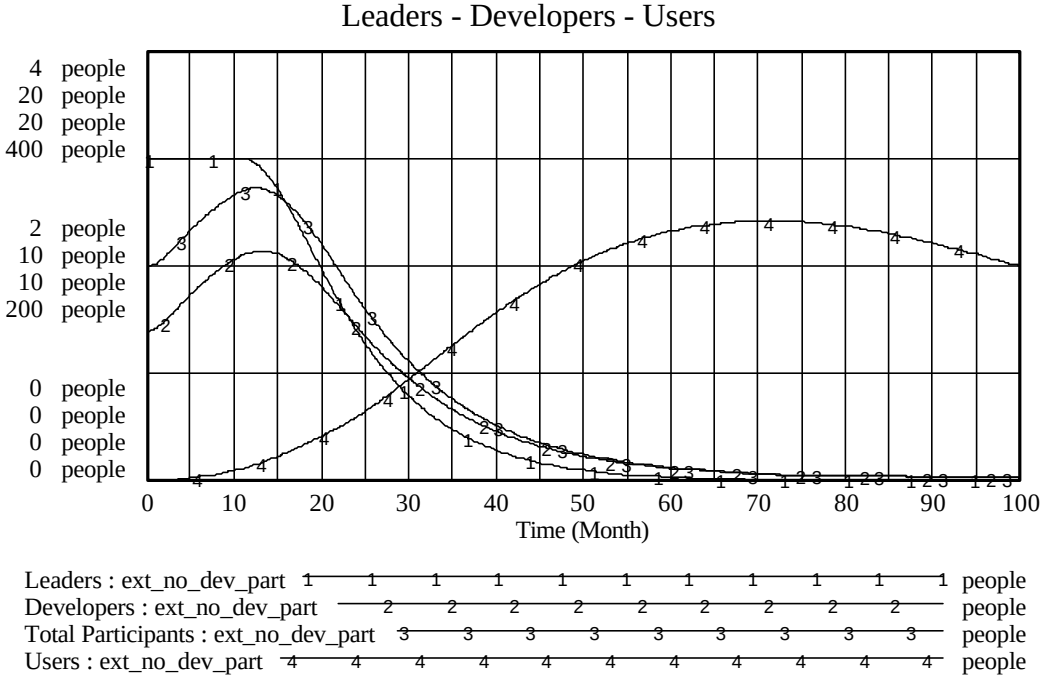


Figure 5.17. Leaders, Developers and Users under "No Developer Participation" Extreme Case

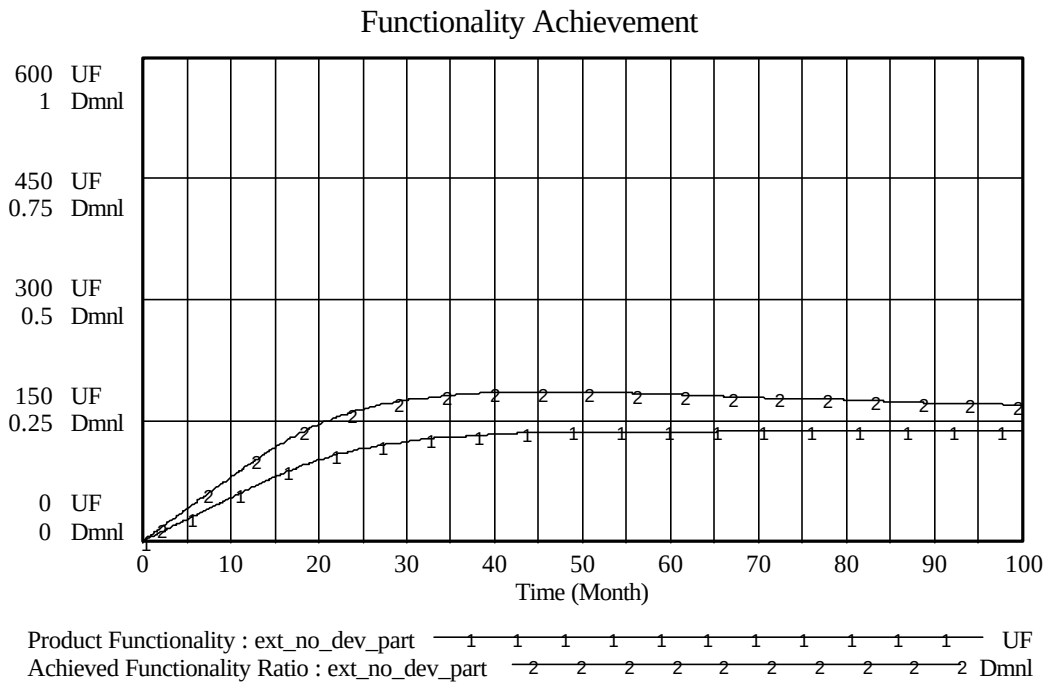


Figure 5.18. Functionality Achievement under "No Developer Participation" Extreme Case

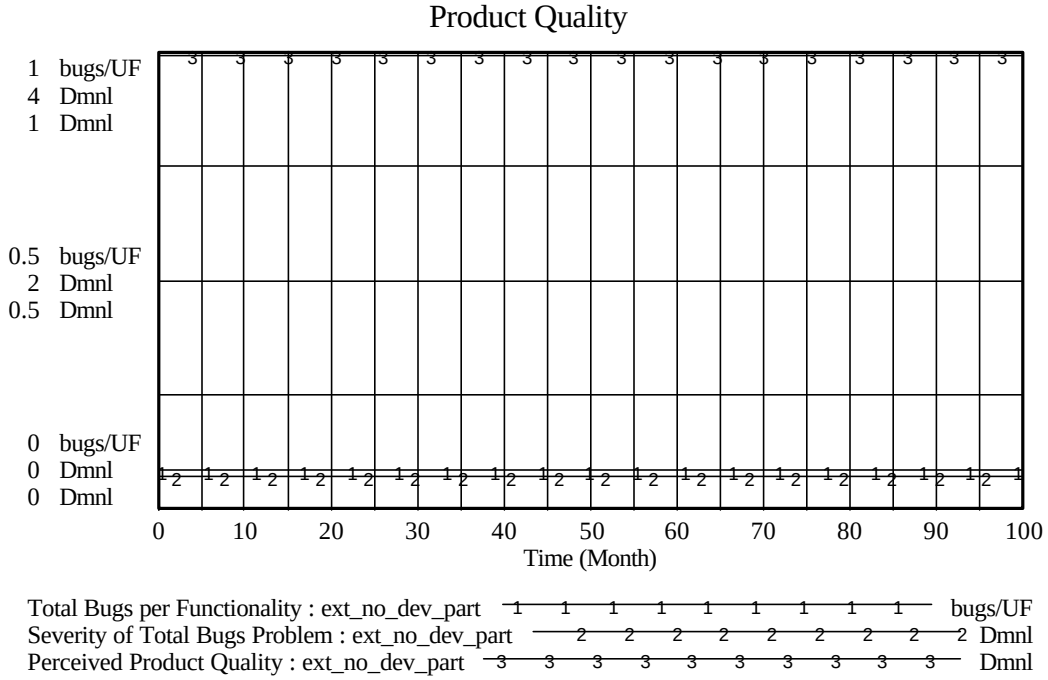


Figure 5.19. Product Quality under "No Developer Participation" Extreme Case

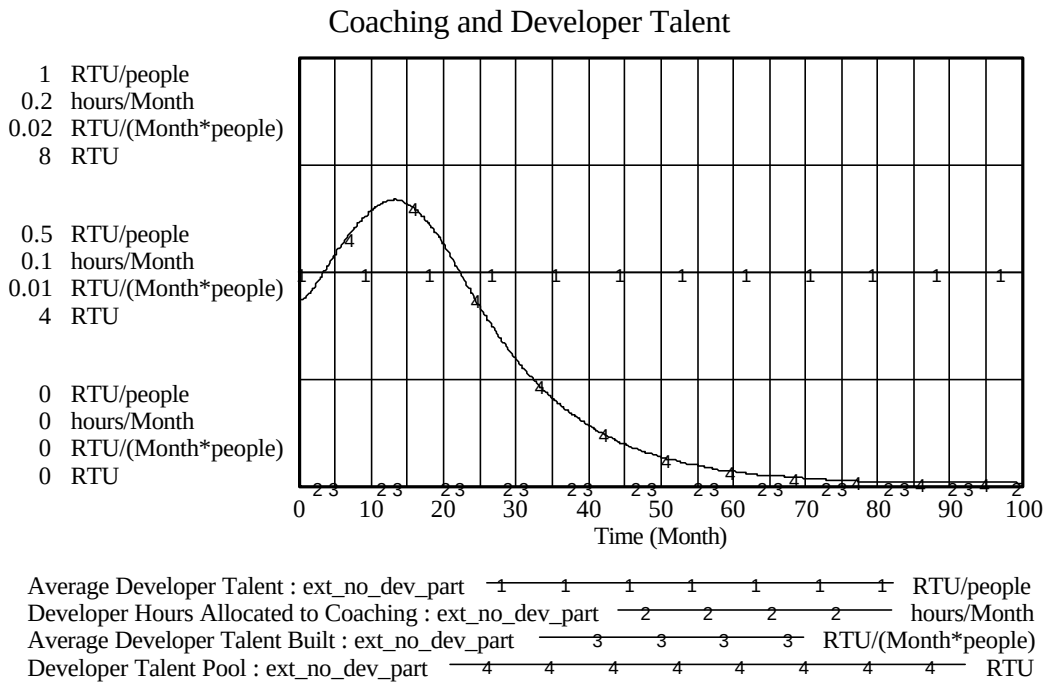


Figure 5.20. Coaching and Developer Talent under "No Developer Participation" Extreme Case

5.3.5. No Participation

In another extreme case applied to the model the community had both developers and leaders, but neither developers nor leaders participated in any activities within the community. Since there was no participation, no production was created and thus no functionality growth was achieved. (See Figure 5.21.)

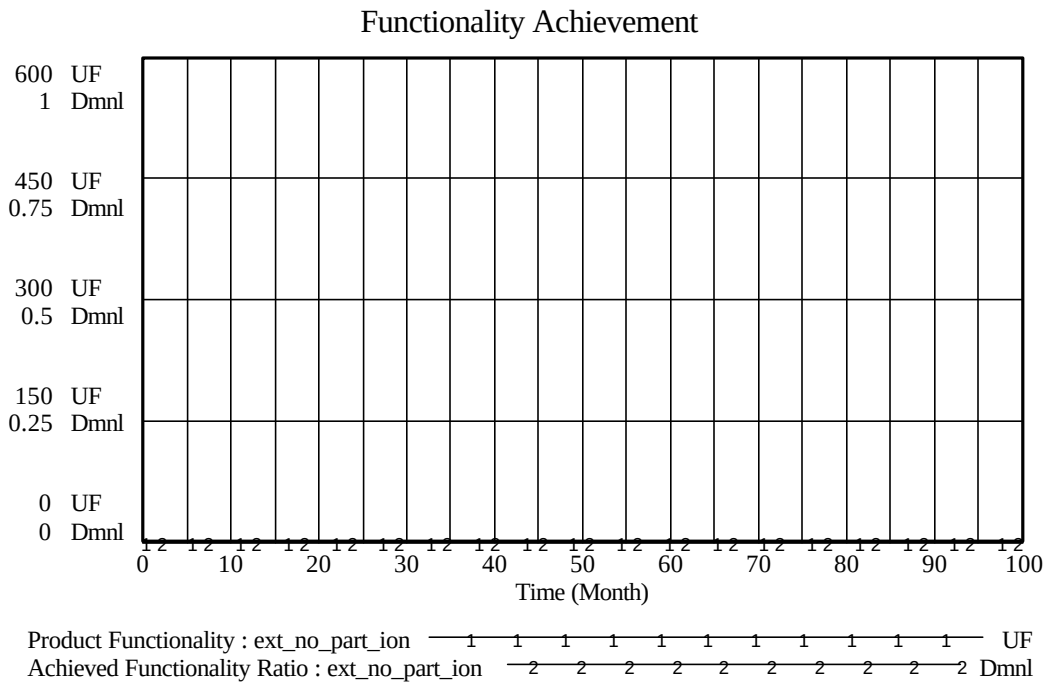


Figure 5.21. Functionality Achievement under "No Participation" Extreme Case

Due to the lack of functionality achievement, both leaders and developers started to leave the community rapidly after approximately month 10. The number of developers continued to increase until that time, since new developers continued to join the community based on the expectations for future functionality growth. (For a discussion about the expected and achieved functionality ratios, and how they affect the attractiveness of the product for developers and users see Section 4.3, "Iteration II: Adding Time Pressure.") The number of users stayed at zero since there was no functionality, and consequently no product to use. (See Figure 5. 22.) Product quality stayed at one since no bugs were introduced, and no bugs problem existed. (See Figure 5.23.) No coaching took place, since there was no participation; and consequently the average developer talent did not change at all. (See Figure 5.24.)

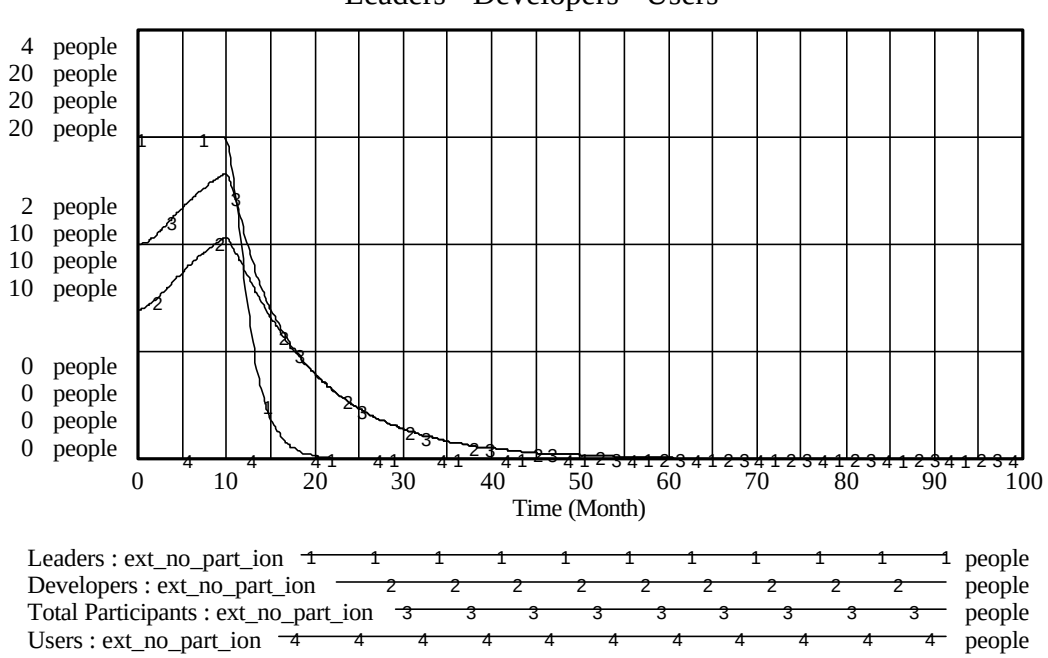


Figure 5.22. Leaders, Developers and Users under “No Participation” Extreme Case

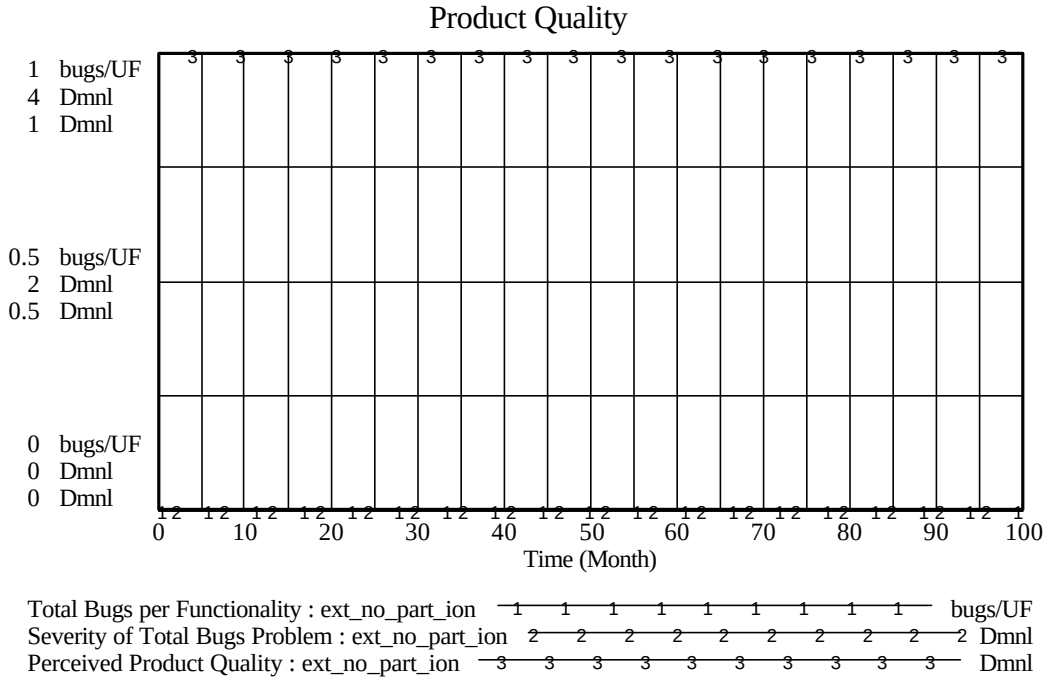


Figure 5.23. Product Quality under “No Participation” Extreme Case

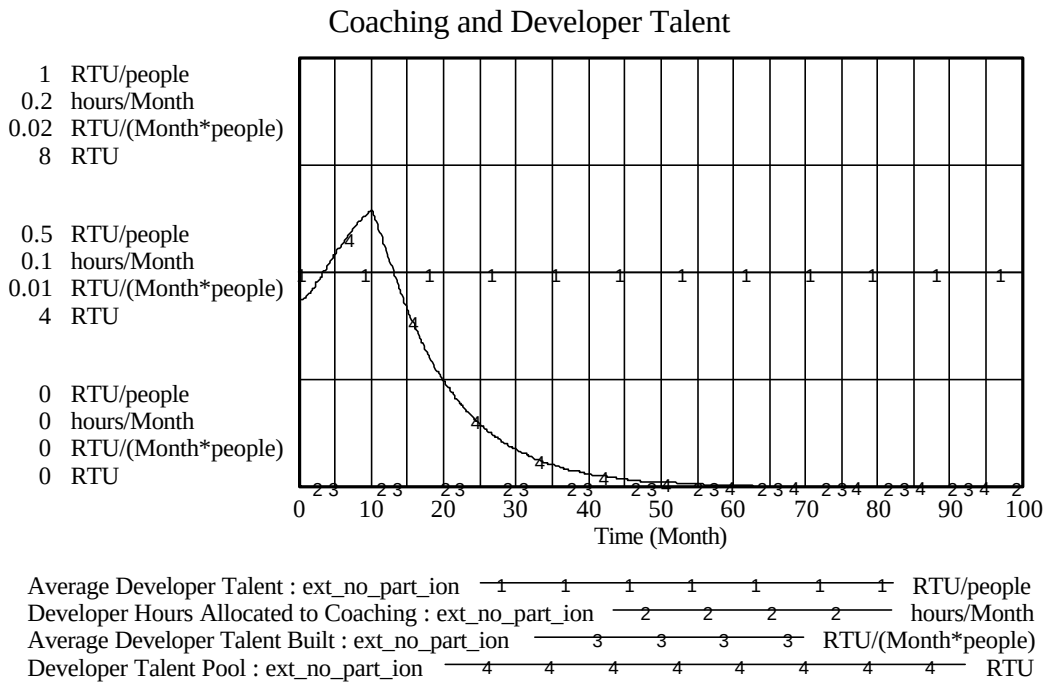


Figure 5.24. Coaching and Developer Talent under "No Participation" Extreme Case

5.3.6. Extremely High Participation

The opposite of the " no participation" case, "extremely high participation" was also applied to the model. In this run, both developer participation and leader participation were set to 10 times their normal level of 30 hours per month per person. As expected, the product functionality increased rapidly and reached the saturation point within the first 5 months. (See Figure 5.25.) This rapid growth in product functionality caused a fast decrease in opportunities for contribution, and thus the developers started to leave the community very early. The number of users increased rapidly, also due to the fast growth in product functionality. (See Figure 5.26.)

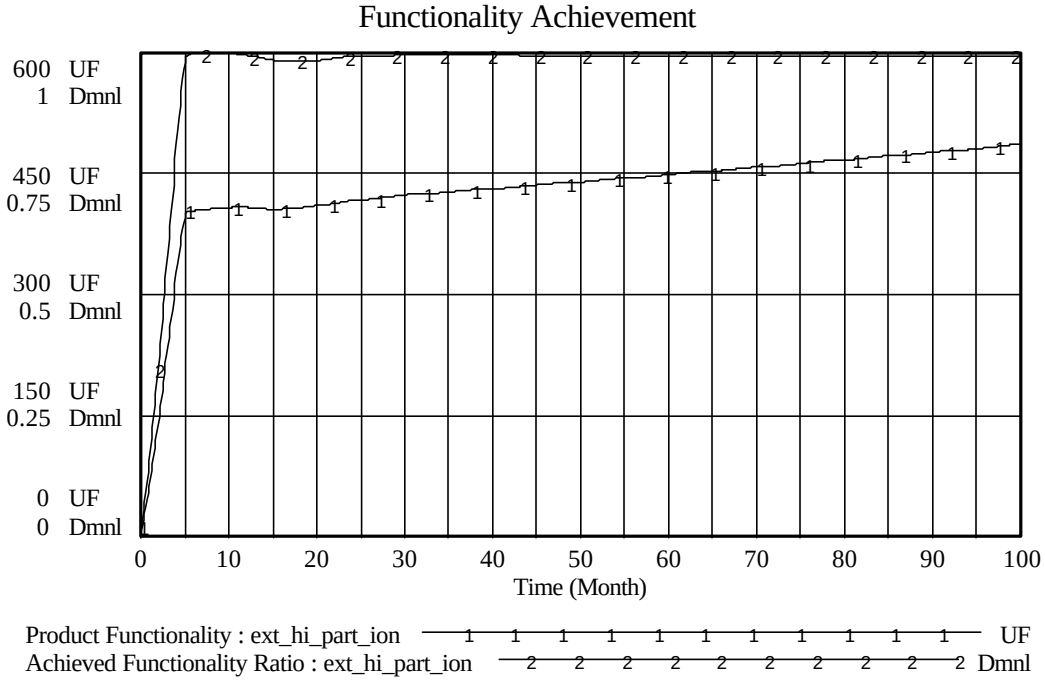


Figure 5.25. Functionality Achievement under "Extremely High Participation" Case

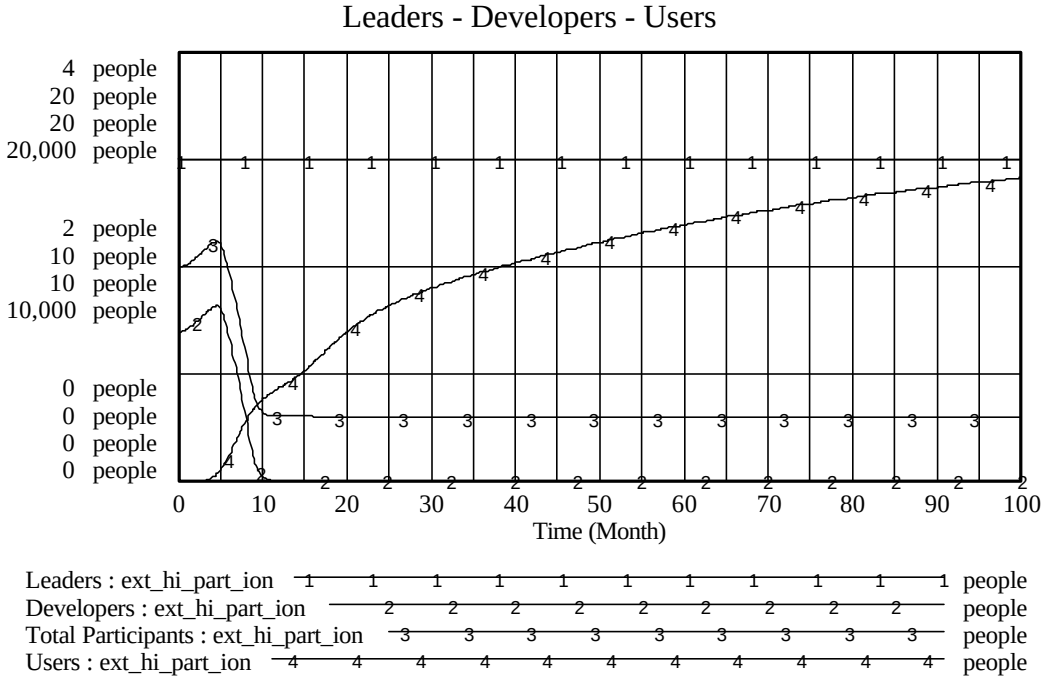


Figure 5.26. Leaders, Developers and Users under "Extremely High Participation" Case

The rapid growth in product functionality generated an equally rapid increase in the number of bugs per functionality. Due to the delay between the assessment of the bugs problem and reallocation of hours for debugging, and another delay between the detection and fixing of the bugs, the bugs problem increased considerably at the beginning of the run before it was under control. That caused the product quality to drop to a very low level during the first 10 months of the project. Eventually the product quality increased to an acceptable level. However, it stayed at an equilibrium that was lower than that in the base case. (See Figure 5.27.)

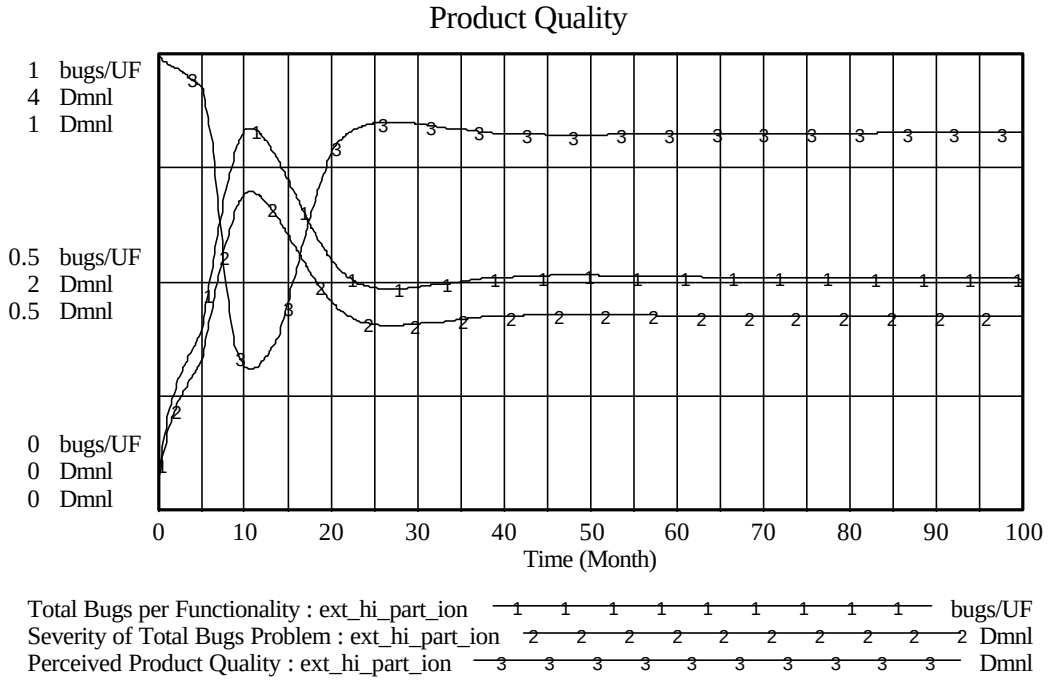


Figure 5.27. Product Quality under “Extremely High Participation” Extreme Case

The overall developer talent pool grew rapidly during the first five month of this run; however it decreased equally rapidly as the developers left the community. Average developer talent increased for the first 10 months as more developer talent was built

through coaching, but as experienced developers left the community it dropped back to its normal level. (See Figure 5.28.)

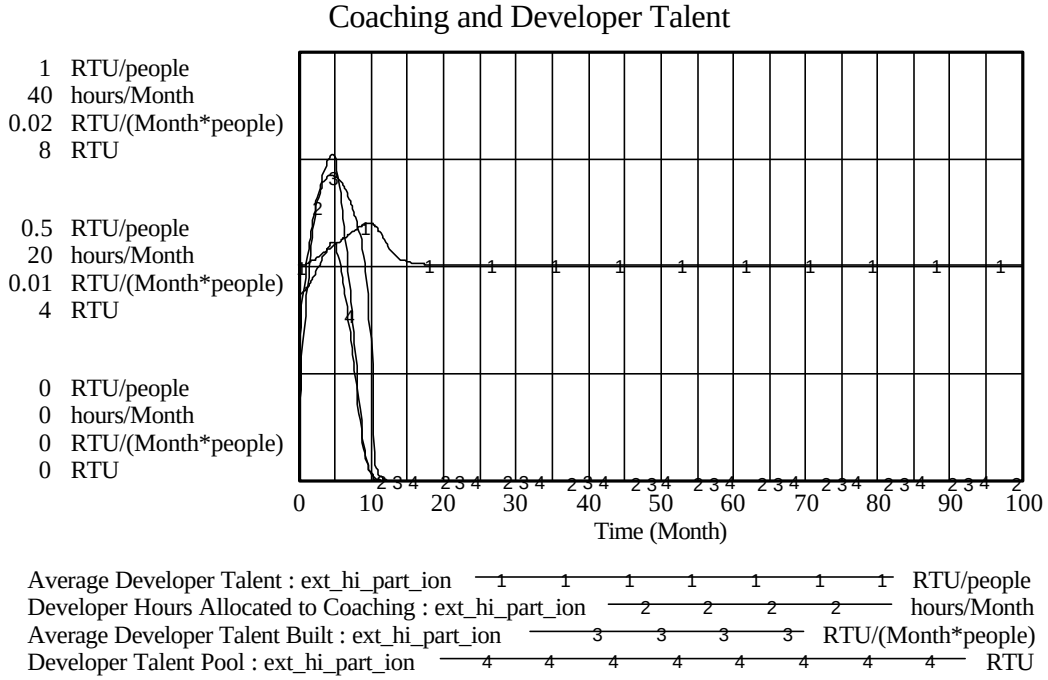


Figure 5.28. Coaching and Developer Talent under "Extremely High Participation" Case

5.3.7. Zero Productivity

An extreme case somewhat similar to "no participation" was "zero productivity." Here, both leaders and developers participate, but their productivity is zero. The results of this run showed similarities with the results of the "no participation" run. There was no increase in product functionality, since the participants could not produce. (See Figure 5.29.) Leaders and developers left the community very early on, and there were no users throughout the run. (See Figure 5.30.)

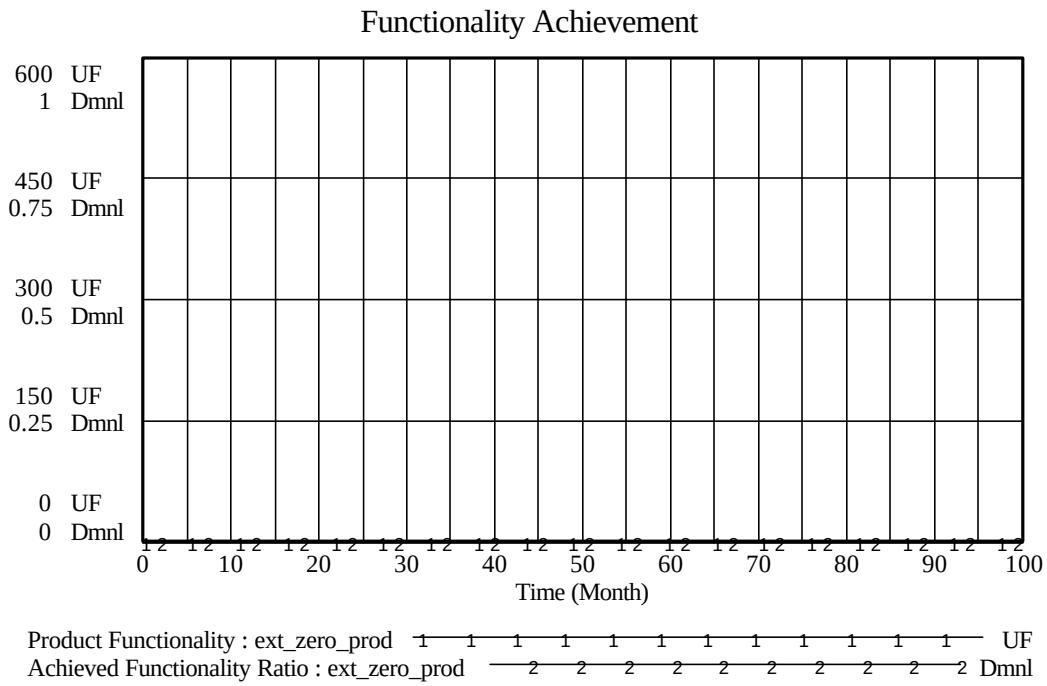


Figure 5.29. Functionality Achievement under “Zero Productivity” Extreme Case

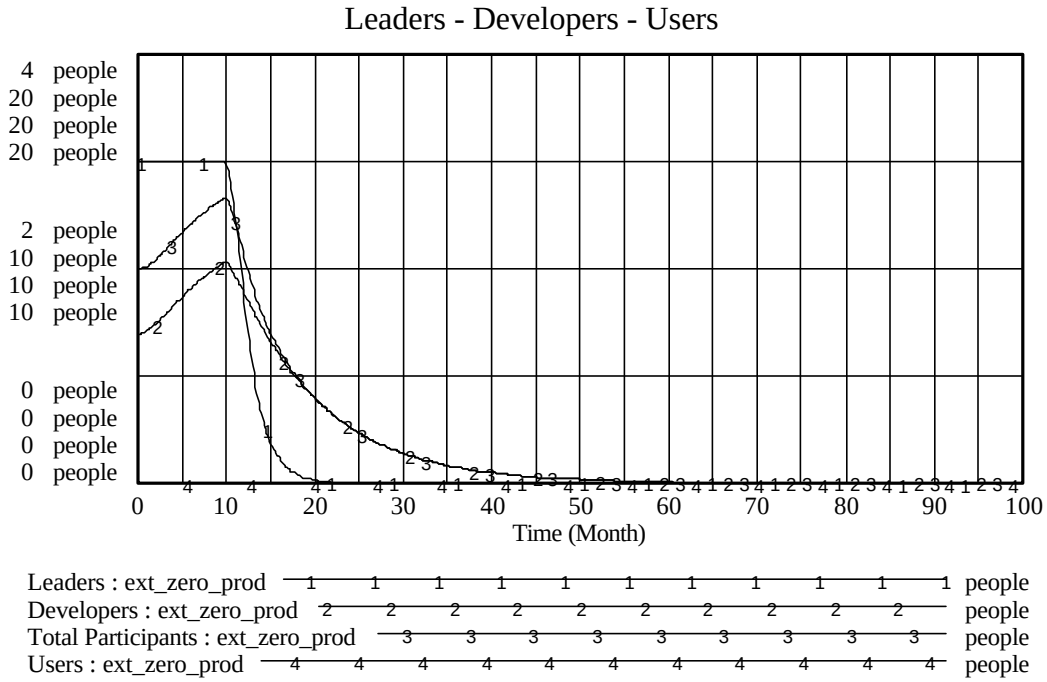


Figure 5.30. Leaders, Developers and Users under “Zero Productivity” Extreme Case

Product quality stayed at one, again, since there was no production to introduce any bugs. (See Figure 5.31.)

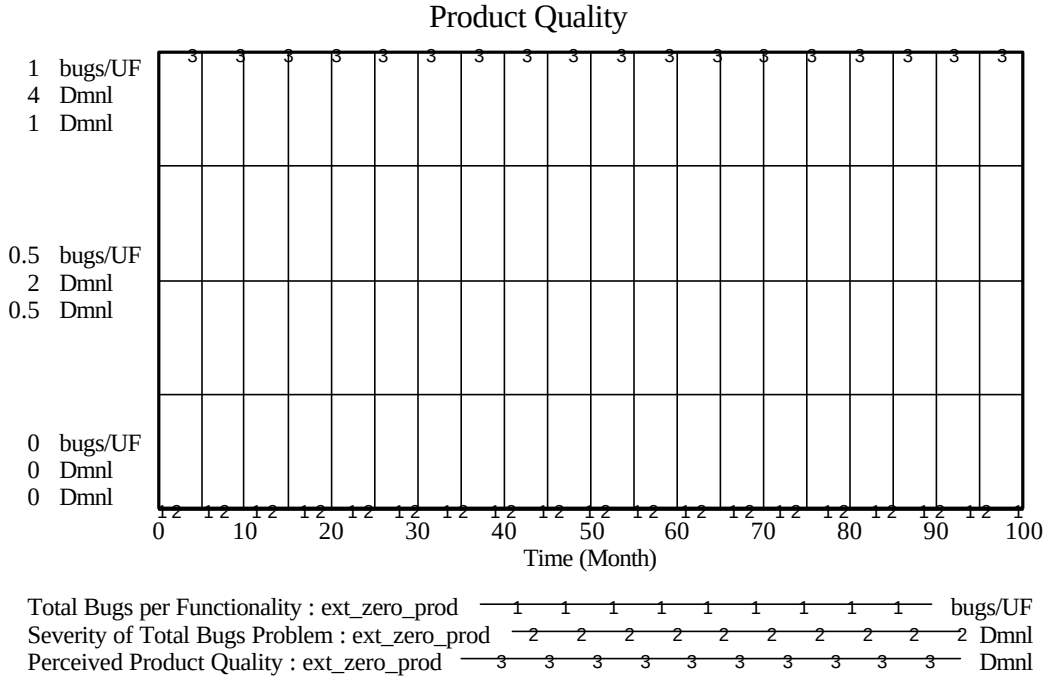


Figure 5.31. Product Quality under “Zero Productivity” Extreme Case

One notable difference from the “no participation” run was the existence of coaching in the “zero production” run, since developers and leaders participated in coaching as well as other activities in this run. Consequently, the average developer talent increased slightly while the participants stayed in the community. That growth stopped, however, as both the leaders and developers started to leave the community. (See Figure 5.32.)

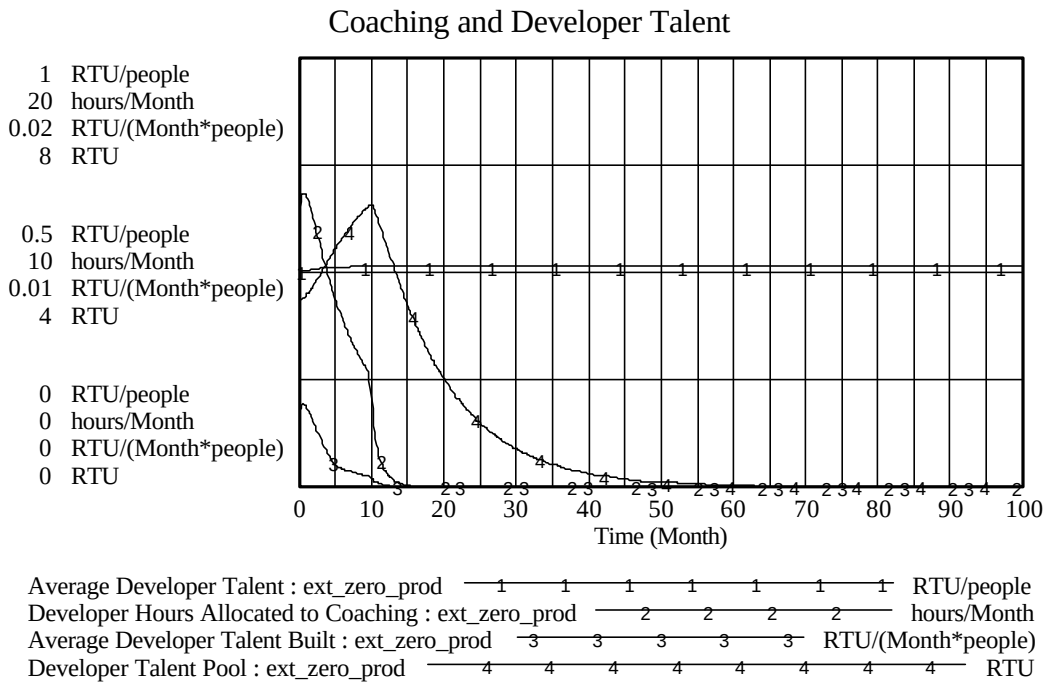


Figure 5.32. Coaching and Developer Talent under "Zero Productivity" Extreme Case

5.3.8. Extremely High Productivity

"Extremely high productivity" represents the opposite of the "zero productivity" case. In this run both the leaders' and developers' productivity levels were set to 10 times their normal values of 10 lines/hour and 5 lines/hour respectively. The results were very similar to those in the "extremely high participation" case. (Compare Figures 5.25 - 5.28 and Figures 5.33 - 5.36.) In this case, the rapid growth was driven by the extremely high productivity yield per hour of participation, as opposed to the extremely high level of participation as the driving factor in the earlier case. One notable difference was the behavior of the average developer talent. Since the level of participation was not as high in this case as in the "extremely high participation" case, there was not as much coaching, and consequently average developer talent did not increase as much as in the earlier case.

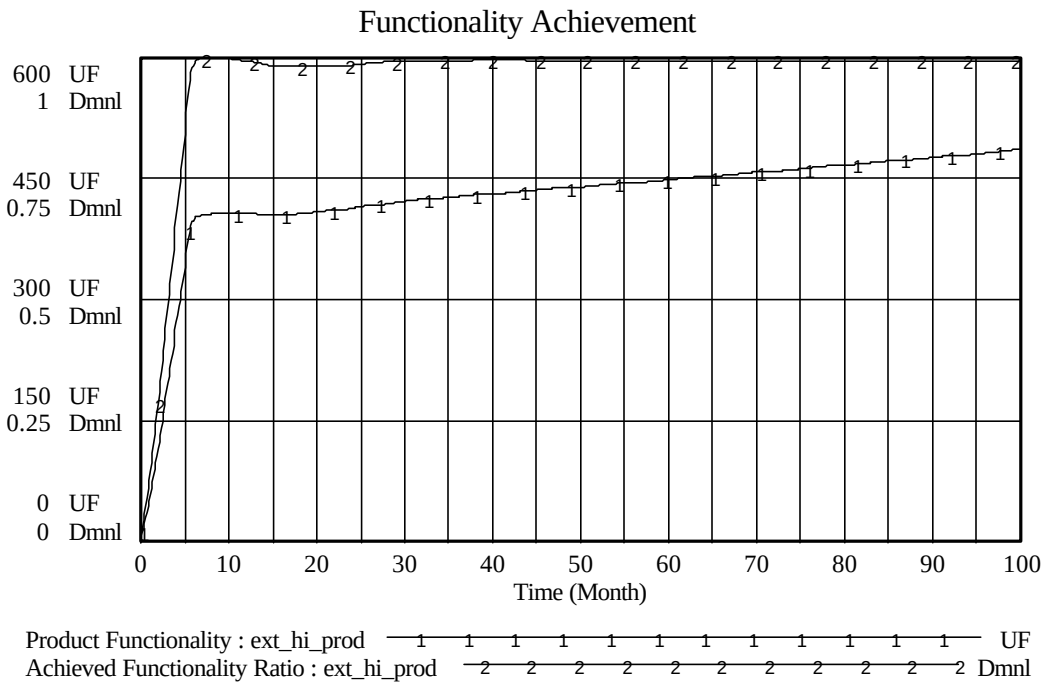


Figure 5.33. Functionality Achievement under "Extremely High Productivity" Case

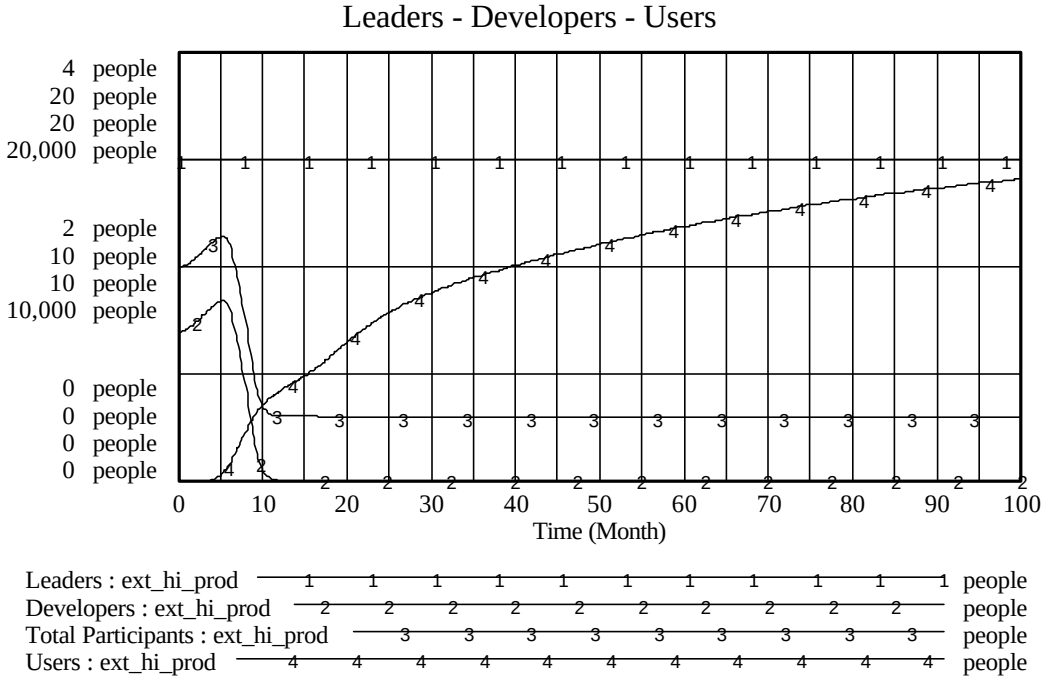


Figure 5.34. Leaders, Developers and Users under "Extremely High Productivity" Case

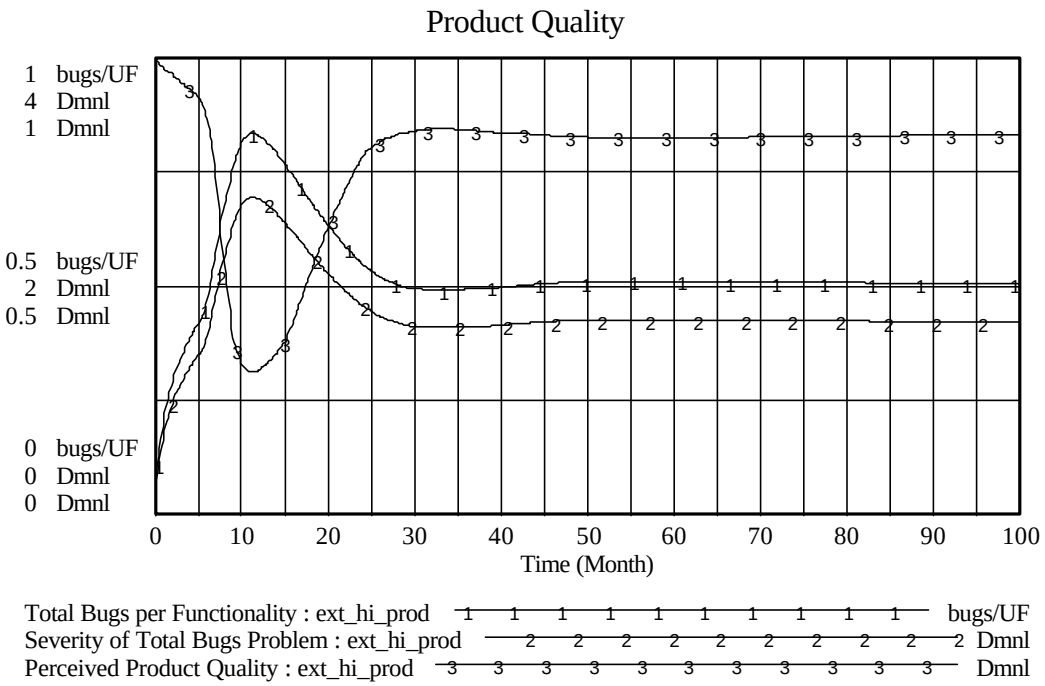


Figure 5.35. Product Quality under "Extremely High Productivity" Case

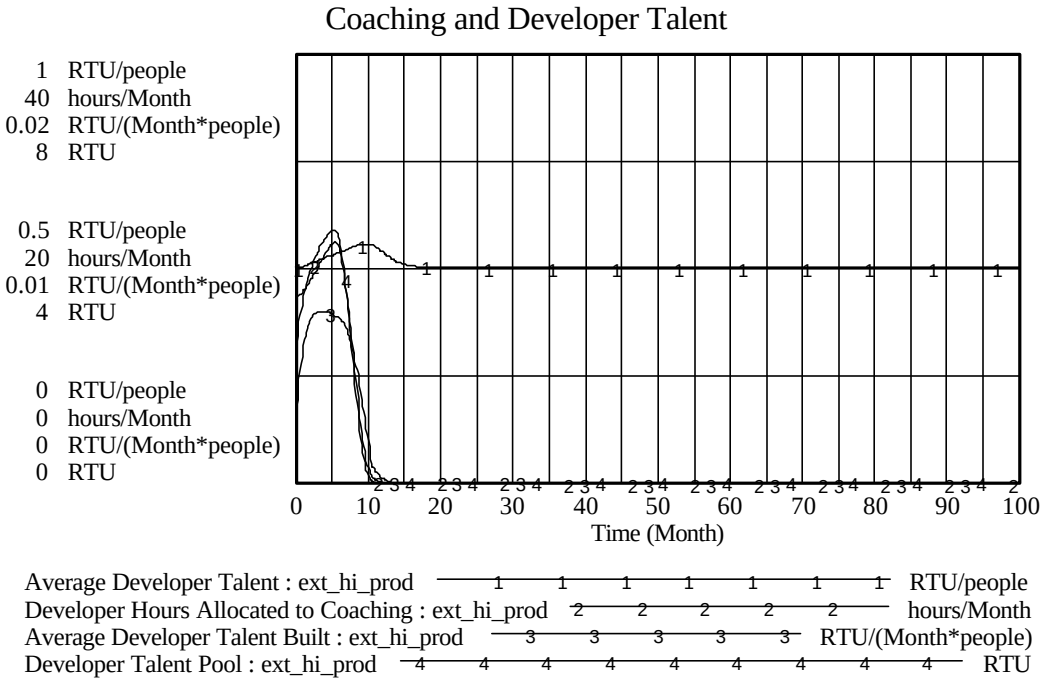


Figure 5.36. Coaching and Developer Talent under "Extremely High Productivity" Case

5.3.9. Zero Bug Generation

In another extreme condition run, the generating rate was set to zero, which meant that leaders and developers did not introduce any bugs while producing functionality. The behavior of the model under this condition was very close to its behavior under the base case condition with respect to functionality achievement and leader, developer and user populations. (Compare Figures 5.1 - 5.2 and Figures 5.37 - 5.38.) The expected behavior under this condition would be a faster growth in product functionality and the user population. This expectation was based on the rationale that no bug generation would save the participants considerable debugging time, which could be channeled to faster production. Figure 5.39 and 5.40 show that achieved functionality ratio and the number of users exhibited essentially the same behaviors under the base and the "zero bug generation" cases. One possible explanation for the small increase in the speed of functionality growth is that the participants worked under a considerably high pressure for production even in the base case, and the lack of debugging duties did not prompt them to achieve an ever faster production schedule in the "zero bug generation." This finding caused some doubt about the confidence in the model, and was noted as a potential analysis point for possible future extensions of this study.

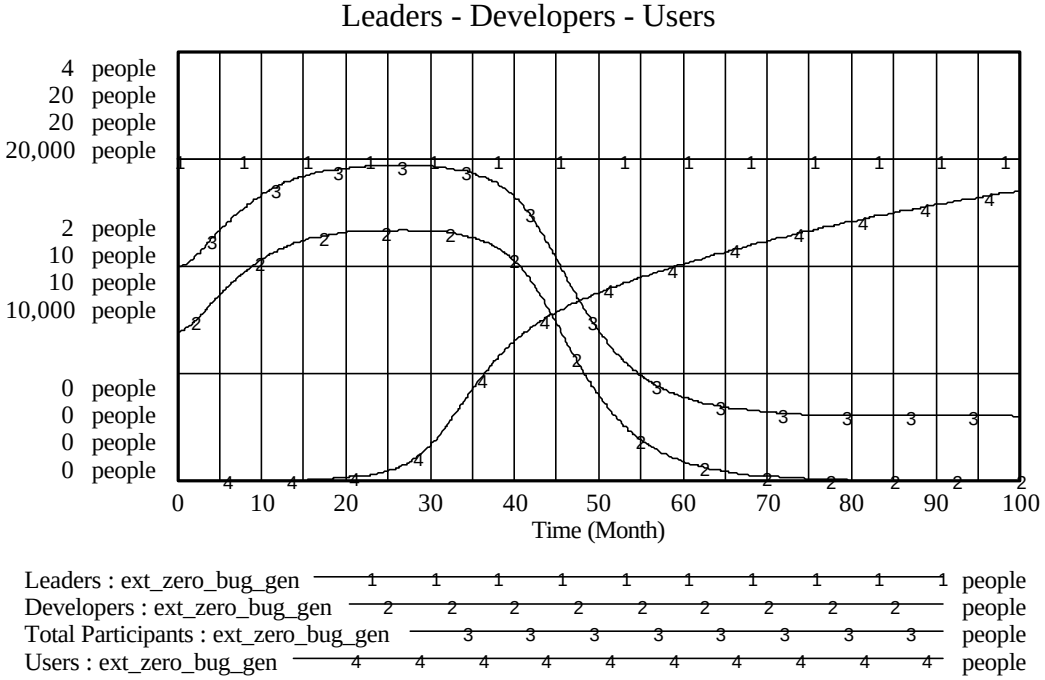


Figure 5.37. Leaders, Developers and Users under “Zero Bug Generation” Case

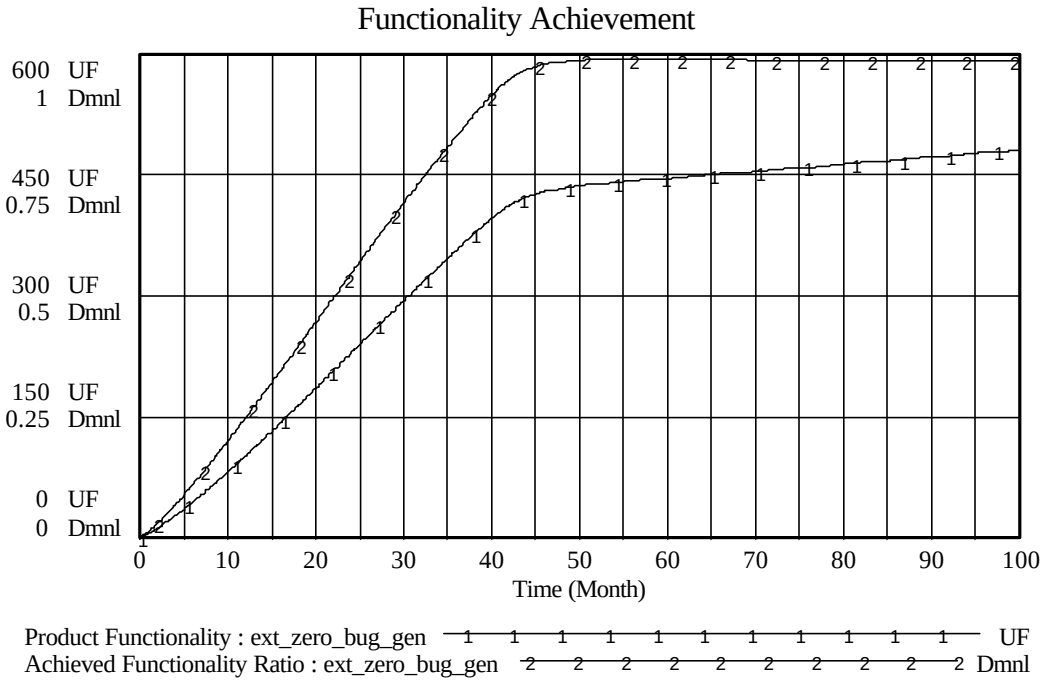


Figure 5.38. Functionality Achievement under “Zero Bug Generation” Case

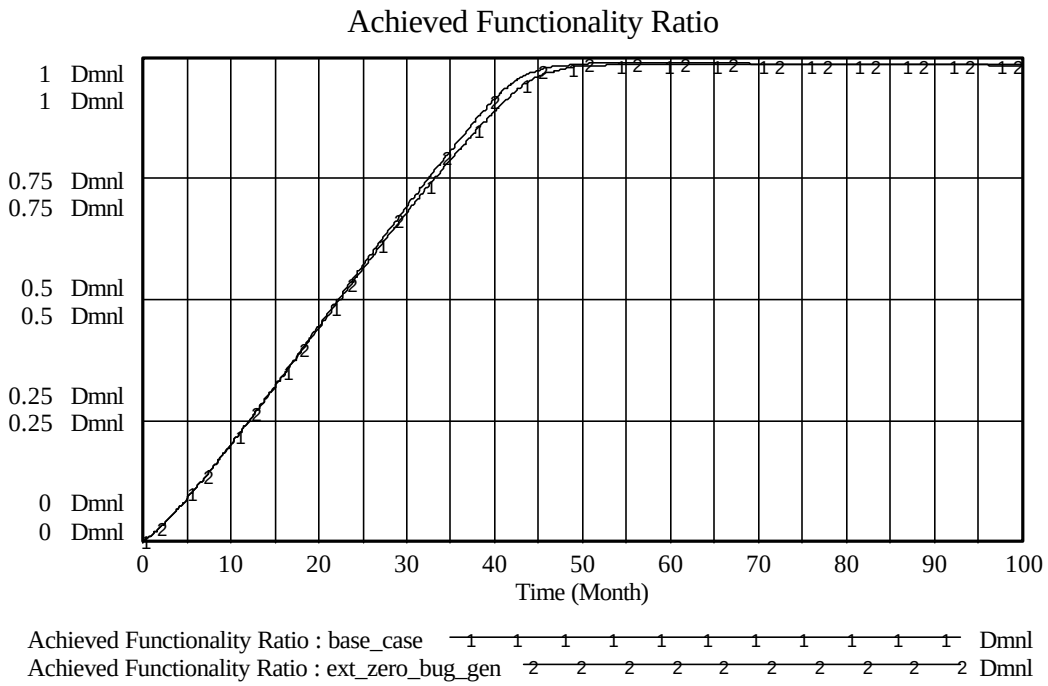


Figure 5.39. Achieved Functionality Ratio under Base Case and "Zero Bug Generation" Case

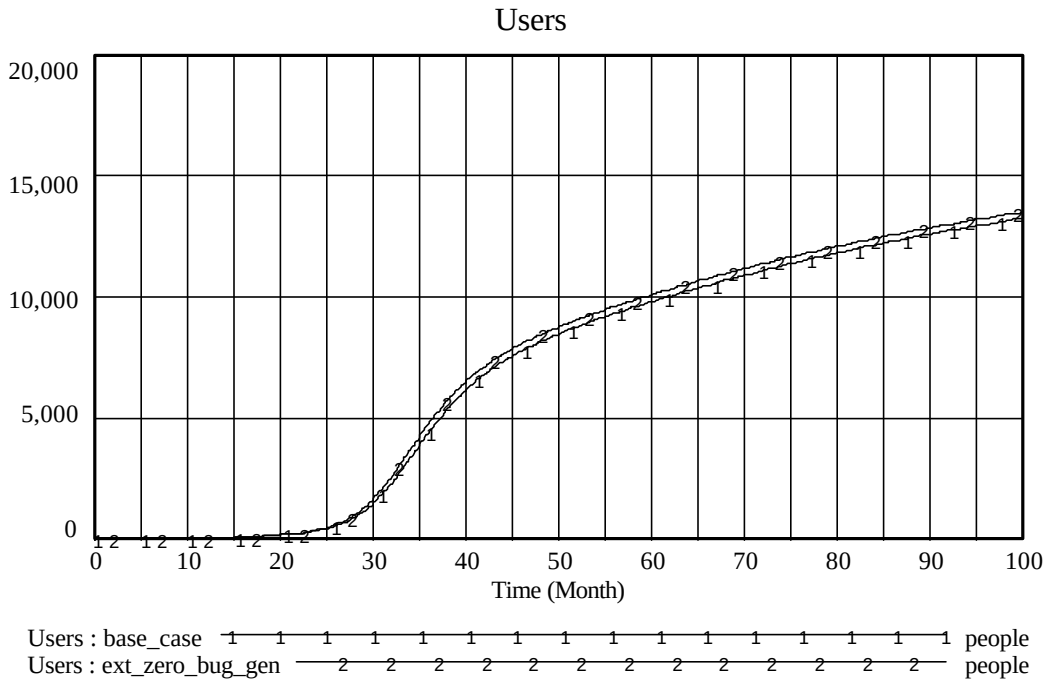


Figure 5.40. Users under Base Case and "Zero Bug Generation" Case

Some of the behaviors yielded by this run were within the expected ranges. For example, product quality stayed at one, since no bugs were introduce to the product. (See Figure 3.41.)

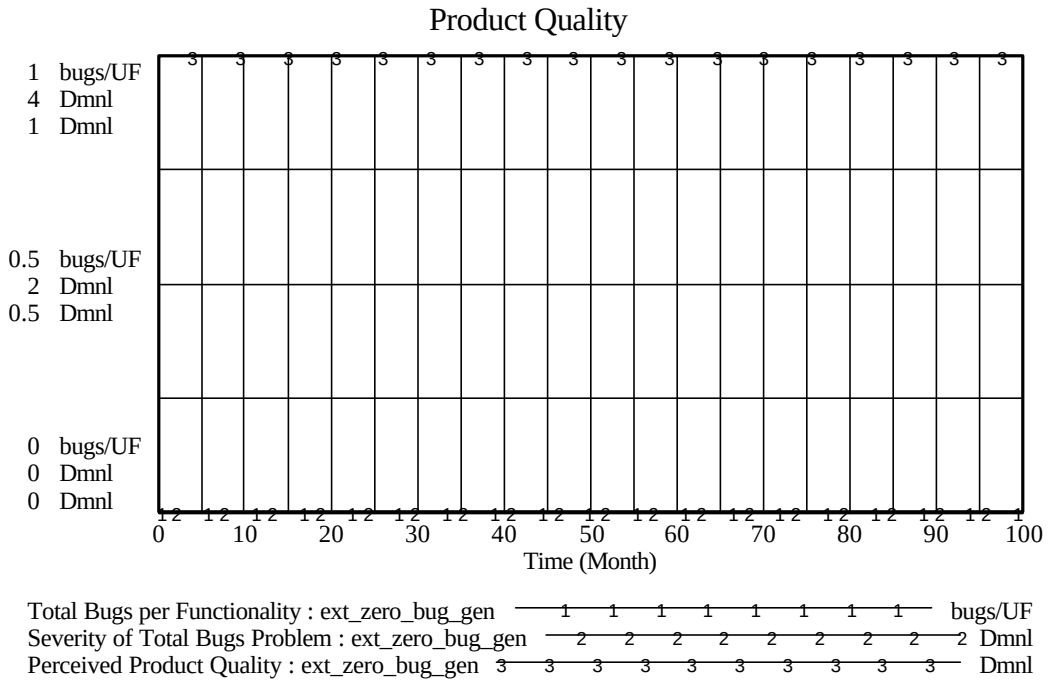


Figure 5.41. Product Quality under “Zero Bug Generation” Case

Another expected behavior was the increase in the average developer talent level. (See Figure 3.42) A portion of the time saved from debugging was channeled to more coaching, and that yielded a higher increase in the long run than that under the base case conditions. (See Figure 5.43.)

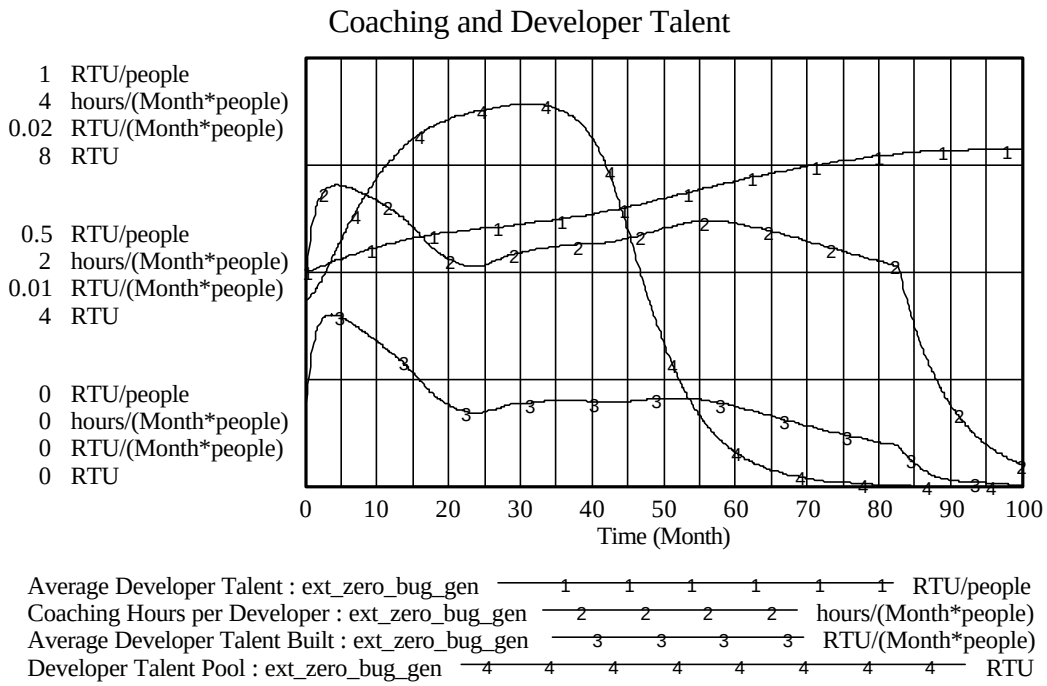


Figure 5.42. Coaching and Developer Talent under "Zero Bug Generation" Case

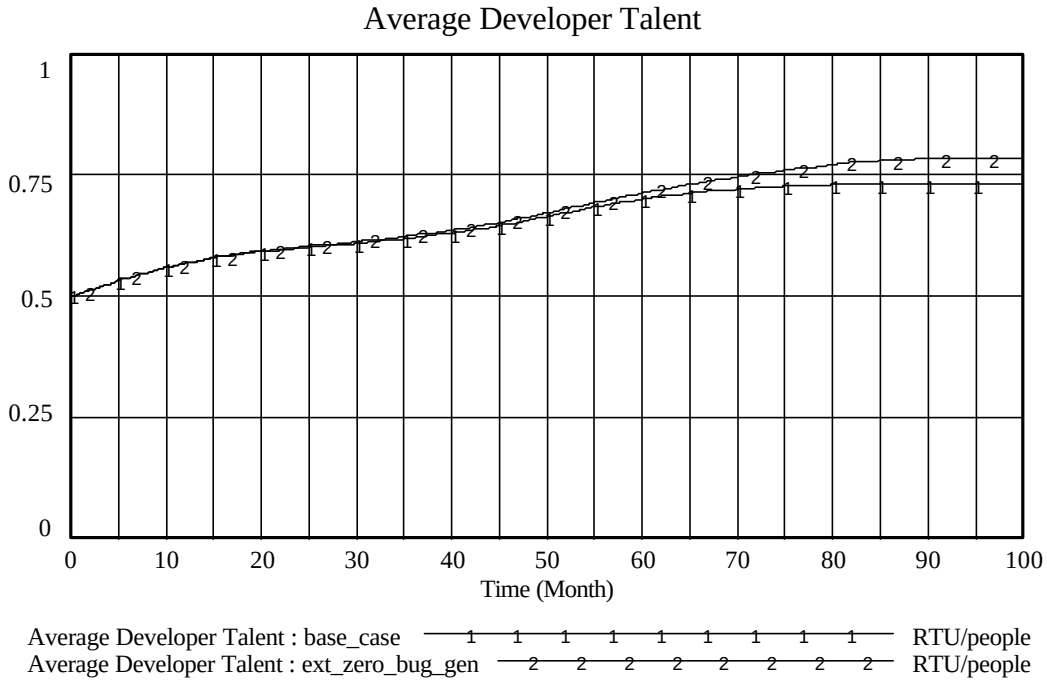


Figure 5.43. Average Developer Talent under Base Case and "Zero Bug Generation" Case

5.3.10. Extremely High Bug Generation

Another extreme condition run was done by setting the bug generating rate to 20 times its normal value of 0.01 bugs per line. As expected, the number of bugs per functionality turned out extremely high under this case, rendering an extremely low product quality. (See Figure 5.44.)

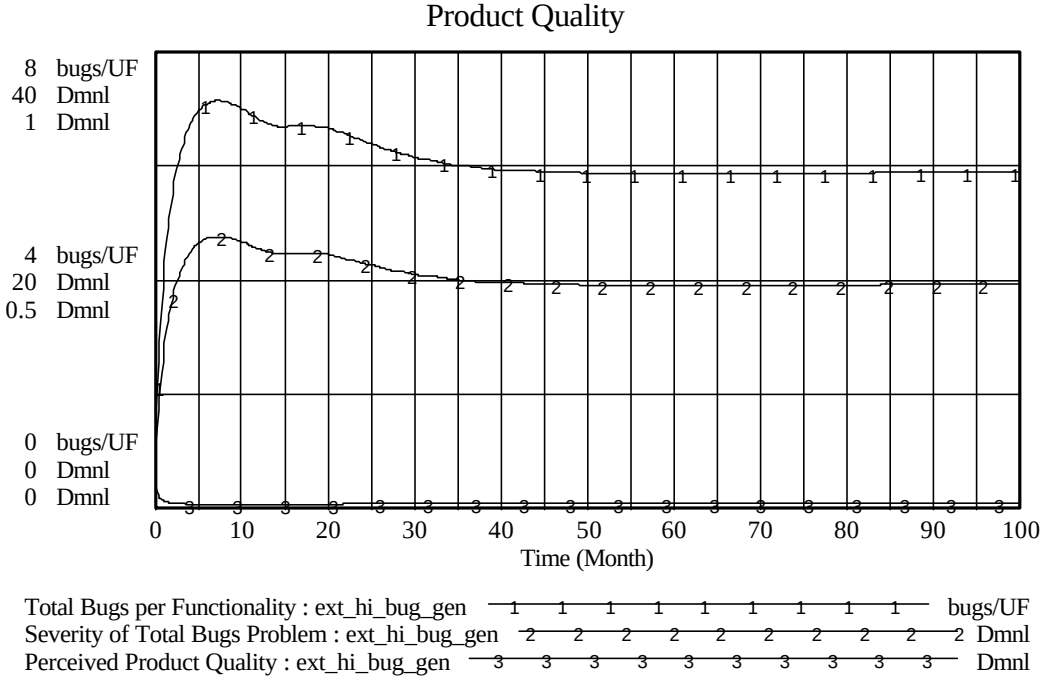


Figure 5.44. Product Quality under “Extremely High Bug Generation” Case

The extremely low level of product quality caused the number of developers to decrease right from the start of the run, and that decrease became sharper when the leaders started to leave the community for quality reasons as well. Also, the product could not attract a notable pool of users due to quality problems. (See Figure 5.45.) Functionality achievement stagnated due to the rapidly decreasing number of developers and leaders. (See Figure 5.46.)

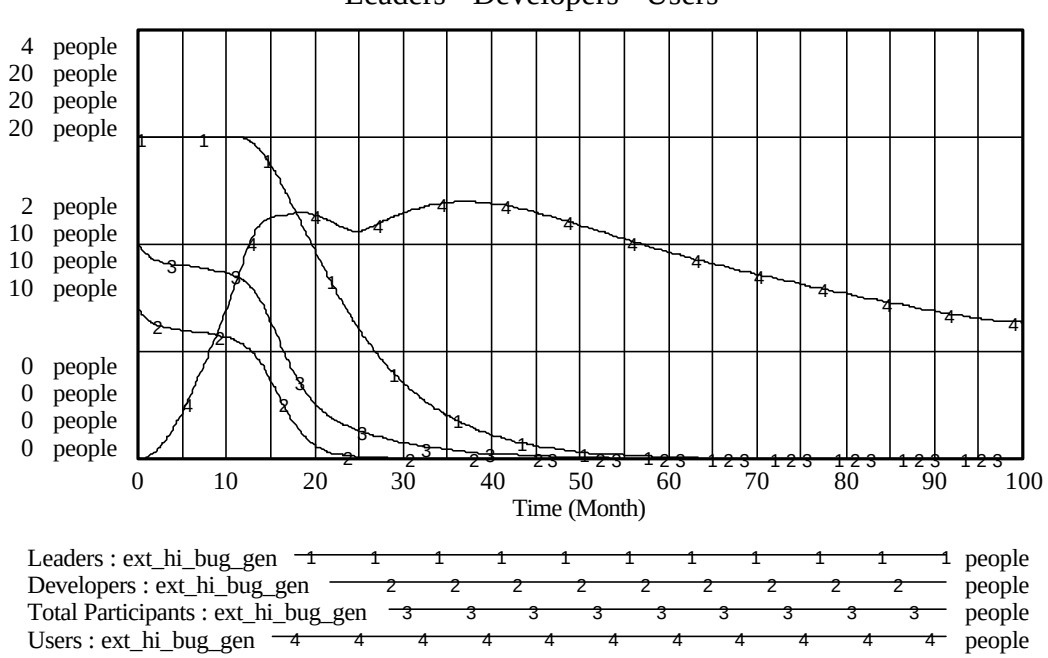


Figure 5.45. Leaders, Developers and Users under "Extremely High Bug Generation" Case

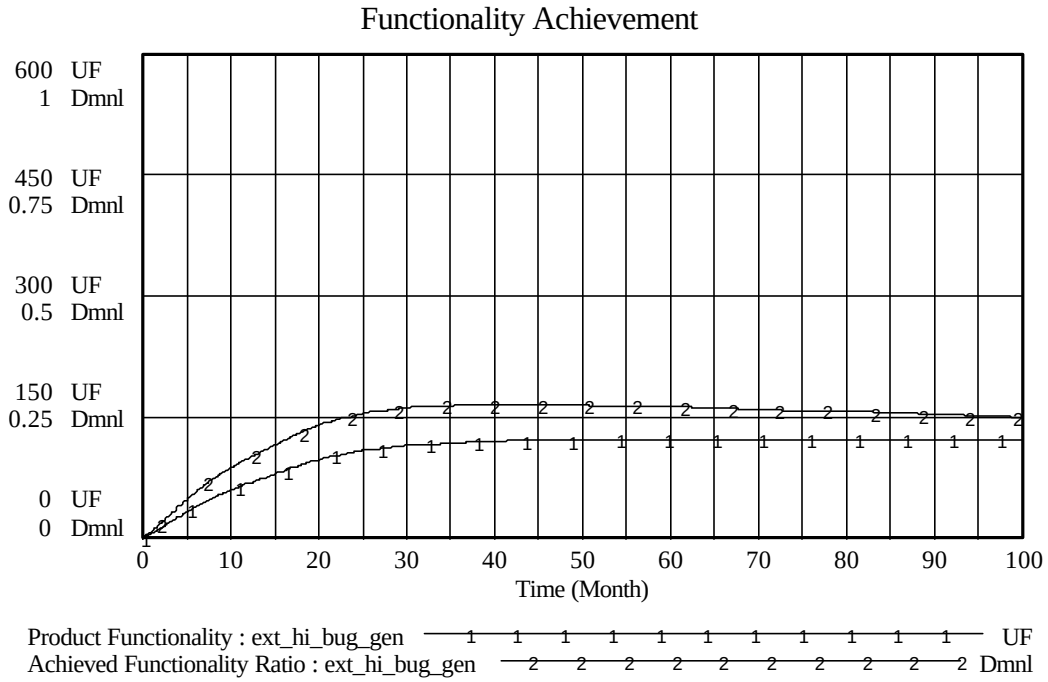


Figure 5.46. Functionality Achievement under "Extremely High Bug Generation" Case

Developer talent increased only until the leaders started to leave the community. After that point it started to decrease until it reached its original value of 0.5 relative talent units per person by month 40, since the talent built through the limited coaching efforts did not compensate for the decrease caused by the developer turnover. (See Figure 5.47.)

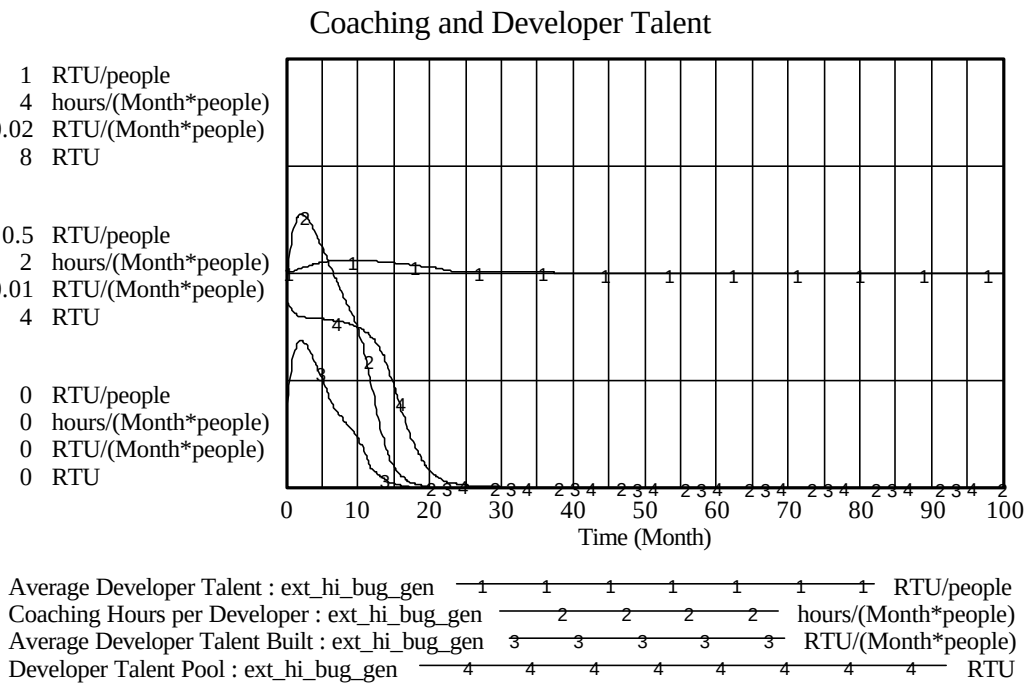


Figure 5.47. Coaching and Developer Talent under "Extremely High Bug Generation" Case

The extreme condition runs yielded mostly expected results, thus building a certain confidence for the model. Some results were outside the expected ranges; however the deviations were not so high as to refute the model altogether. The unexpected deviations can be used as analysis foci for possible future studies based on the model.

5.3.11. Implications of the Extreme Condition Runs

The extreme condition run performed on the OSSD model showed that the model exhibited expected behaviors under a substantial number of conditions that deviate extremely from normal conditions. Thus, the results of the extreme condition runs contributed to building confidence in the OSSD model. A limitation of extreme condition runs in general is that while they provide very useful information for confidence building, they do not provide much information for decision making based on the model. The reason for that is that decision making involves setting policy parameters to a choice of normal values under normal conditions, while extreme condition runs focus on abnormal conditions. Sensitivity runs, another type of model tests, provide important information for decision making as well as for building confidence in the model. The application of sensitivity runs to the OSSD model is discussed below.

5.4. Sensitivity Runs

Sensitivity runs are done in order to test whether the model exhibits the expected range of behavior under a range of parameter values. The model should not be abnormally sensitive to parameter changes. Substantial changes in model behavior for relatively small changes in parameter values would decrease the confidence in model. On the other hand, the model should exhibit the expected variety of behavior for relatively large changes in parameter values.

Sensitivity analysis has another important role beyond its function as a model-testing tool. It is possible to use sensitivity runs as preliminary analysis tools for policy analysis. Sensitivity runs done for policy variables, which can be controlled by decision and policy makers, may give initial hints about what policy variables yield the greatest

improvement, and what values of these variables yield results that are better than the base case.

Many sensitivity runs were performed on the OSSD model. Several sensitivity runs, which yielded the most critical findings, are discussed below. Among these are runs that served as preliminary policy runs, such as the runs for refusal rate and rejection rate. (See Section 5.4.7 and Section 5.4.8.)

5.4.1. Average Developer Participation

The model was run for different values of average developer participation. The runs yielded results that are within a reasonable range. The runs where average developer participation was set to 5, 10, 45 and 60 hours/(month*person) are discussed below, along with the base case, where average developer participation was 30 hours/(month*person.) Figure 5.48 displays the behaviors of product functionality for different values of average developer participation. As the average participation increased so did the speed of product functionality growth. In the runs where average participation was set to 10 and 5 hours/(month*person) the product functionality level did not reach the saturation point during the 100-month simulation horizon. (See Figure 5.48.) In fact, under a 5 hours/(month*people) average participation condition product functionality reached a low equilibrium of around 125 UF, which indicates that all the participants have left the community.

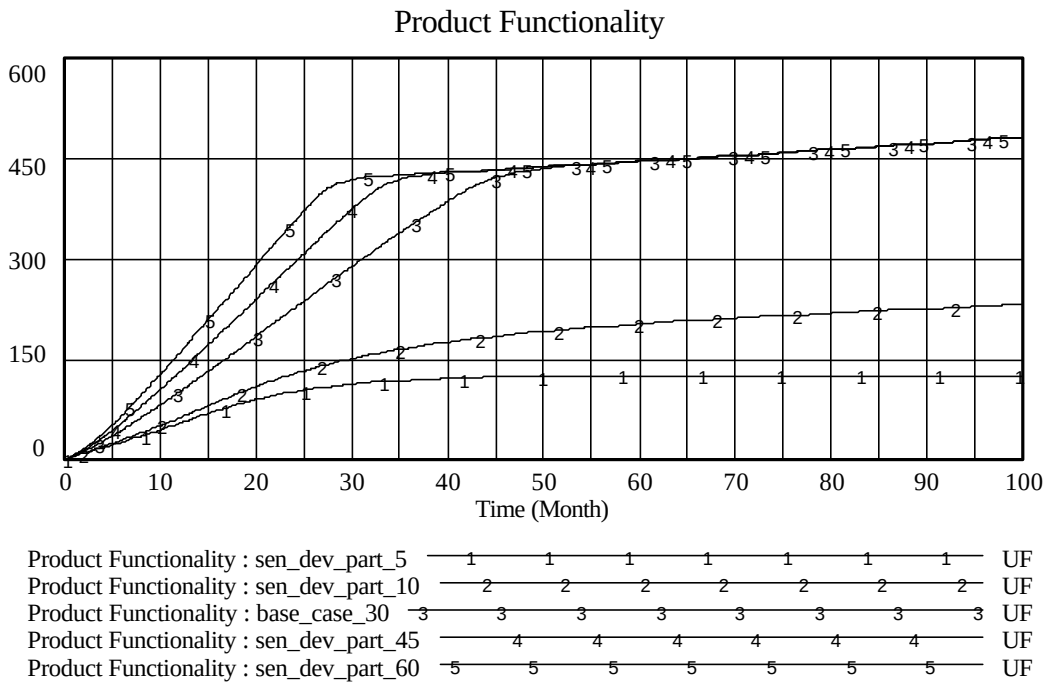


Figure 5.48. Product Functionality for Different Values of Average Developer Participation

The number of total participants showed different behaviors for different values of average developer participation as well. Figure 5.49 shows that as average participation increased, the change in the number of developers happened more quickly. As the speed of functionality growth increased, opportunities for contribution got scarcer faster. That caused the developers to leave the community earlier for higher values of average participation. (See Figure 5.49.) Figure 5.49 shows that all the participants left the community by month 75 for the run where average participation was set to 5 hours/(month*person.)

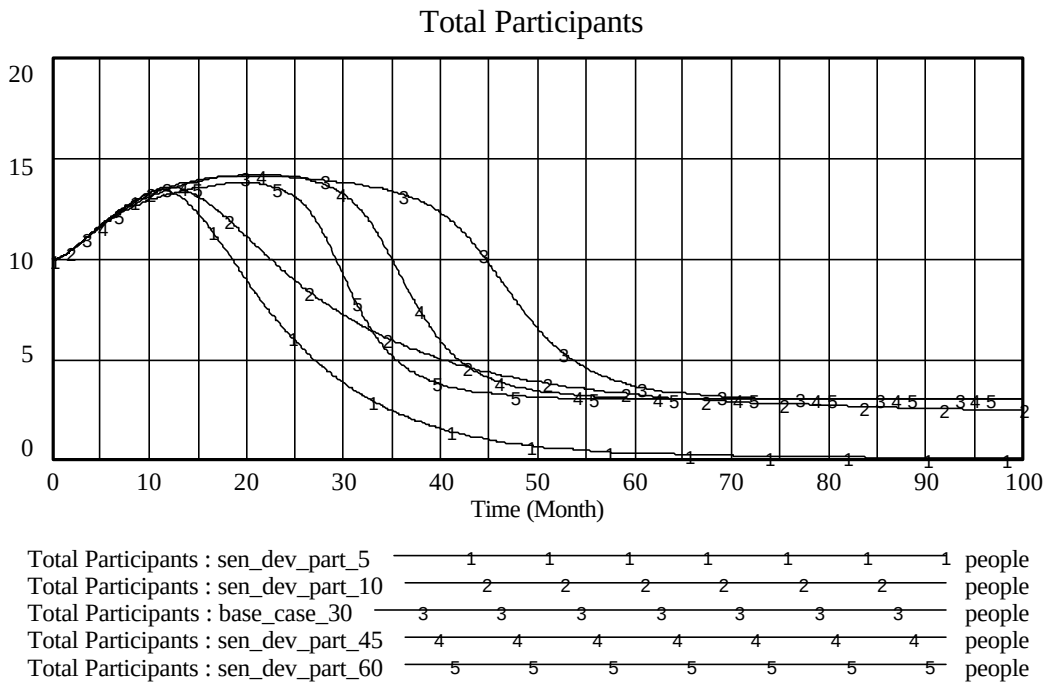


Figure 5.49. Total Participants for Different Values of Average Developer Participation

Figure 5.50 shows the behaviors of the number of users under different average participation values. Here again, as the average participation increased, the growth of the number of users became faster. Decreasing the average participation value impeded the growth of the user population.

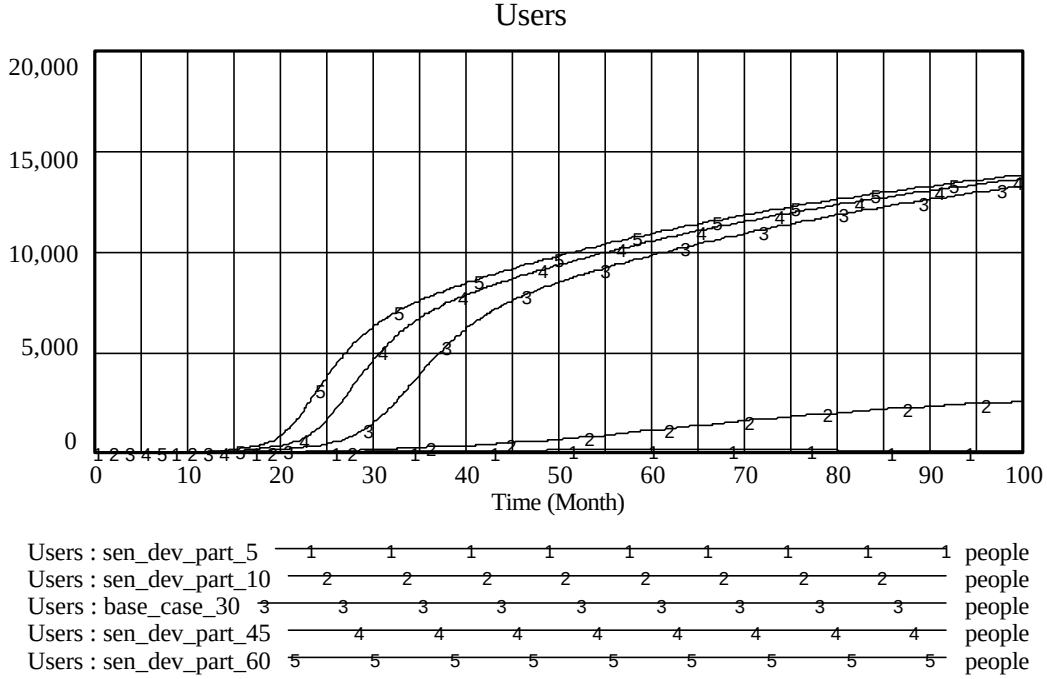


Figure 5.50. Users for Different Values of Average Developer Participation

The analysis indicated that there is a critical value of average developer participation for the given model, below which the community would not be able to sustain itself. Running the model for 200 months instead of the original 100 months revealed that the critical value lies between 10 and 11 hours/(month*person). Figure 5.51 shows that the number of participants decreased early on after a short period of increase for both cases where average participation was set to 10 and 11 hours/(month*person). However, in the case where average participation was 11 hours/(month*person), the number of participants started to increase once again, due to increasing interest among potential developers, since the product had reached a critical level of functionality, and continued with healthy growth. (See Figure 5.52.) Such an increase does not happen in the case where average participation was 10 hours/(month*person), indicating that the community would eventually cease to exist.

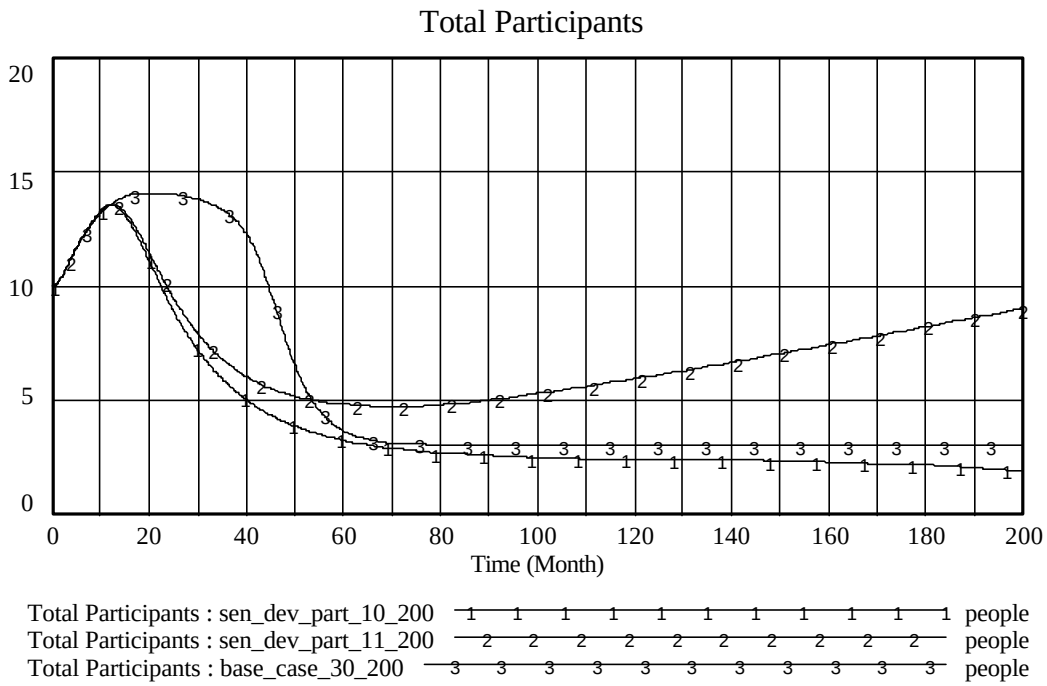


Figure 5.51. Total Participants for Different Values of Average Developer Participation

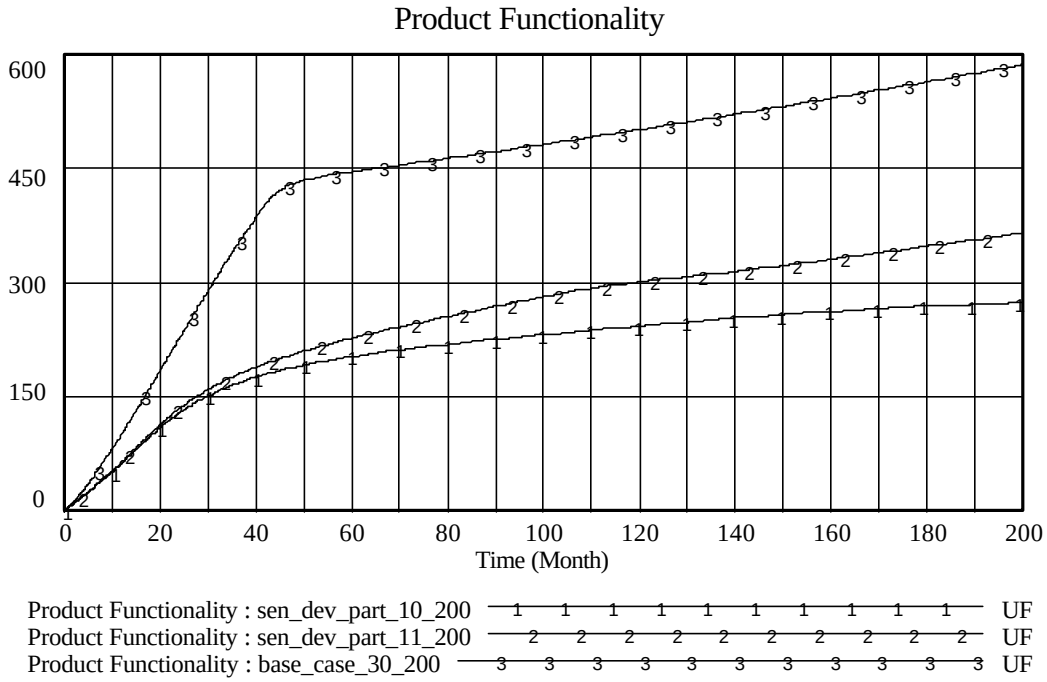


Figure 5.52. Product Functionality for Different Values of Average Developer Participation

Figure 5.53 shows that the number of users started to decrease after a certain point for 10 hours/(month*people) average developer participation, thus confirming that the community would dissolve under that condition. Meanwhile, the number of users continued to increase under the condition of 11 hours/(month*people).

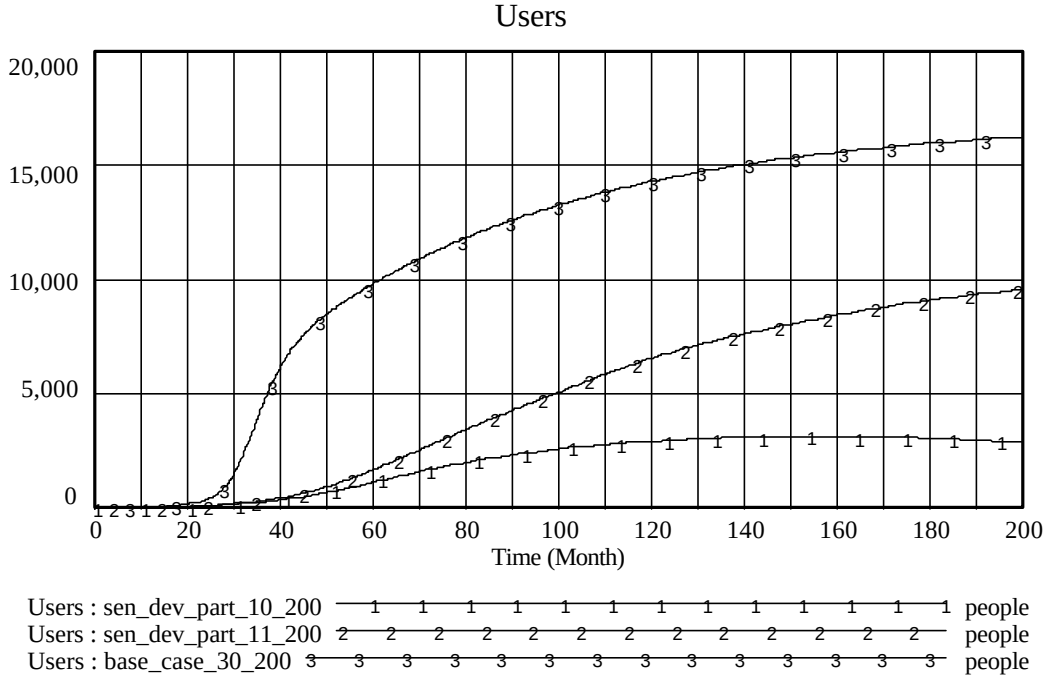


Figure 5.53. Users for Different Values of Average Developer Participation

Perceived product quality exhibited larger decreases at the beginning of the project for higher values of average developer participation. (See Figure 5.54.) This is attributable to the fact that the proportion of code produced by developers was higher for higher values of average developer participation, and developers introduced more bugs per functionality compared to leaders. However it can also be seen in Figure 5.54 that the perceived product quality improved faster as the average participation level increased, due to more developer hours available for debugging. Perceived product quality stayed considerably high for very low average participation levels, due to the limited amount of

code produced by developers. However, higher quality did not help the community in those cases, since the quality of a product that is not functional would be irrelevant for users.

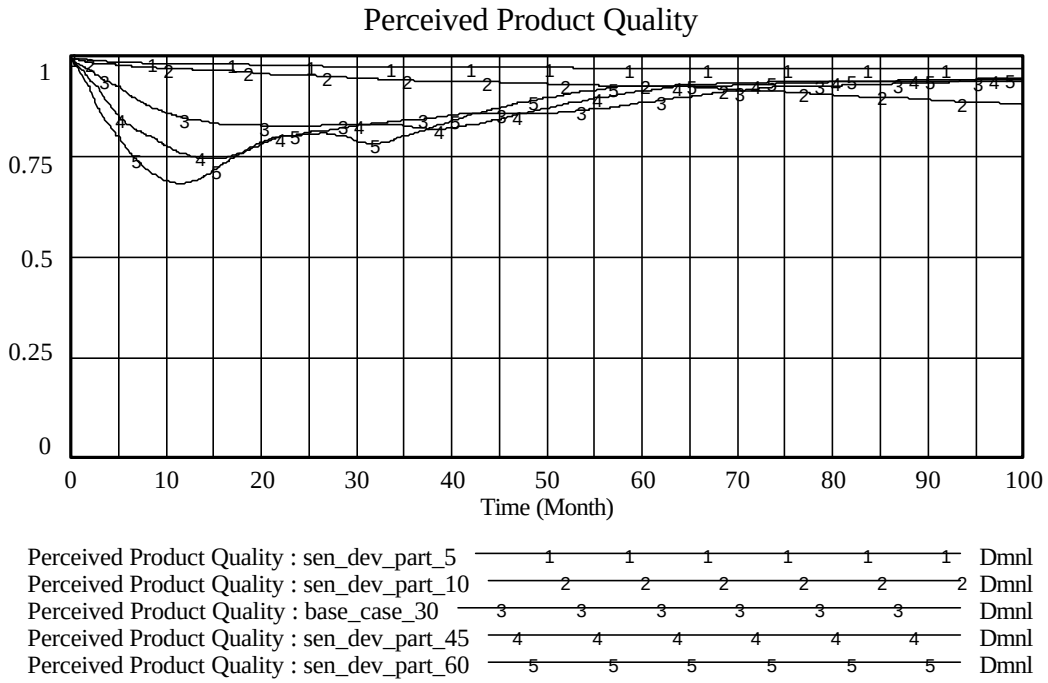


Figure 5.54. Product Quality for Different Values of Average Developer Participation

Developer talent increased faster for higher values of average participation, since more developer hours were available for coaching. However, the average talent reached lower equilibriums for higher values of average participation, since developers left the community earlier in those runs, and did not have the time to have more coaching. (See Figure 5.55.)

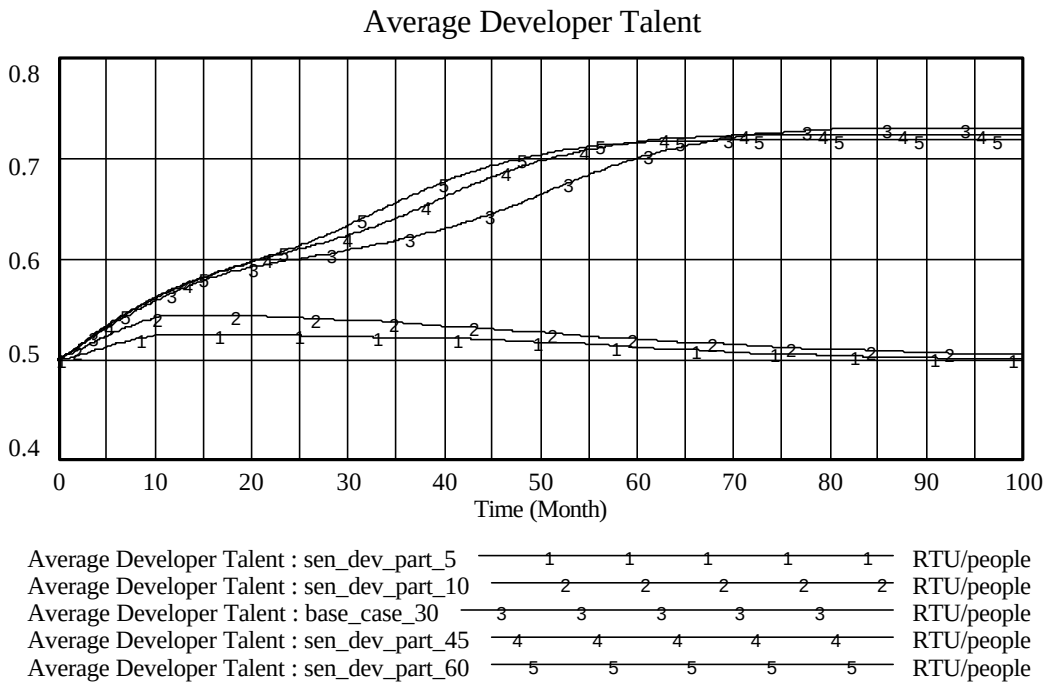


Figure 5.55. Average Developer Talent for Different Values of Average Developer Participation

5.4.2. Average Developer Productivity

Another set of sensitivity runs was done for different values of average developer productivity. Average developer productivity in the base case run was 5 lines/hour. The results of the runs where average developer productivity was set to 1, 2.5, 7.5 and 10 lines/hour are discussed below. As expected, higher average developer productivity yielded faster product functionality growth. (See Figure 5.56.)

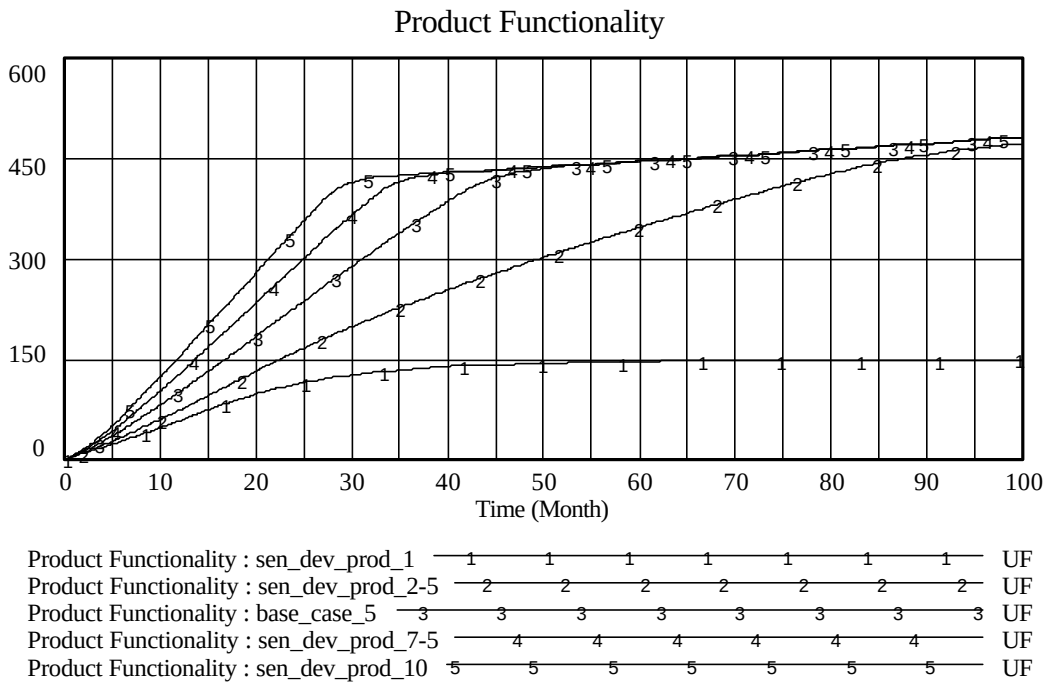


Figure 5.56. Product Functionality for Different Values of Average Developer Productivity

Higher values of average developer productivity also caused the number of total participants to exhibit its fundamental behavior pattern and reach equilibrium earlier. (See Figure 5.57.) Total participants decreased early on in the case where the average developer productivity was set to 1 lines/hour, since many developers left the community due to very low product functionality levels. The leaders followed the developers, thus bringing the total number of participants to zero by the end of the simulation horizon for that run. An interesting alternative behavior pattern was observed when average developer productivity was set to 2.5 lines/hour. In that run the number of total participants decreased early on as well, due to the low product functionality level. However, that decrease slowed down as the developer interest in the project was rekindled due to improving functionality achievement. Finally the decrease accelerated

again due to scarce contribution opportunities as the product functionality approached the saturation point. (See Figure 5.57.)

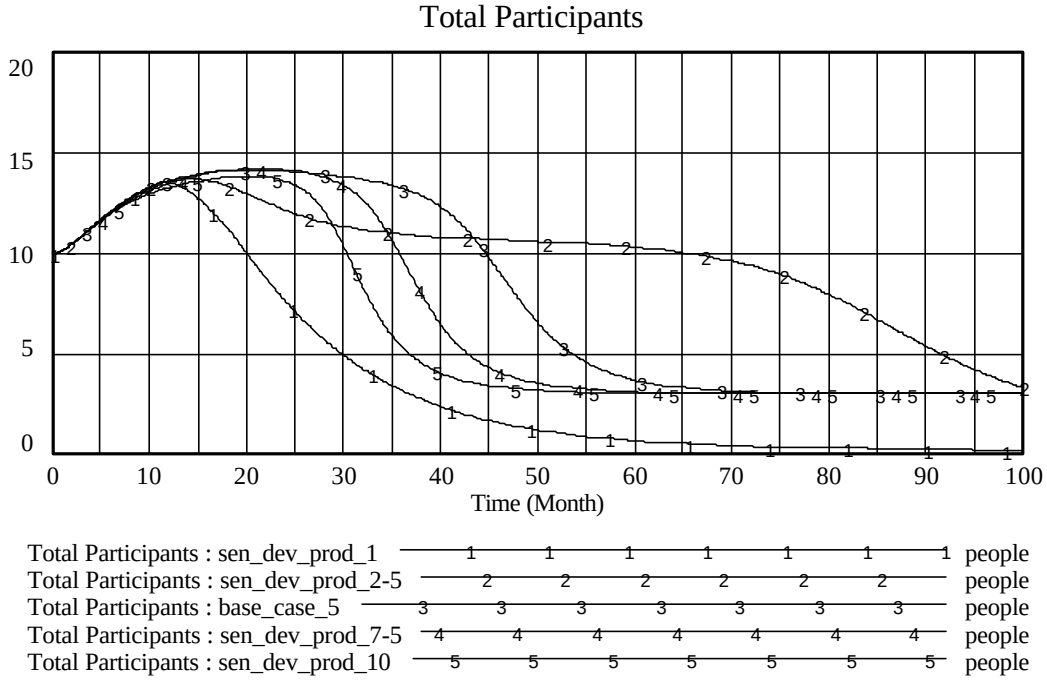


Figure 5.57. Total Participants for Different Values of Average Developer Productivity

Figure 5.58 shows how the behaviors of the number of users unfolded under different average productivity conditions. Basically, as the average productivity increased, the growth of the number of users became faster, as expected. The behavior of number of users for very low average productivity values, together with the behaviors of the number of total participants, indicated that there should be a critical value for average developer productivity below which the community would not be able to sustain itself. Further analysis revealed that the critical value lies between 1.6 and 1.7 lines/hour. (See Figures 5.59 - 5.61.)

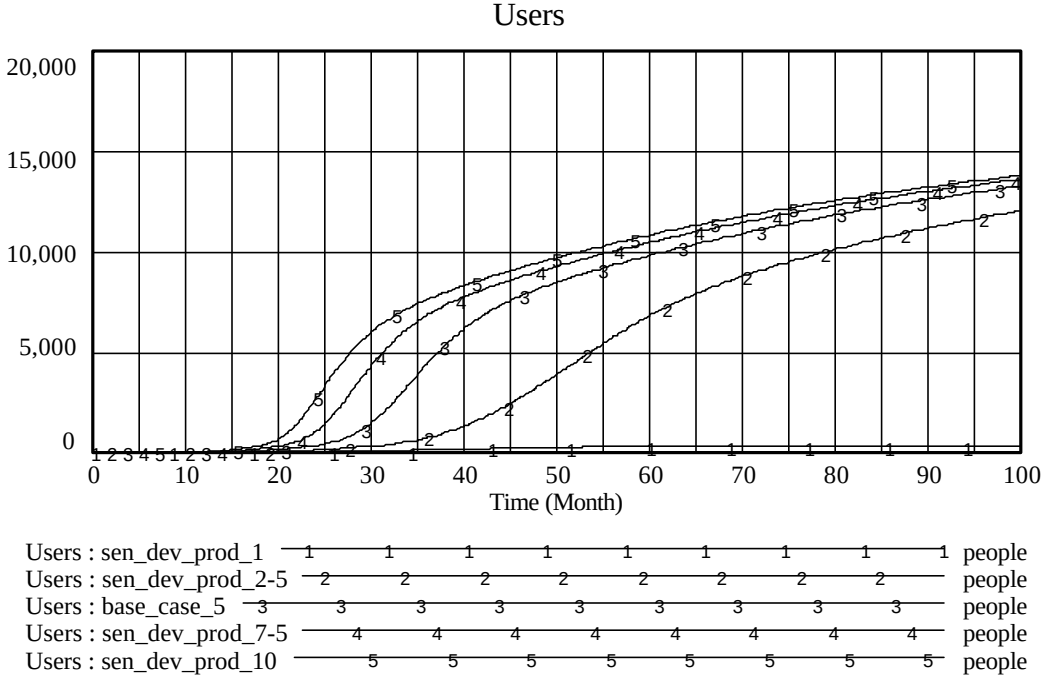


Figure 5.58. Users for Different Values of Average Developer Productivity

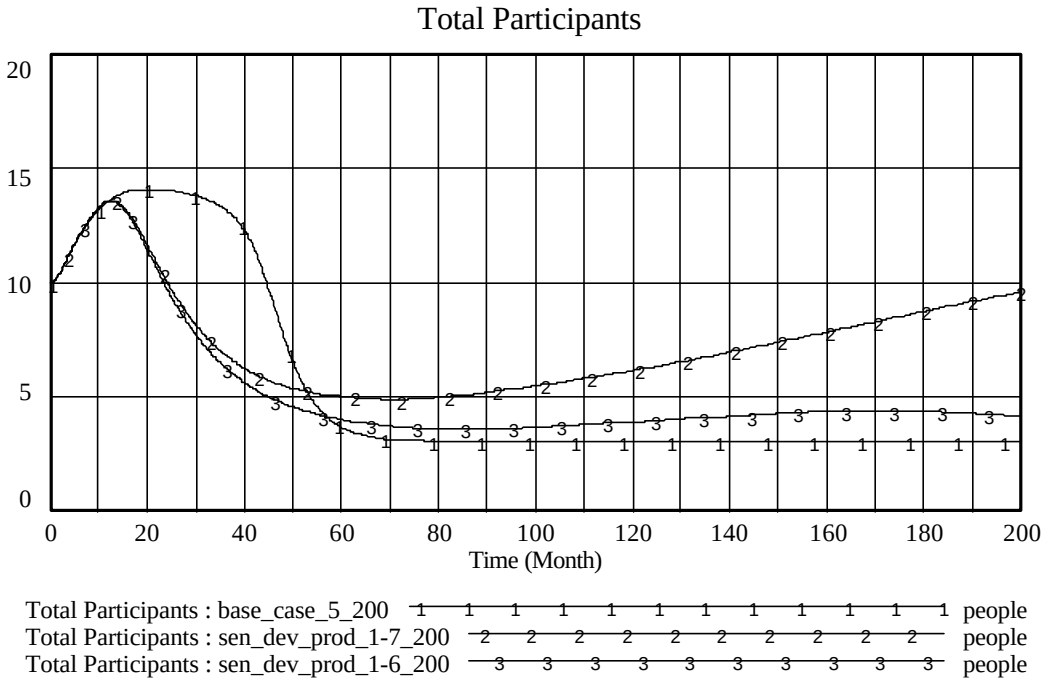


Figure 5.59. Total Participants for Different Values of Average Developer Productivity

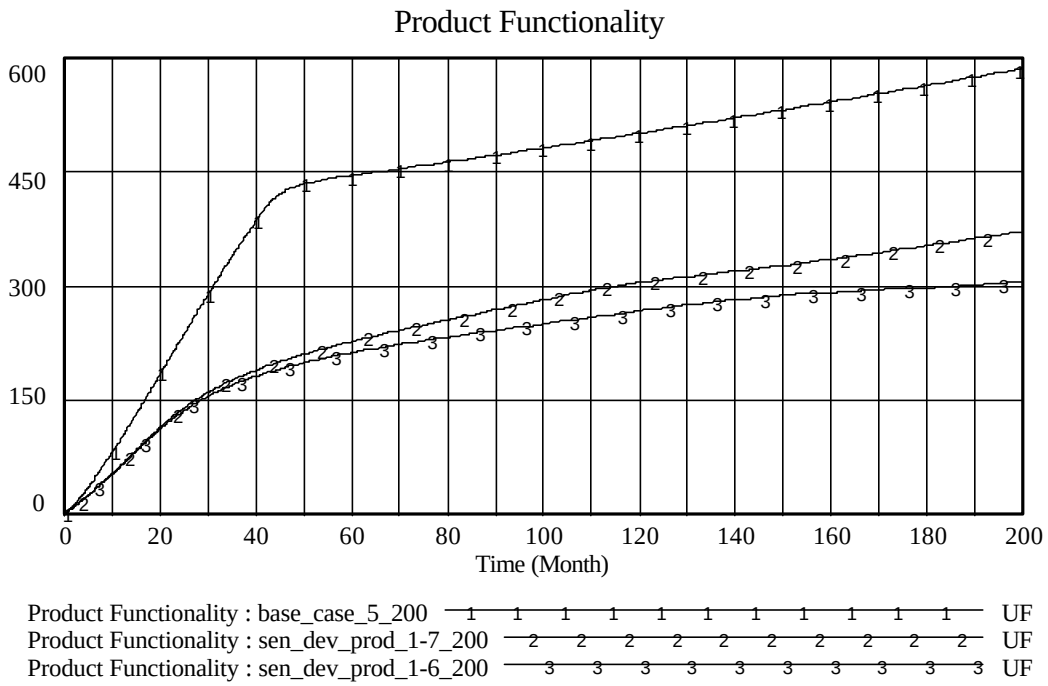


Figure 5.60. Product Functionality for Different Values of Average Developer Productivity

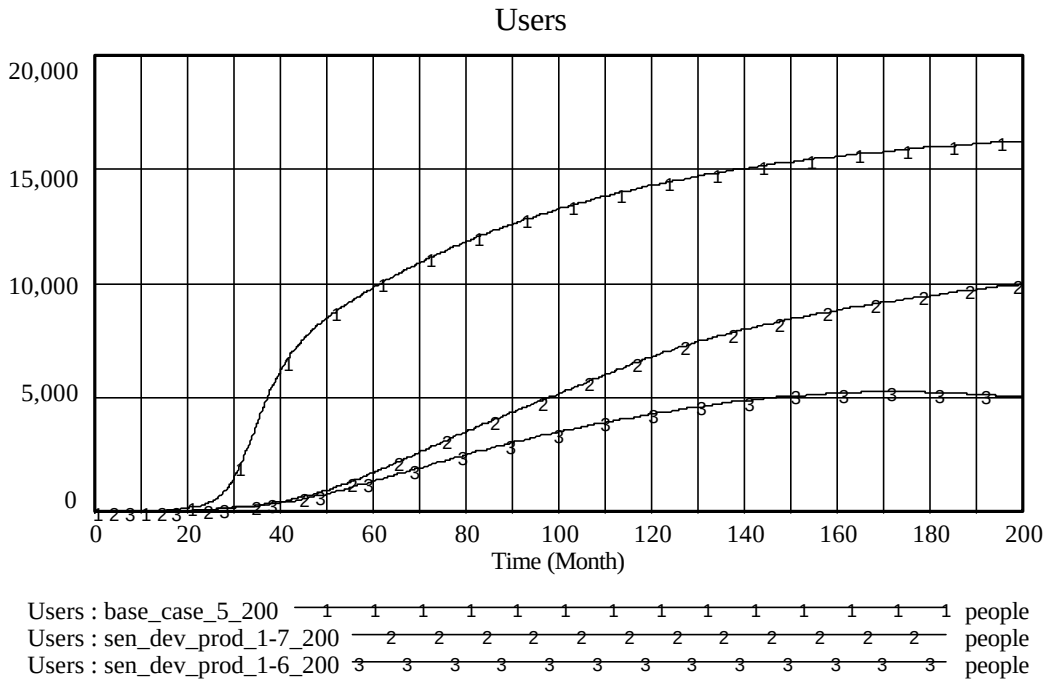


Figure 5.61. Users for Different Values of Average Developer Productivity

Like the case with average developer participation, higher values of average developer productivity yielded larger decreases in perceived product quality at the beginning of the project. (See Figure 5.62.) This is again attributable to the increased portion of code produced by developers in the overall code base as average developer productivity increased. The improvement in perceived product quality was faster as the average production level increased. Perceived product quality stayed high for very low average developer productivity levels, again due to the limited amount of code produced by developers. (See Figure 5.62.)

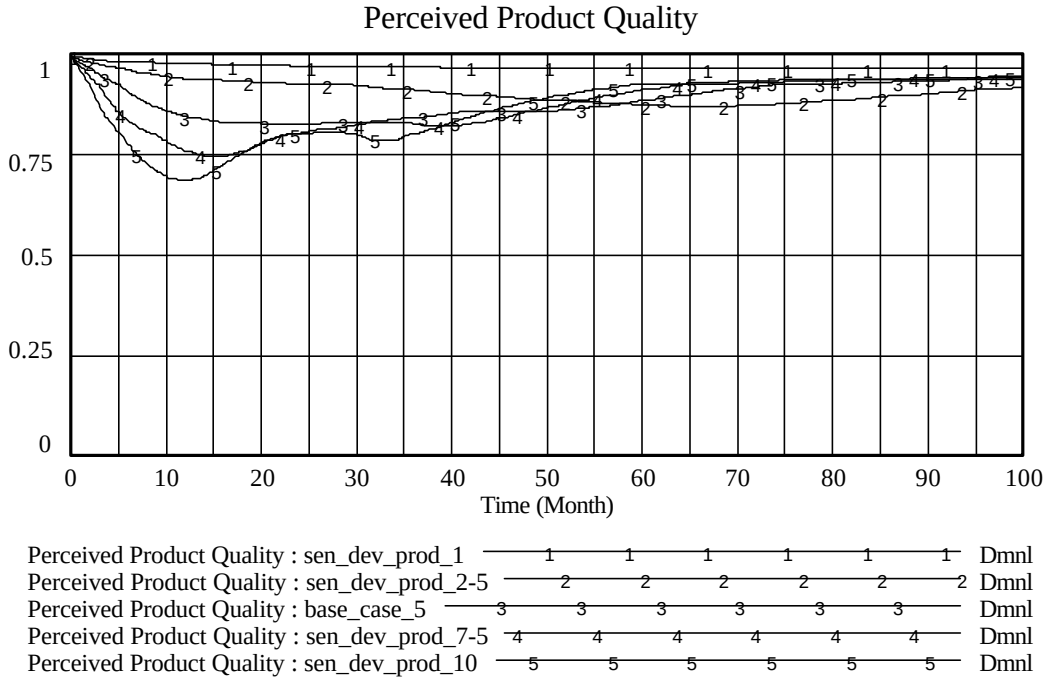


Figure 5.62. Users for Different Values of Average Developer Productivity

5.4.3. Bug Generating Rate Normal

The model was also run under different values of bug generating rate normal, namely 0.002, 0.005, 0.020, 0.050 bugs/line. The base case value of bug generating rate normal was 0.010 bugs/line.

As expected, higher values of bug generating rate normal caused higher levels of total bugs per functionality, and consequently, lower levels of perceived product quality. (See Figure 5.63. and Figure 5.64.) Perceived product quality improved after a decline in most runs, but it failed to do so in some runs with very high values of bug generating rate normal. The run where the rate was set to 0.050 bugs/line was one of those cases, as seen in Figure 5.64. This indicated that bug generating rate normal should also have a critical value, above which the community would fail due to low product quality.

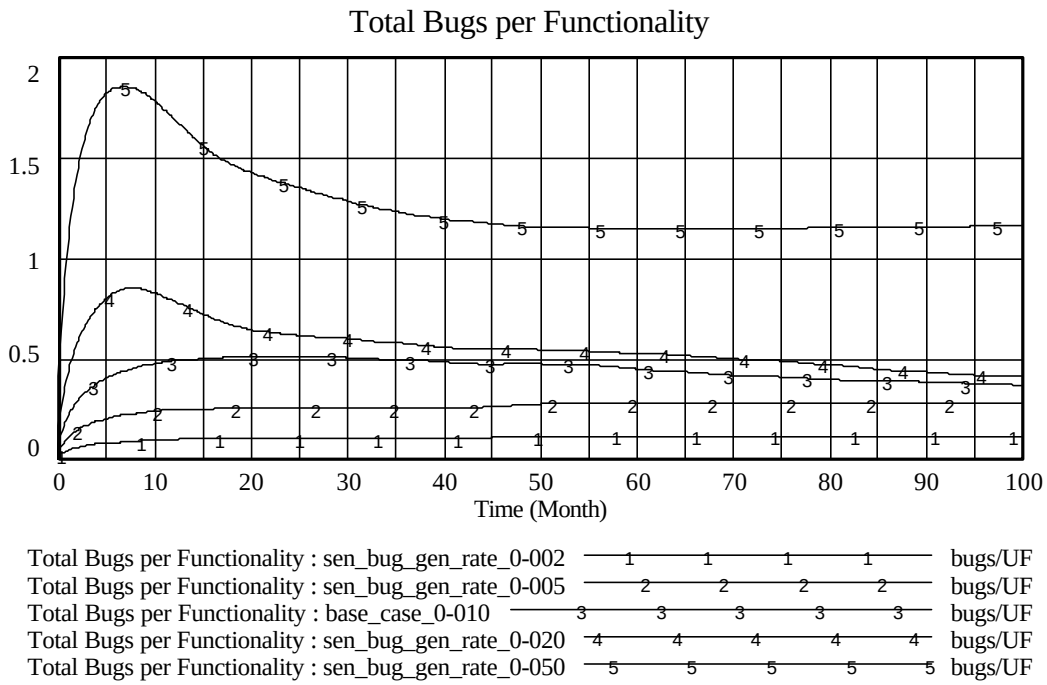


Figure 5.63. Total Bugs per Functionality for Different Values of Bug Generating Rate Normal

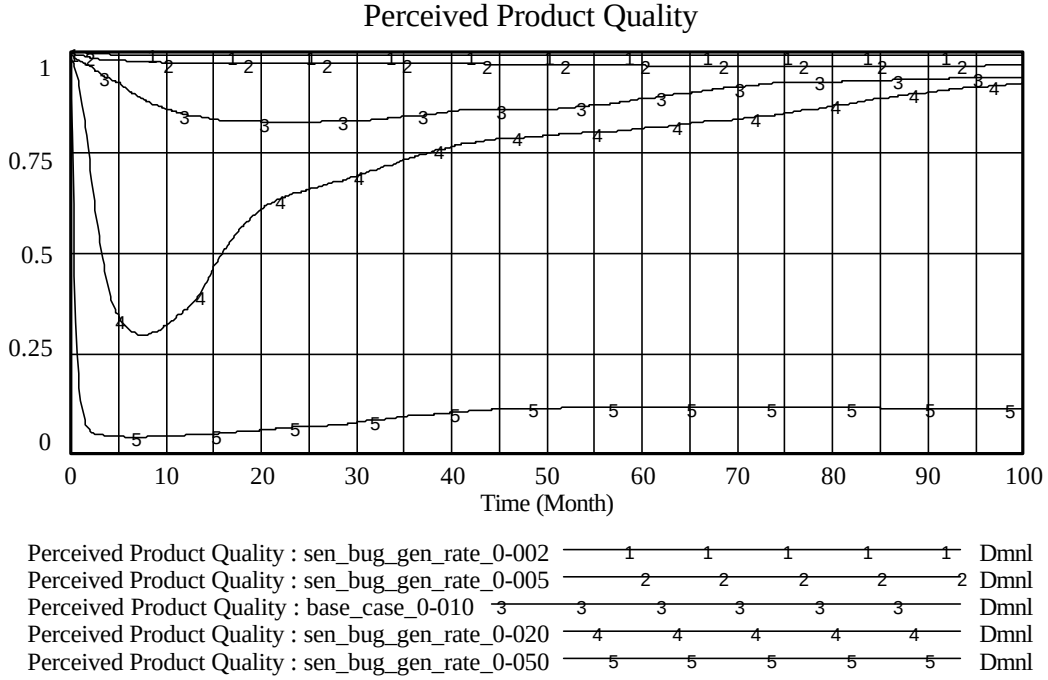


Figure 5.64. Perceived Product Quality for Different Values of Bug Generating Rate Normal

The behaviors of product functionality, number of users, and number of total participants in the run where the rate was set to 0.050 bugs/line supported the idea about the existence of a critical value for bug generating rate normal. (See Figures 5.65 through 5.67.) Based on further runs, the critical value for bug generating rate normal was found to lie between 0.025 and 0.030 bugs/line. (See Figures 5.68 through 5.70)

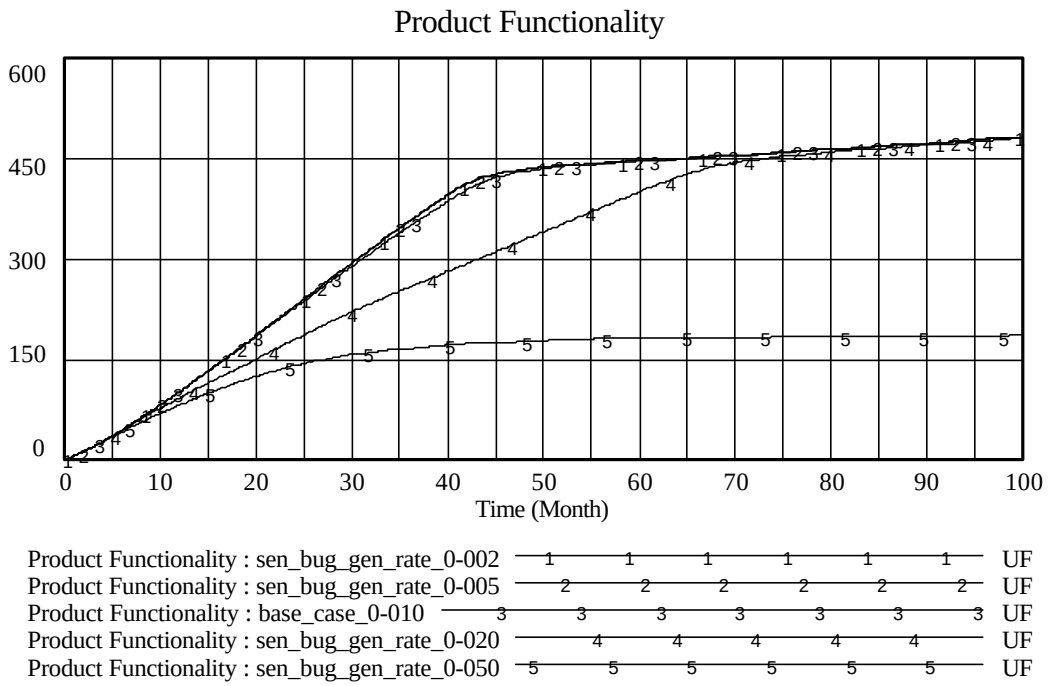


Figure 5.65. Product Functionality for Different Values of Bug Generating Rate Normal

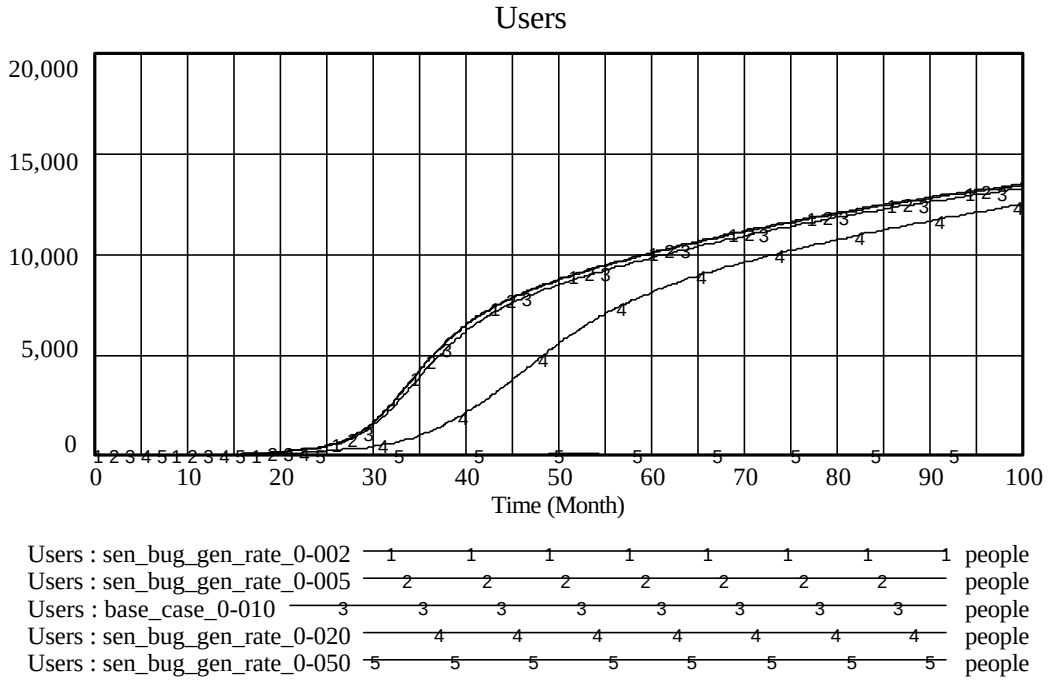


Figure 5.66. Users for Different Values of Bug Generating Rate Normal

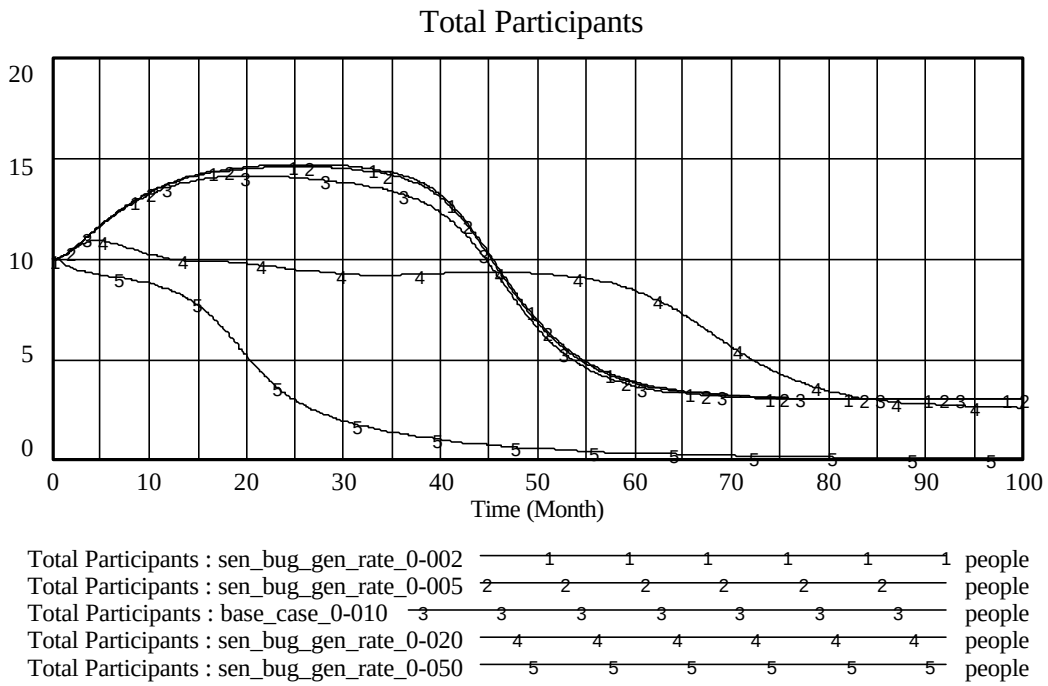


Figure 5.67. Total Participants for Different Values of Bug Generating Rate Normal

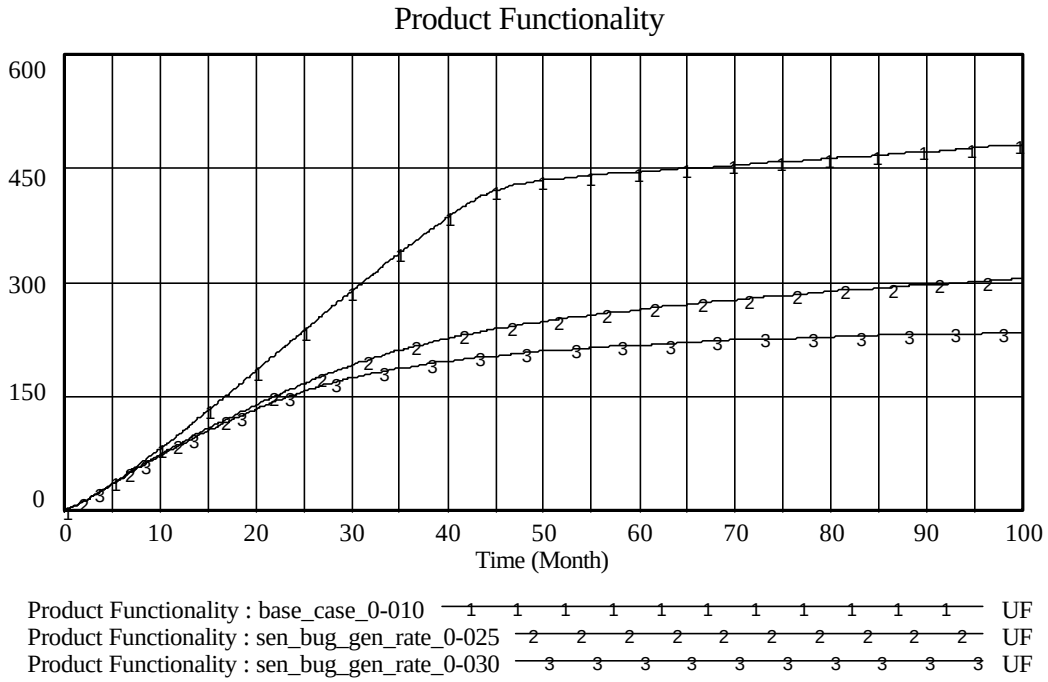


Figure 5.68. Product Functionality for Different Values of Bug Generating Rate Normal

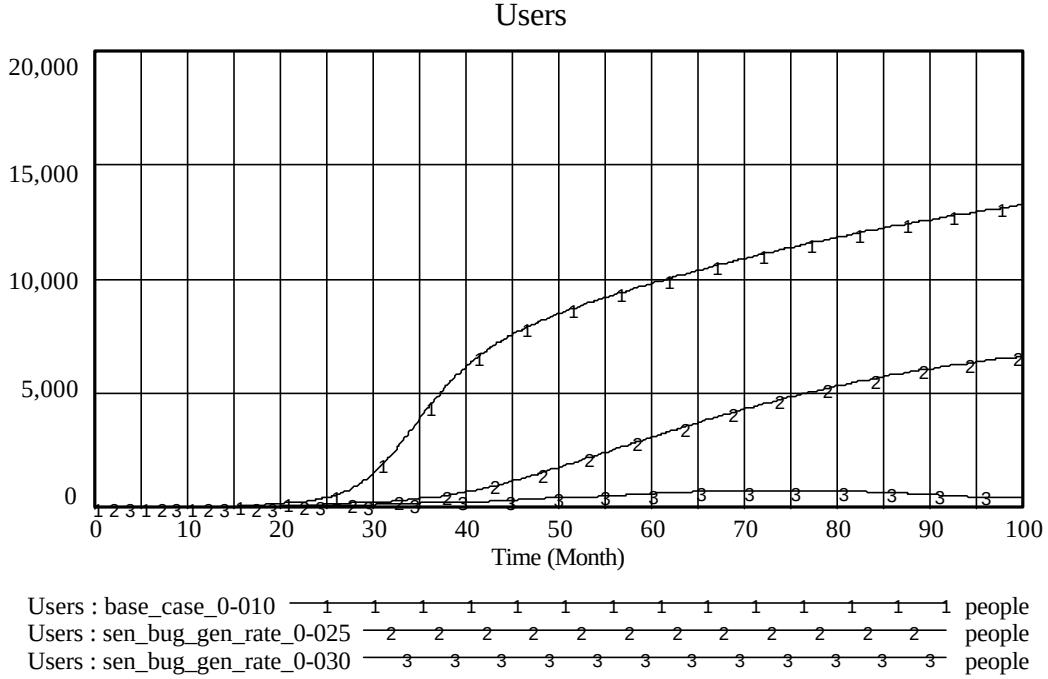


Figure 5.69. Users for Different Values of Bug Generating Rate Normal

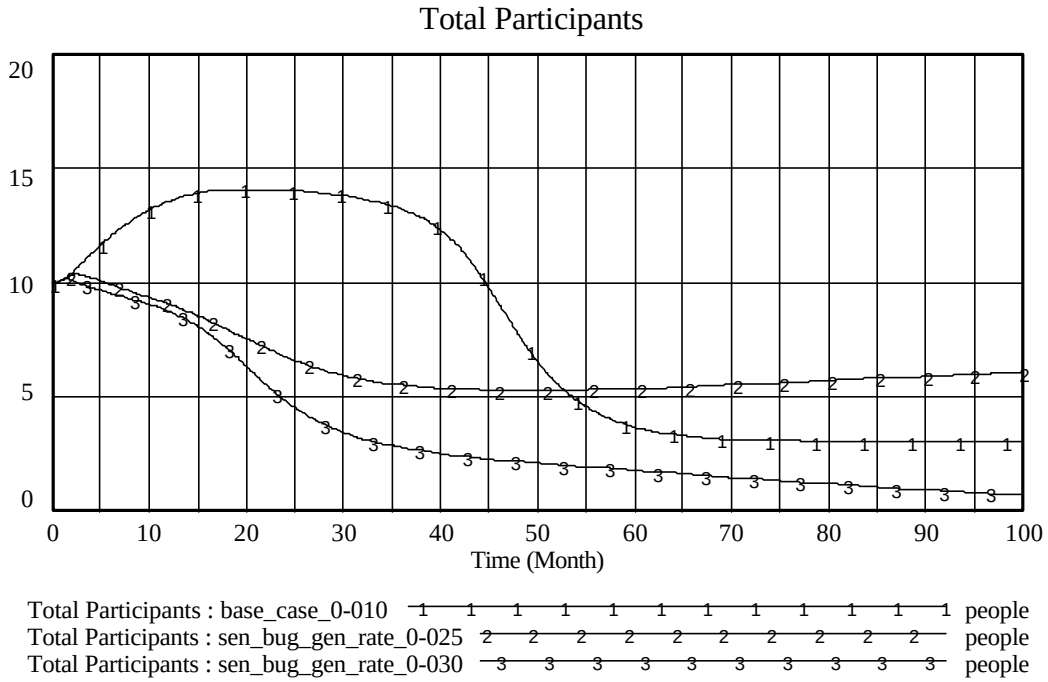


Figure 5.70. Total Participants for Different Values of Bug Generating Rate Normal

5.4.4. Normal Time to Attract Developers

Normal time to attract developers was the basis for another set of sensitivity runs. In the base case the value of the normal time to attract developers was 10 months. The runs where normal time to attract developers was set to 2, 5, 20 and 30 months are discussed below.

As expected, lower values of normal time to attract developers caused the number of developers to increase faster. Also, the decline in the number of developers happened earlier for runs with lower normal times to attract developers, since the limit on product functionality was achieved earlier due to a larger developer population. (See Figure 5.71 and Figure 5.72.) As a consequence of faster functionality growth, number of users grew faster under higher values of normal time to attract developers. (See Figure 5.73.)

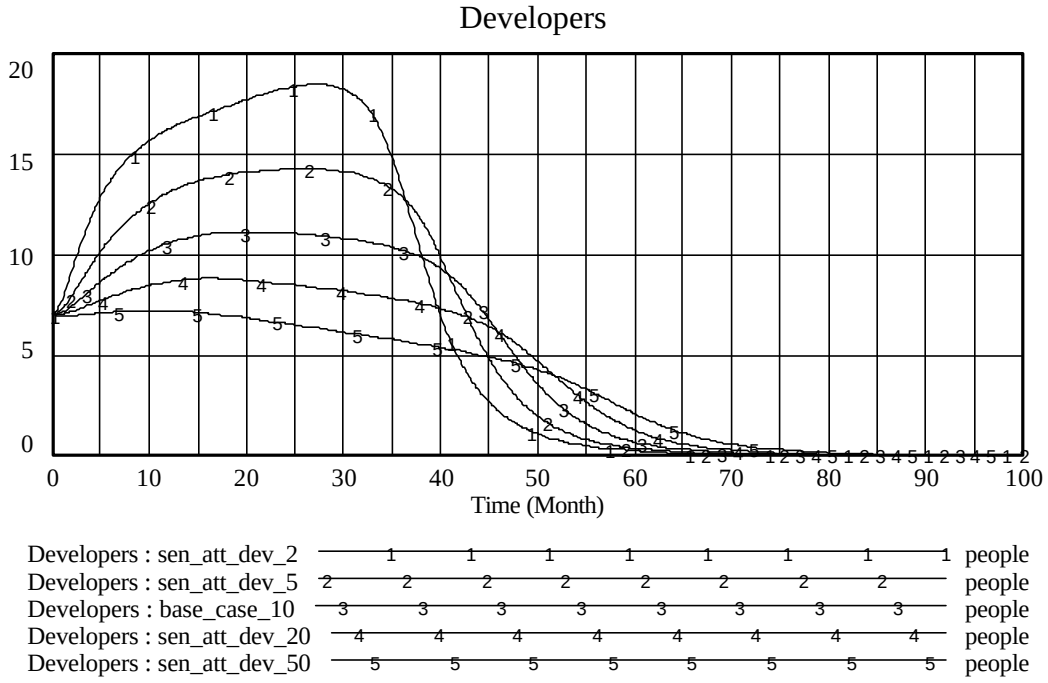


Figure 5.71. Developers for Different Values of Normal Time to Attract Developers

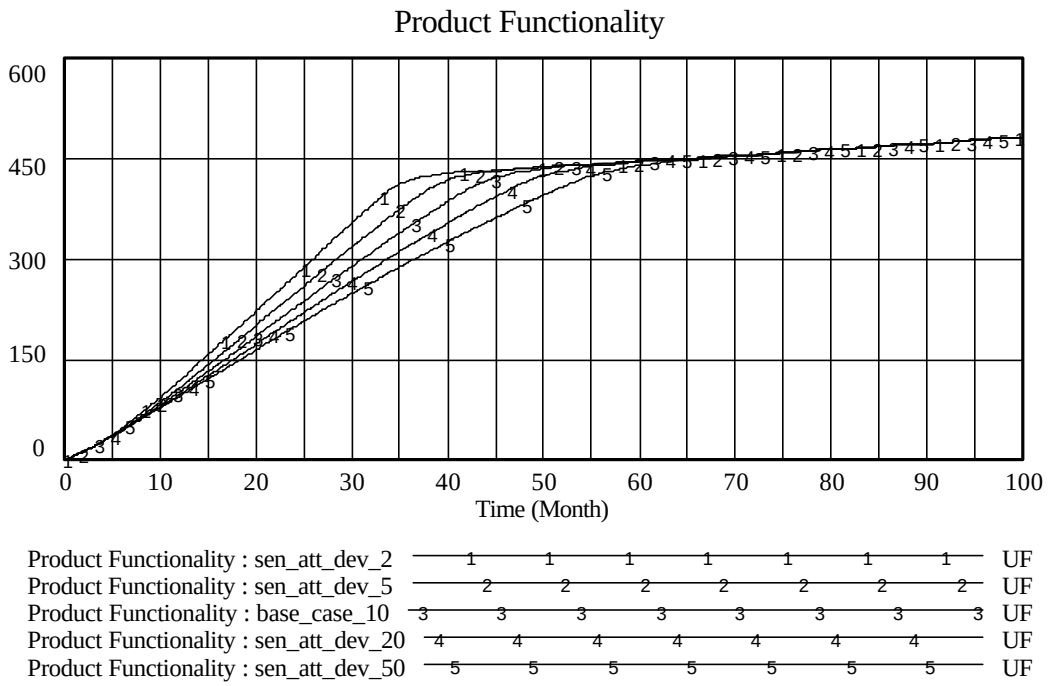


Figure 5.72. Product Functionality for Different Values of Normal Time to Attract Developers

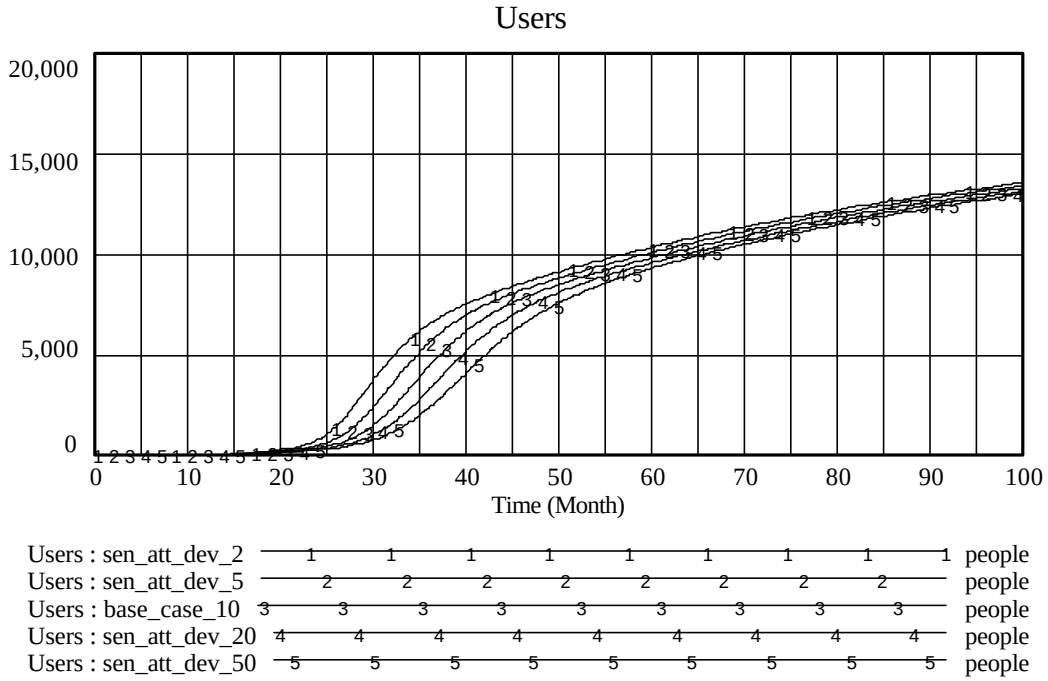


Figure 5.73. Users for Different Values of Normal Time to Attract Developers

Perceived product quality decreased faster and reached a lower level for runs with higher values of normal time to attract developers. This was due to the higher portions of code produced in these runs due to higher numbers of developers. However, perceived product quality improved and reached to about the same level by the end of the simulation horizon in all the runs. (See Figure 5.74.)

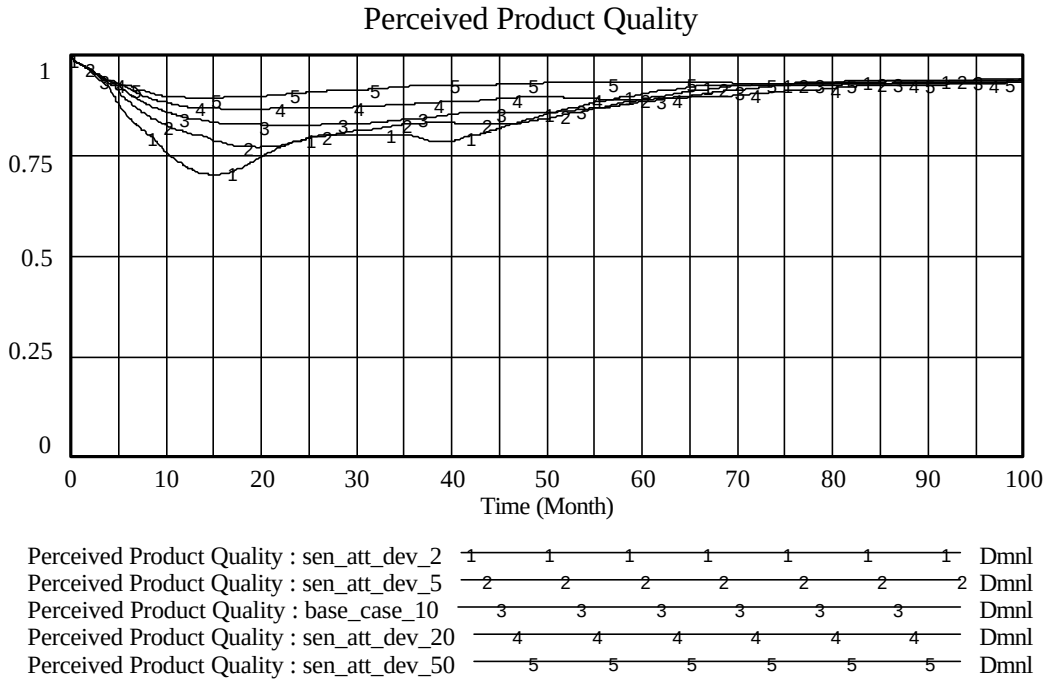


Figure 5.74. Users for Different Values of Normal Time to Attract Developers

Average developer talent increased faster and reached higher equilibriums for higher values of normal time to attract developers, since the developers stayed in the community longer, and thus had a longer period of coaching than in the runs with lower normal times to attract developers. (See Figure 5.75.)

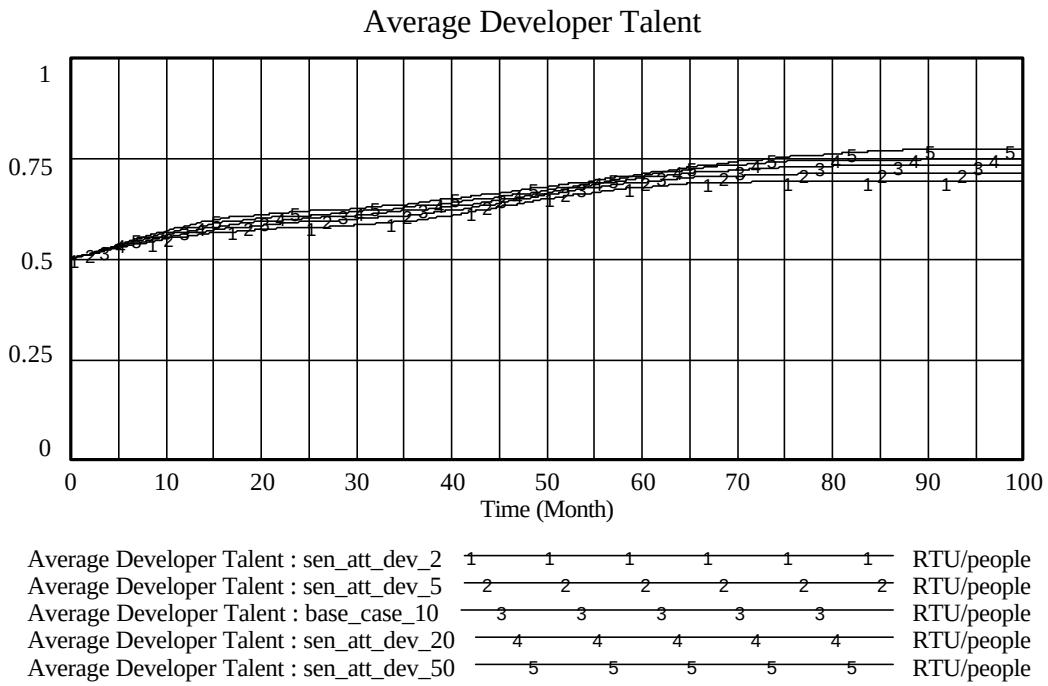


Figure 5.75. Users for Different Values of Normal Time to Attract Developers

It can be argued that the model did not yield the expected diversity of behavior for different values of normal time to attract developers. It was expected that the community would fail to grow to a sustainable level for very high values of this variable. However, even five times the base case value did not yield such as result. This fact was noted as an opportunity for a future model refinement study, where this and other variables would be revised in order to improve the model.

5.4.5. Normal Time for Developers to Leave

Another set of sensitivity runs was based on different values of normal time for developers to leave, namely 16, 48, 144 and 198 months. The value of normal time for developers to leave was 96 months in the base case. The findings of these runs were not too different than those of the runs under different values of normal time to attract

developers, except for the fact that the diversity of behavior was even smaller in this case. Hence, this variable was noted as a candidate for a future model refinement study, as well.

Although the behavioral differences among the runs were not substantial, the number of developer started to decrease earlier for runs with lower values of normal time for developers to leave, as expected. That was due to the increased number of leaving developers in these runs. (See Figure 5.76.)

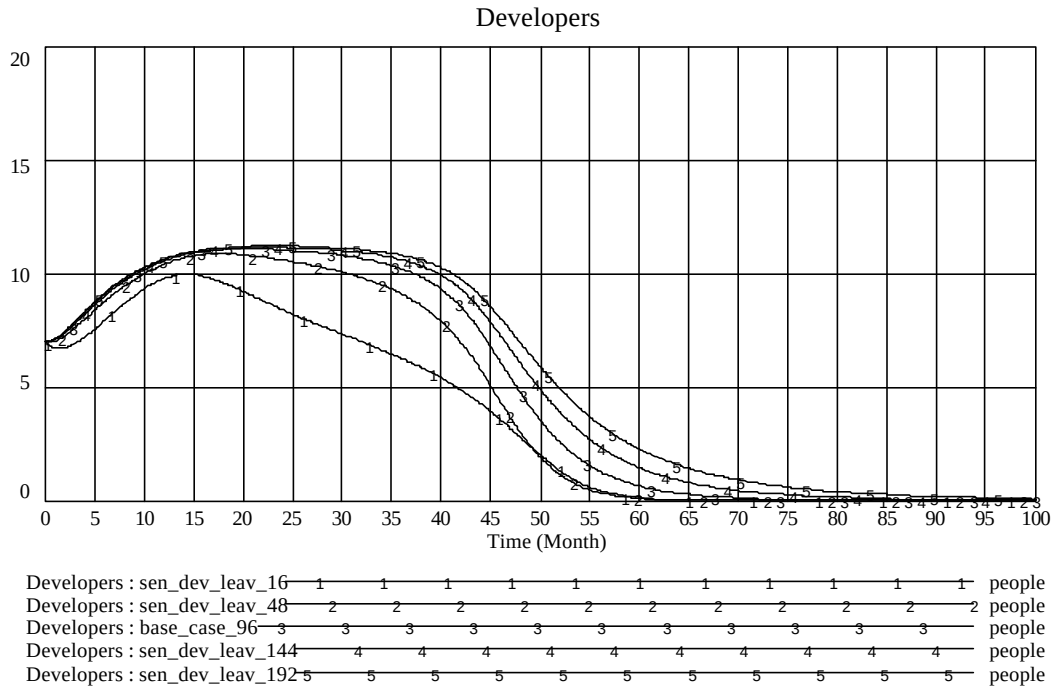


Figure 5.76. Developers for Different Values of Normal Time for Developers to Leave

Since the developers left the community earlier, causing the number of developers to stay lower, product functionality growth under lower values of normal time for developers to leave was slower. However, the differences in the pace of functionality

growth were far from being substantial. (See Figure 5.77.) That was another fact that cast doubt on the validity of the way this variable was included in the model.

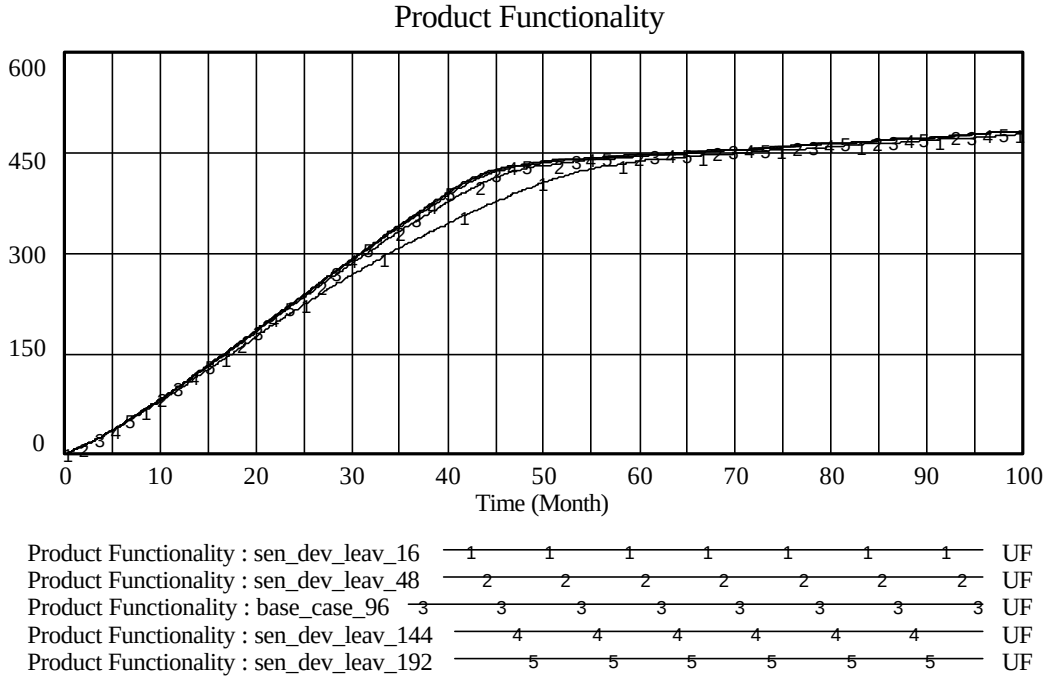


Figure 5.77. Product Functionality for Different Values of Normal Time for Developers to Leave

Since the differences in the pace of product functionality growth across the runs were very small, the differences between the behaviors of the number of users in each run were also small, contrary to the expectation. Nevertheless, the number of users increased more slowly under lower values of normal time for developers to leave, due to slower product functionality growth. (See Figure 5. 78.)

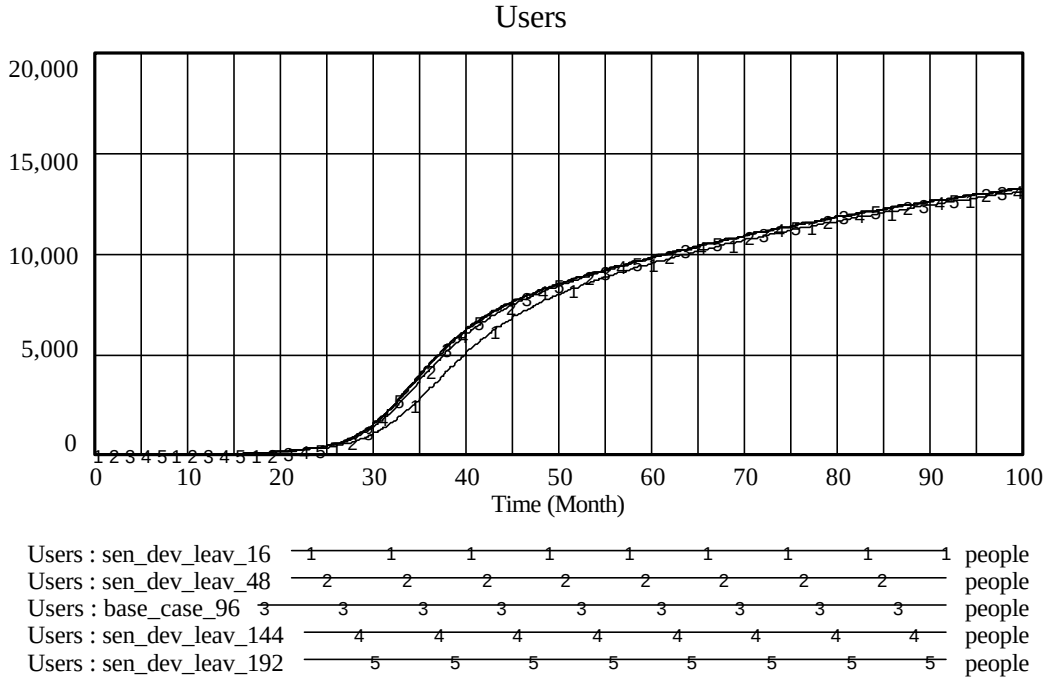


Figure 5.78. Users for Different Values of Normal Time for Developers to Leave

Perceived product functionality exhibited behaviors which were essentially the same for different for different values of normal time for developers to leave, again contrary to expectation. However, it should be noted that the product quality was slightly better for runs with lower values of normal time for developers to leave. (See Figure 5. 79.) This can be attributed to the lower portion of code produced by developers within the overall code base in these runs.

Average developer talent increased slightly faster and reached higher equilibriums for runs with higher values of normal time for developers to leave, as expected. This was due to developers staying in the community longer, and thus having a longer period of coaching. (See Figure 5.80.)

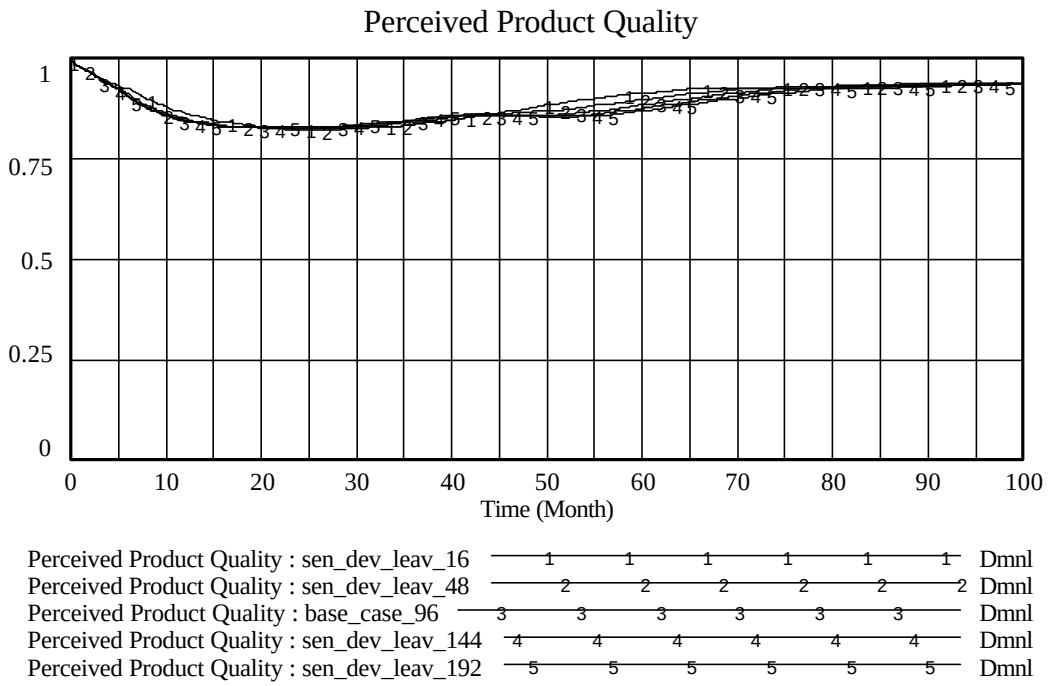


Figure 5.79. Perceived Product Quality for Different Values of Normal Time for Developers to Leave

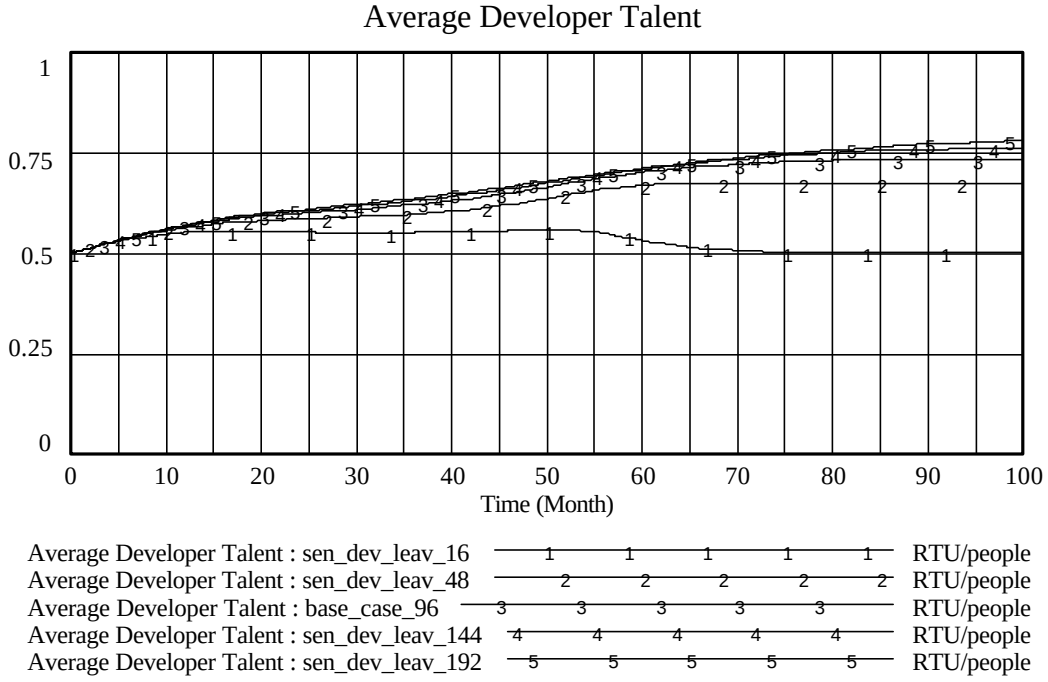


Figure 5.80. Average Developer Talent for Different Values of Normal Time for Developers to Leave

5.4.6. Normal Time to Attract Users

Another set of sensitivity runs was done for different values of normal time to attract users. The runs where the variable was set to 6, 18, 72 and 108 months are discussed below. As expected, the growth in the number of users was slower for runs with higher value of normal time to attract users. (See Figure 5.81.)

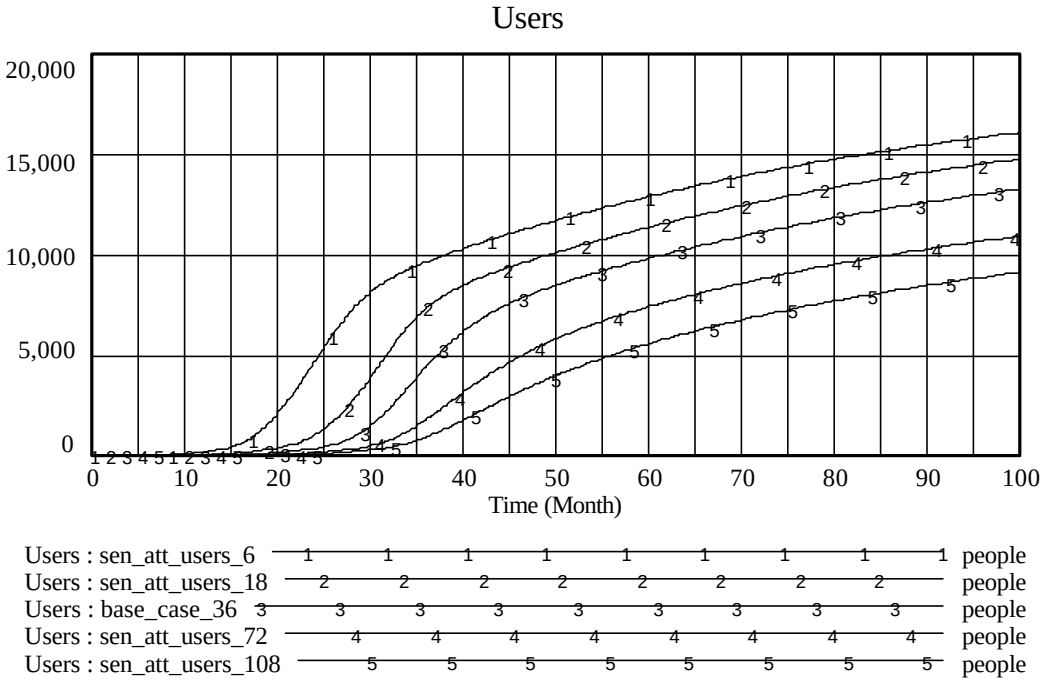


Figure 5.81. Users for Different Values of Normal Time to Attract Users

The number of users was modeled as a critical motivation factor for the developers to join the community. Consequently, it was expected that the number of developers would increase considerably faster in cases where the number of users increased faster. However, the change in the behavior of the number of developers as the normal time to attract users changed was smaller than expected. (See Figure 5.82.)

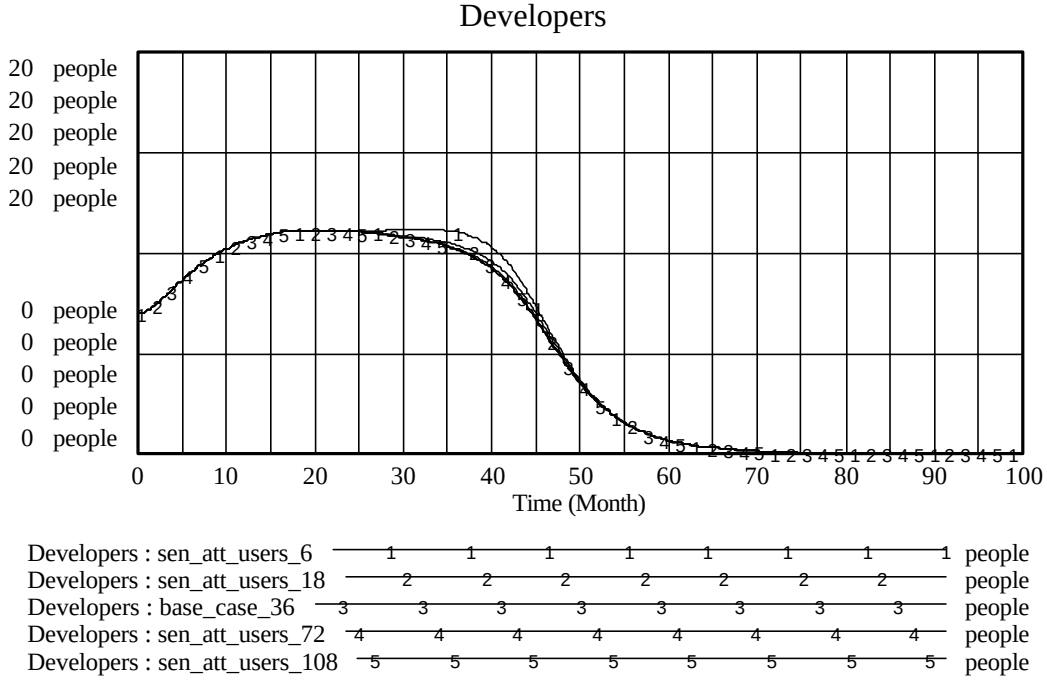


Figure 5.82. Developers for Different Values of Normal Time to Attract Users

Further analysis revealed that the attractiveness of joining the project due to the number of users changed considerably for different values of normal time to attract users. (See Figure 5.83.) However, overall attractiveness of joining the project did not change as much, except for very low values of normal time to attract users. (See Figure 5.84.) Even in such cases the difference occurred over a limited period. For example, for the run where normal time to attract users was set to 6 months, the difference was limited to the period between months 20 and 40, and it was not large enough to change the behavior of the number of developers substantially. (See Figure 5.84.) Product functionality growth was not accelerated, due to the limited acceleration in the growth of the number of developers. (See Figure 5.85) Normal time to attract users, too, was noted as a candidate for a future model improvement study, since it caused suspicion about the confidence level of the model.

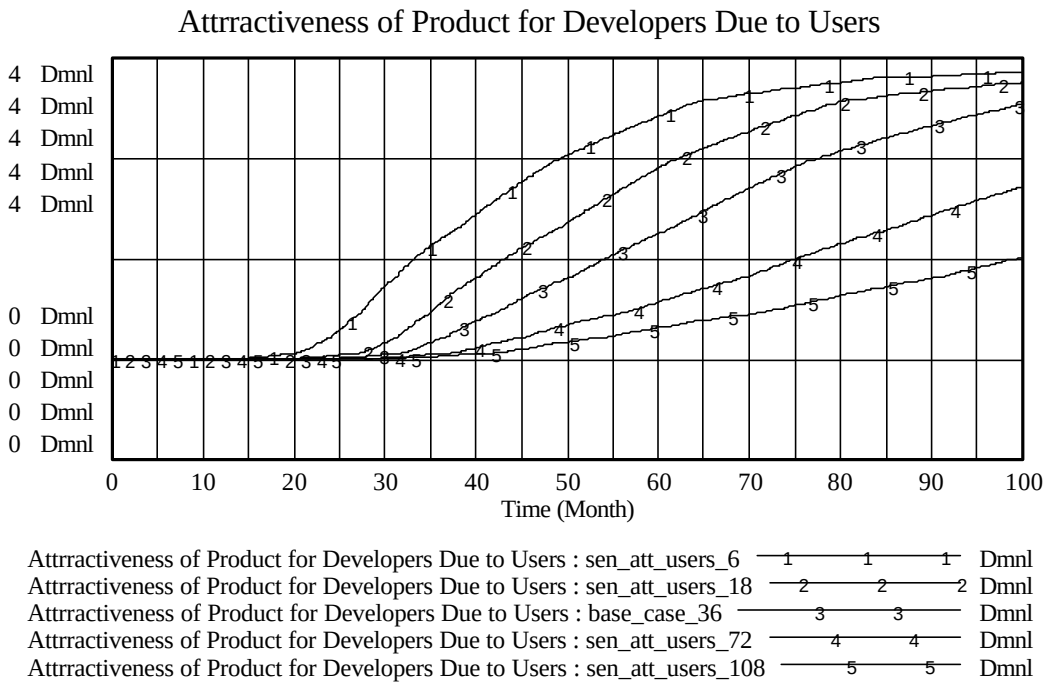


Figure 5.83. Attractiveness of Product for Developers Due to Users for Different Values of Normal Time to Attract Users

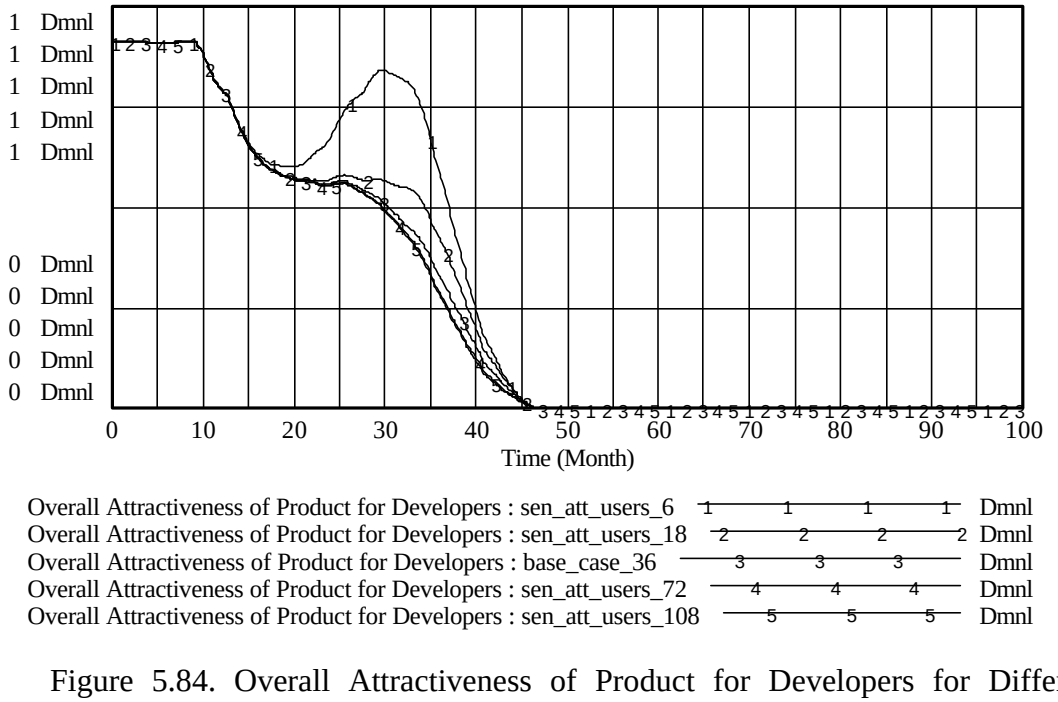


Figure 5.84. Overall Attractiveness of Product for Developers for Different Values of Normal Time to Attract Users

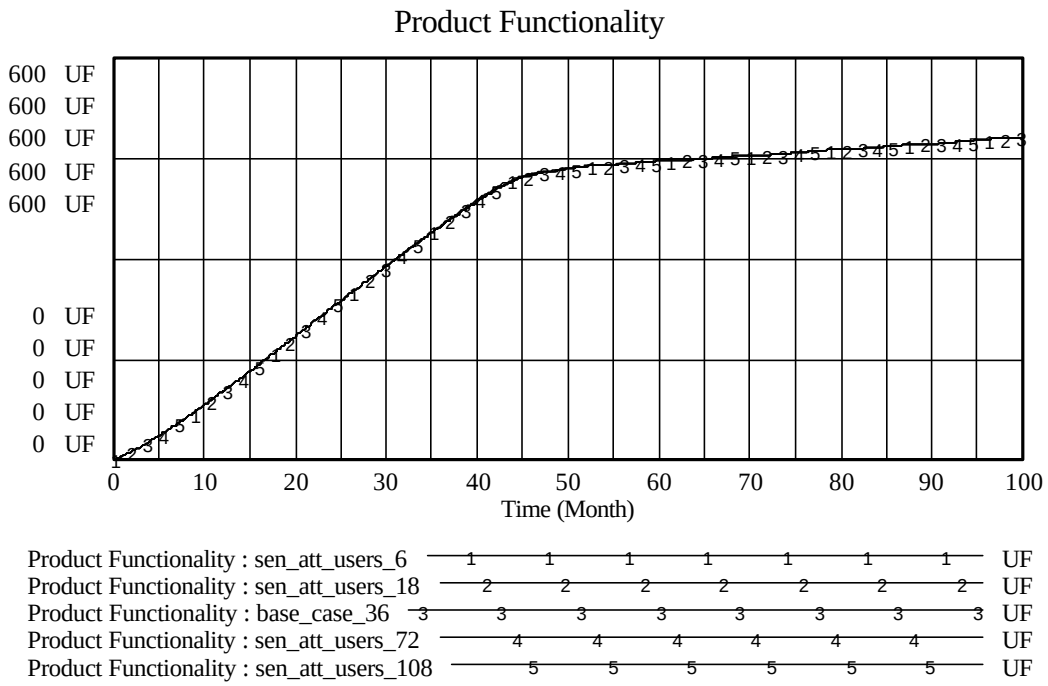


Figure 5.85. Product Functionality for Different Values of Normal Time to Attract Users

5.4.7. Refusal Ratio

Refusal ratio was the basis for another set of sensitivity runs. In the base case run refusal rate was set to 0.1. The sensitivity runs where the refusal rate was set to 0.02, 0.05, 0.3 and 0.8 are discussed below.

Refusal ratio directly affects two things: the number of incoming developers and the average talent level of those incoming developers. As refusal ratio increases, a smaller number of developers with a higher average talent level join the community. Accordingly, higher values of refusal ratio were expected to decrease the number of developers, and increase the average developer talent. As expected, higher refusal ratios decreased the number of incoming developers, and consequently the number of developers. (See Figure 5.86 and Figure 5.87.)

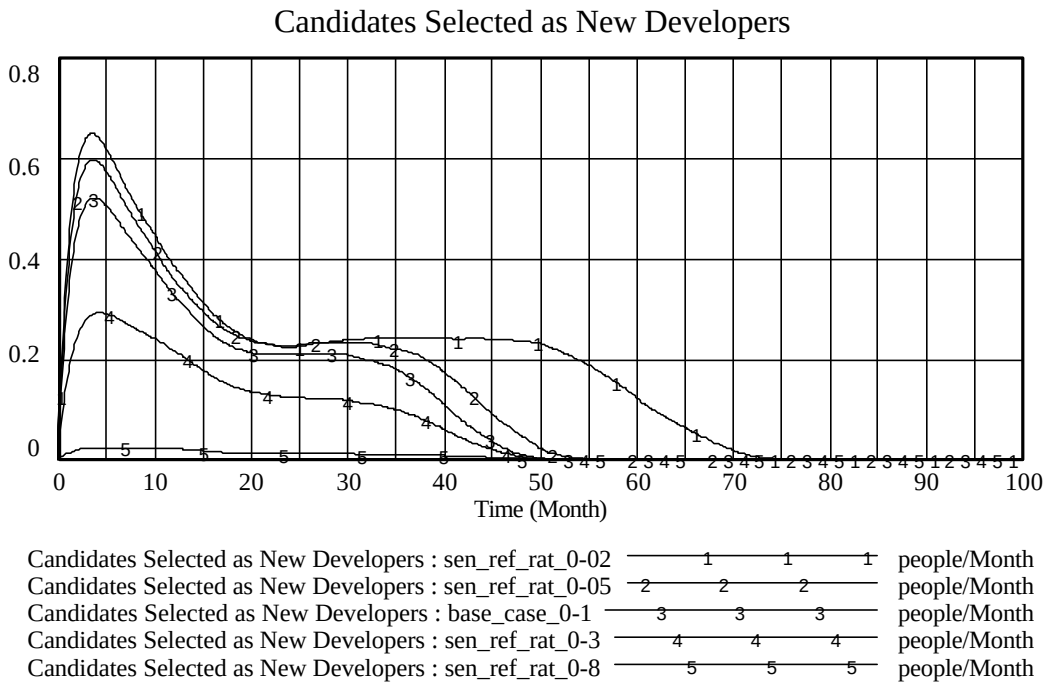


Figure 5.86. Candidates Selected as New Developers for Different Values of Refusal Ratio

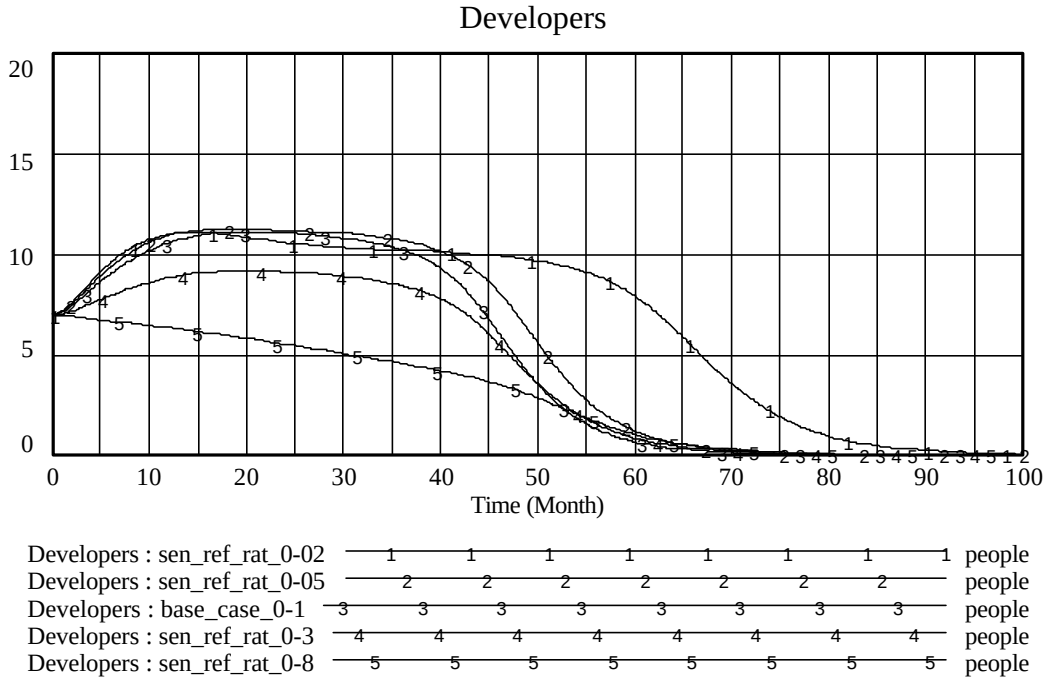


Figure 5.87. Developers for Different Values of Refusal Ratio

However, when the refusal ratio was set to 0.02, a very low level, the initial increase in the number of developers was followed by an earlier decrease. This was attributed to the large decrease in perceived product quality, which itself was a consequence of code produced by developers with a very low talent level. (See Figure 5.88.) In general, perceived product quality decreased less in runs with higher refusal ratios, as expected. (See Figure 5.88.) This is attributable to the fact that average developer talent started at higher levels and increased even higher in runs with higher refusal ratios. (See Figure 5.89.) Basically, better developers produced better code.

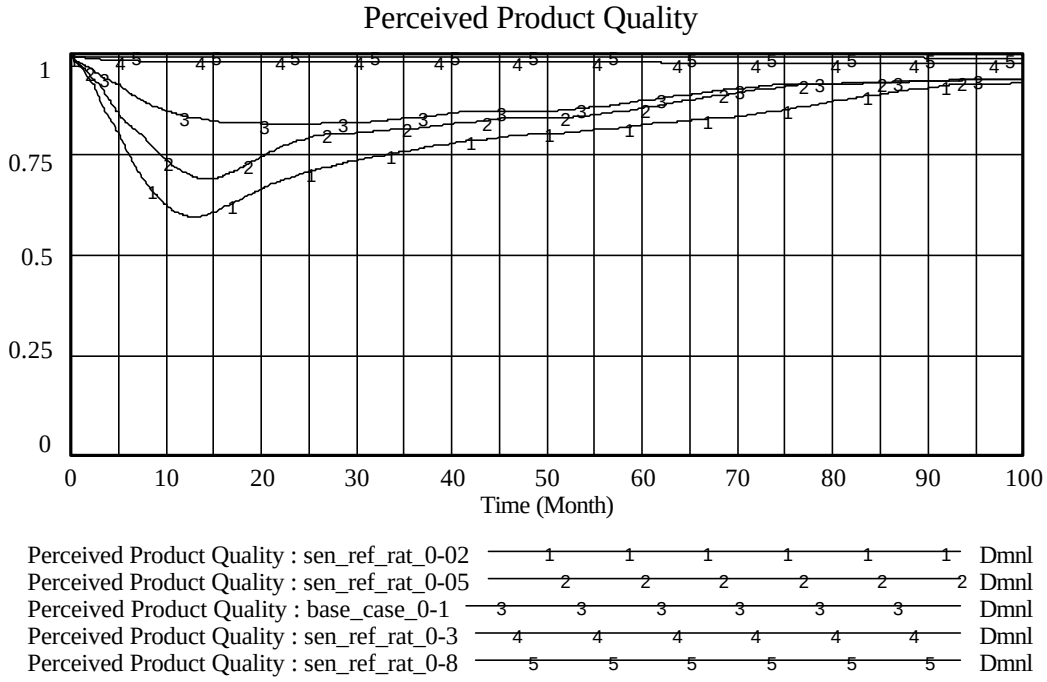


Figure 5.88. Perceived Product Quality for Different Values of Refusal Ratio

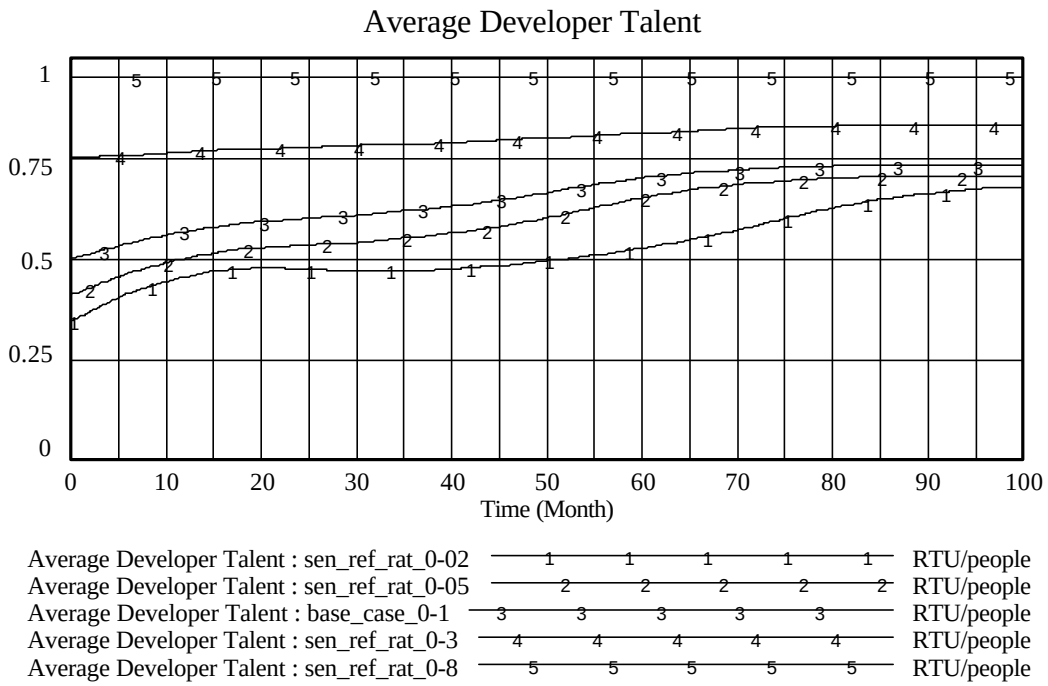


Figure 5.89. Average Developer Talent for Different Values of Refusal Ratio

Higher refusal ratios impede the increase of the number of developers. Since a smaller number of developers would produce a smaller amount of product functionality, it was expected that the growth of product functionality would be slower under higher refusal ratios. However, higher refusal ratios did not always yield slower functionality growths. (See Figure 5.90.) In fact, the slowest functionality growth among the runs in the exhibited set took place under a very low refusal ratio level. Although the number of developers was higher for a longer period of time in that run, a lot of the available developer time had to be channeled to debugging and coaching activities, instead of production. On the other hand, increasing the refusal ratio beyond a point yielded slower functionality growth. (See Figure 5.90.) Since the marginal quality gain by increasing the refusal ratio became very small in such runs, it was concluded that there should be a

critical value of refusal ratio that would yield an optimal combination of higher quality and faster functionality growth. This critical value was found to be around 0.3.

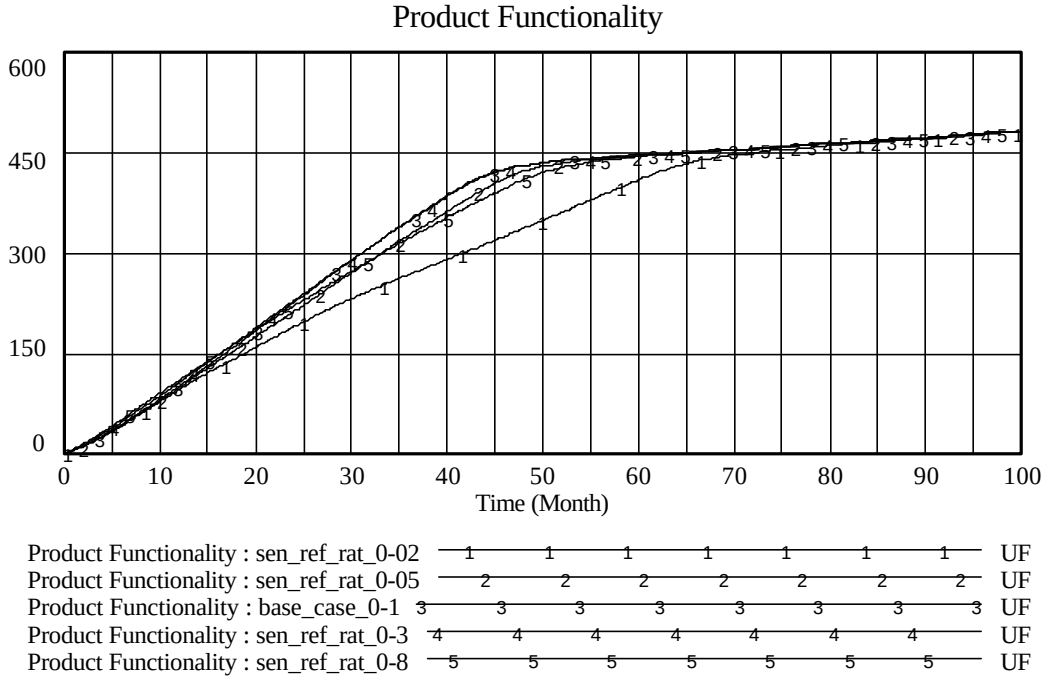


Figure 5.90. Product Functionality for Different Values of Refusal Ratio

Although the sensitivity runs based on different refusal ratios provided valuable insights about the model, the range of behaviors observed in these runs was smaller than expected. It was expected that the community would not be able to sustain itself with the limited number of developers under very high refusal ratios. However, it was found that the initial group of seven developers was enough to bring the product functionality above the critical level before they left the community, even if no new developers were accepted into the community. This was noted as a point to consider for future model improvement.

5.4.8. Rejection Ratio

Rejection ratio, too, was used as the basis for a set of sensitivity runs. The base case value of rejection ratio was 0.2. The sensitivity runs where the rejection ratio was set to 0.05, 0.1, 0.4 and 0.8 are discussed below.

Rejection ratio determines both the amount and the quality of the code added to the overall code base by developers. It also affects the level of average developer participation. A higher rejection ratio yields a smaller amount of code, which is of higher quality. A higher rejection ratio also yields a lower level of average developer participation. In the actual sensitivity runs, lower rejection ratios caused the total production to increase faster at the beginning of the project due to a greater amount of accepted code by developers, and a higher level of developer participation. (See Figure 5.91 and Figure 5.92.) For refusal ratios below 0.3, the fundamental behavior pattern of total production stayed the same; however it unfolded faster as the refusal ratio decreased. In other words, total production increased more slowly, but started to decrease later due to product functionality saturation, as the refusal ratio increased up to 0.3. The behavior pattern was different in runs with refusal ratios above 0.3. The initial increase continued to slow down as refusal ratio increased; however, the decrease started earlier, rather than later, under higher refusal ratios. The reason for the decrease in those runs was low functionality achievement, rather than the depletion of opportunities for contribution due to functionality saturation. This can be observed better in Figure 5.93, which displays the behaviors of the number of developers under different refusal ratios.

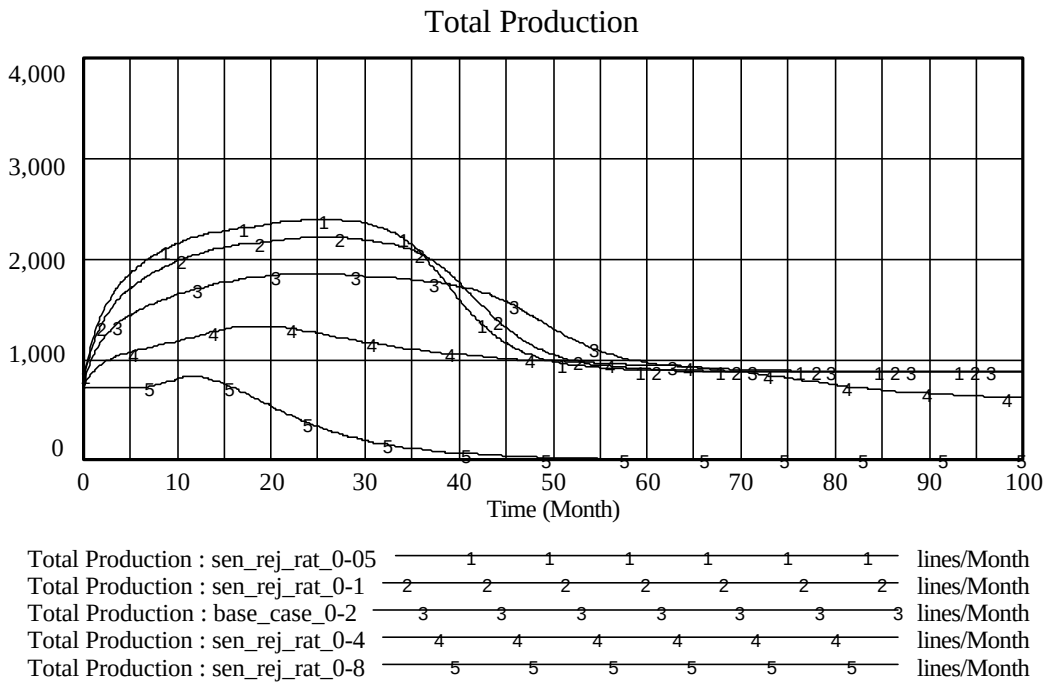


Figure 5.91. Total Product for Different Values of Rejection Ratio

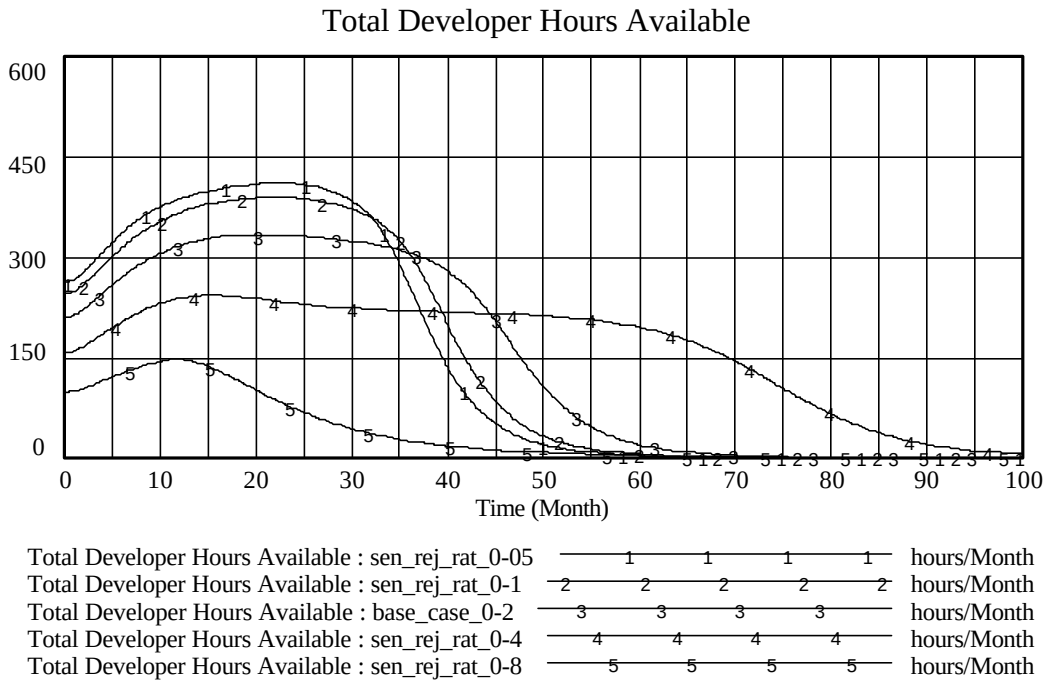


Figure 5.92. Total Developer Hours Available for Different Values of Rejection Ratio

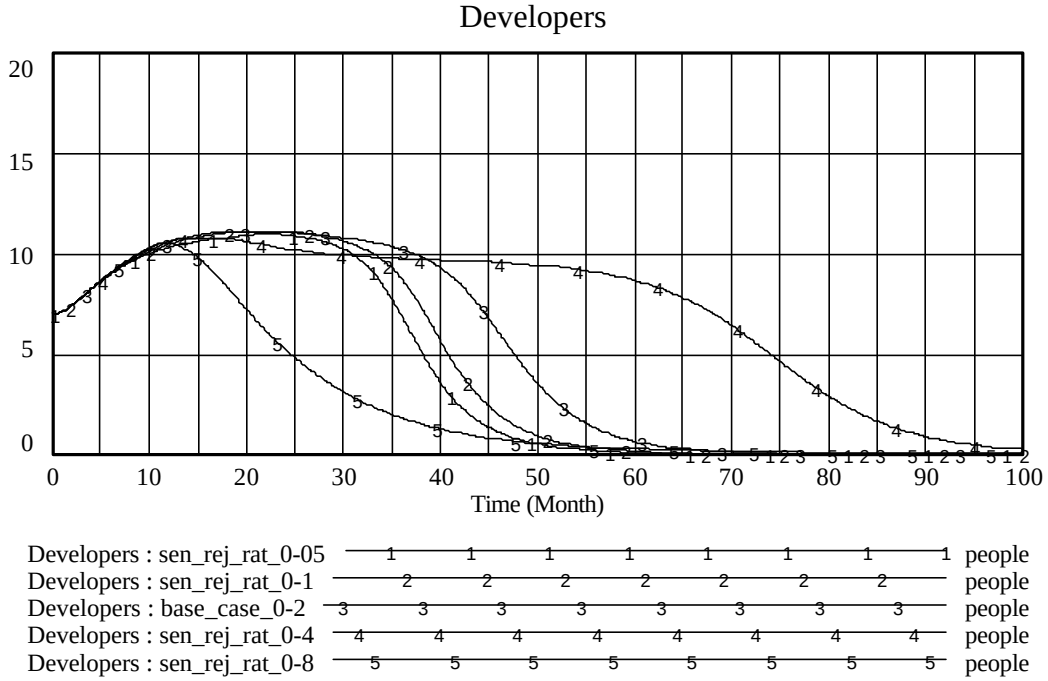


Figure 5.93. Developers for Different Values of Rejection Ratio

Functionality growth was slower for higher rejection ratios, as expected. (See Figure 5.94.) Observing the behaviors of the number of users under different rejection ratios indicated that rejection ratios above a critical value would cause the community to fail to sustain itself in the long run. (See Figure 5. 95) The critical value was found to lie between 0.50 and 0.55.

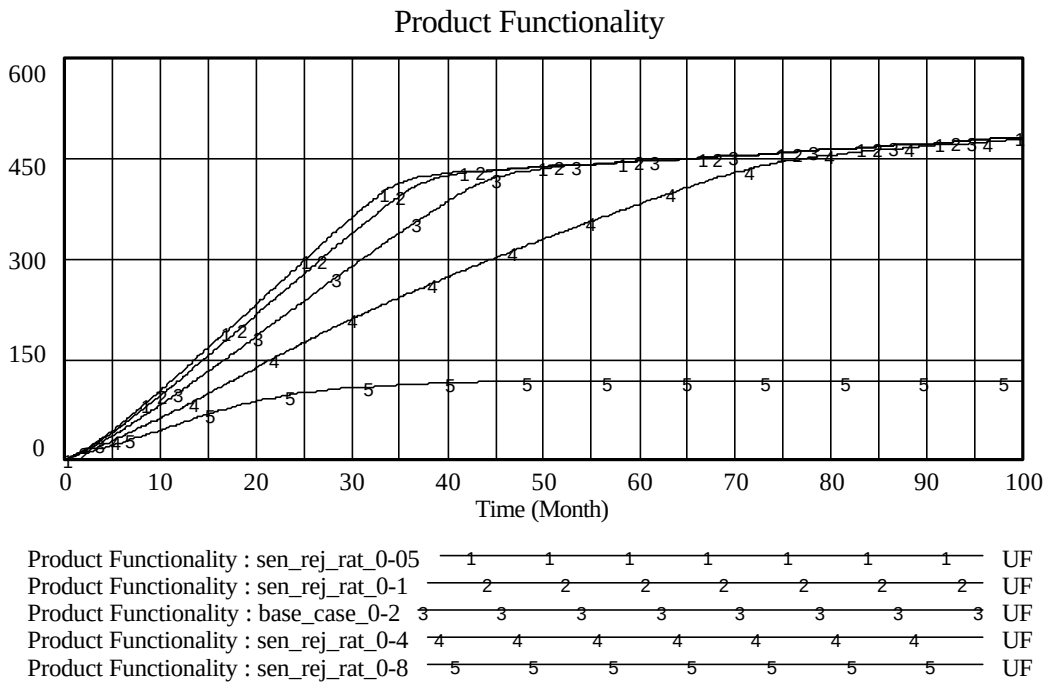


Figure 5.94. Product Functionality for Different Values of Rejection Ratio

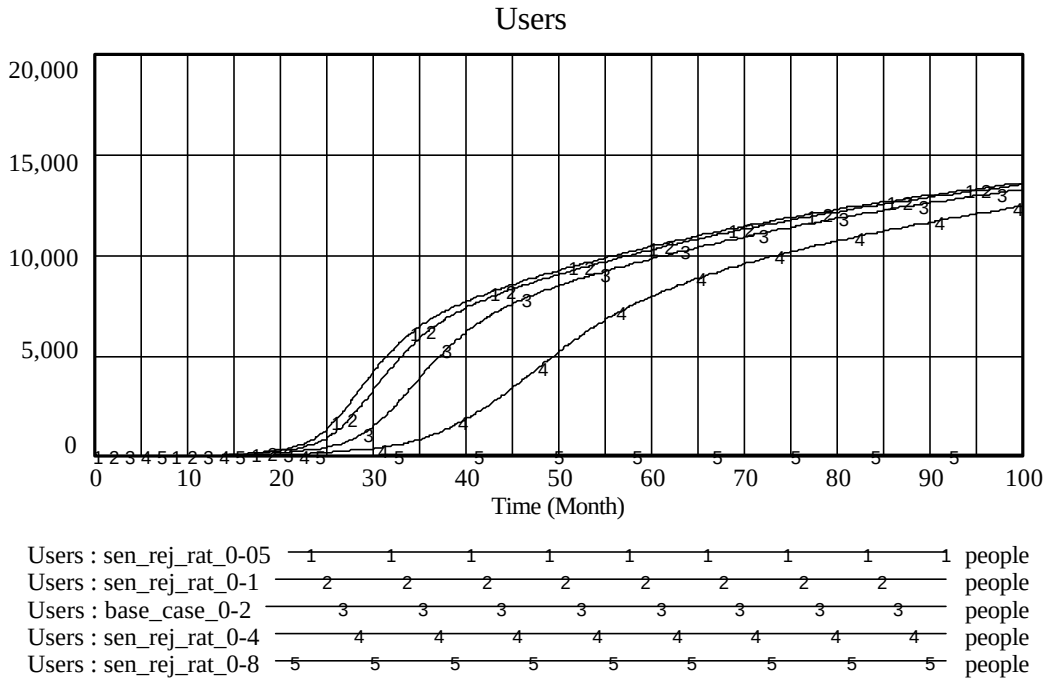


Figure 5.95. Users for Different Values of Rejection Ratio

An important finding was that the improvements in perceived product functionality caused by higher rejection rates were not as large as the improvements by higher rejection rates. (Compare Figure 5.88 and Figure 5.96) This was noted as an important implication for policy analysis runs, which followed the sensitivity analysis phase.

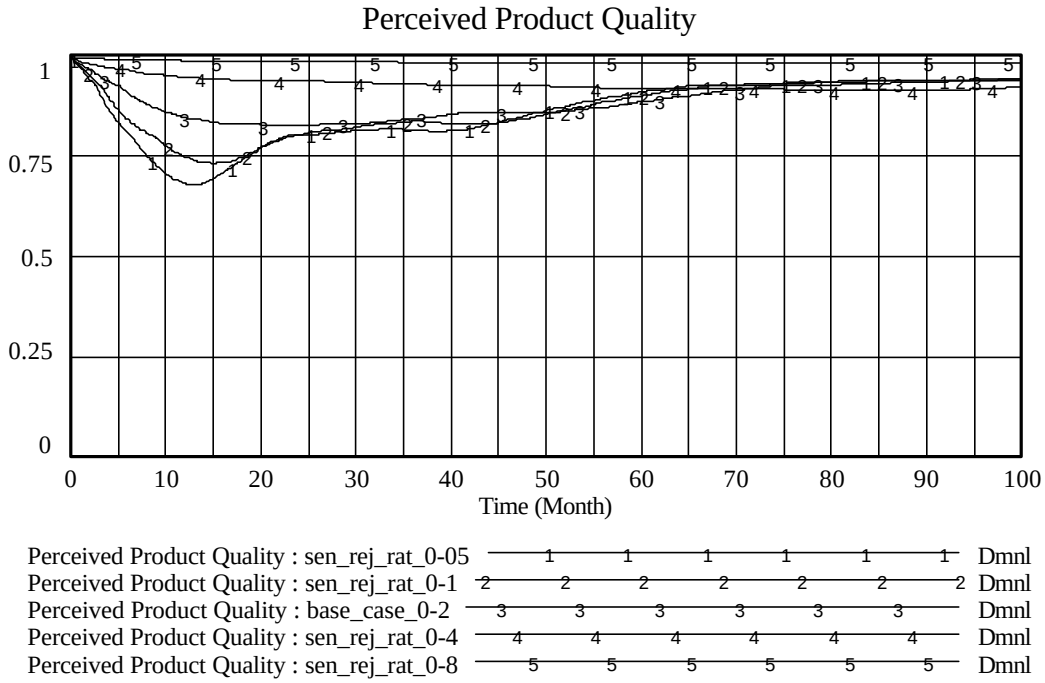


Figure 5.96. Perceived Product Quality for Different Values of Rejection Ratio

5.4.9. Implications of the Sensitivity Runs

Sensitivity runs provided critical insights about the OSSD model. An important finding was about variables that determine the amount of functionality added to the product within a given period of time. These variables, such as average developer participation and average developer productivity, have critical values below which an open source software community fails to sustain product functionality and community

growth. The critical value for a given variable can differ from community to community, but the fact that there are such critical values for these variables would hold for any open source software community.

Variables that eventually determine the level of perceived product quality also have critical values. One such variable, which emerged from the sensitivity analysis, was bug generating rate normal. Above a critical value of bug generating rate normal, the number of bugs per functionality becomes so overwhelmingly high that the participants fail to maintain an acceptable level of product quality and consequently the community dissolves.

The model did not show a wide variety of behaviors under different values of some variables. For example, running the model for different values of normal time to attract developers, normal time for developers to leave, and normal time to attract all users yielded different model behaviors, but the variety of behavior was not very wide. That finding indicated that the model might be improved by refining the equations involving these variables. This was noted as a potential future research opportunity.

Sensitivity runs also provided some important implications for the policy runs. Running the model under different values of refusal ratio and rejection ratio showed that there are optimal values for these variables that are high enough to improve the product quality substantially, but still low enough to sustain functionality and community growth. Increasing refusal ratio and rejection ratio above those values did not yield a considerable marginal improvement in product quality, but impeded product functionality and community growth. In fact, the community failed to sustain itself above a certain value of rejection ratio. The sensitivity analysis did not reveal such a critical value for refusal

ratio, since the highest possible value for refusal ratio could be 1 and even that value did not fail the community. However, it can be argued that a very high refusal ratio combined with low average developer participation or average developer productivity value could fail the community in sustaining itself.

5.5. Policy Runs

Policy runs involve simulating a model under a set of policy settings. Policy settings apply to parameters that can be determined by the policy or decision makers of the real system the model represents. Policy runs basically have two purposes. First, they are used to test whether the model exhibits a plausible variety of behavior under different policy options. In that sense they are close to sensitivity tests. Second, they are used to simulate the consequences of different policy options in order to evaluate and compare them. In this study, policy runs were used both to build confidence in the OSSD model, and to analyze a set of policy options before they were discussed with interview subjects in the empirical component of the study.

5.5.1. Higher Barriers to Entry

A set of policy runs was done on the model to see the consequences of different levels of barriers to entry to the community. The barriers to entry policies were conceptualized as a combination of different refusal ratios and initial number of developers, since a higher barrier to entry would mean a higher scrutiny level for accepting developer into the community. Table 5.1 summarizes the conditions of the three policy runs along with the base case conditions.

Table 5.1. Barriers to Entry Policy Settings

Run	Refusal Ratio	Initial Number of Developers
Base Case	0.10	7
Higher Barriers to Entry 1	0.35	5
Higher Barriers to Entry 2	0.60	3
Higher Barriers to Entry 3	0.80	1

Figure 5.97 shows that the number of developers started at a lower level and increased less under higher barriers to entry settings.

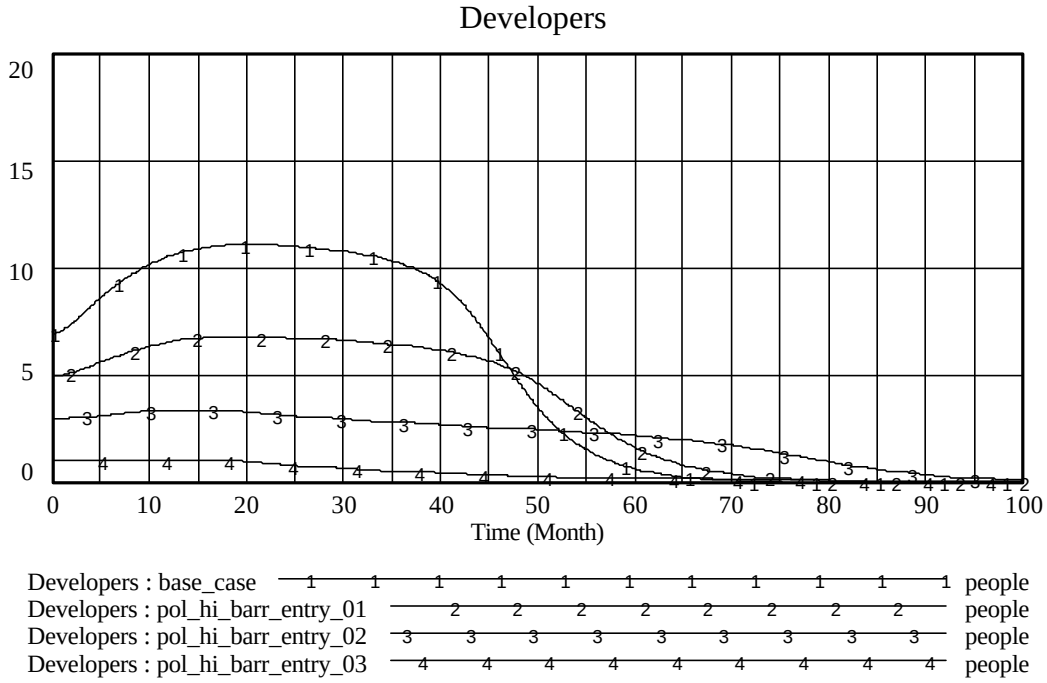


Figure 5.97. Developers under Different Barriers to Entry Policy Settings

Figure 5.98 shows that product functionality growth was slower under higher barriers to entry settings. In fact, under very high barriers to entry settings the community

failed to achieve a viable level of product functionality and to sustain itself in the long run. (See Figure 5.98 and Figure 5.99.)

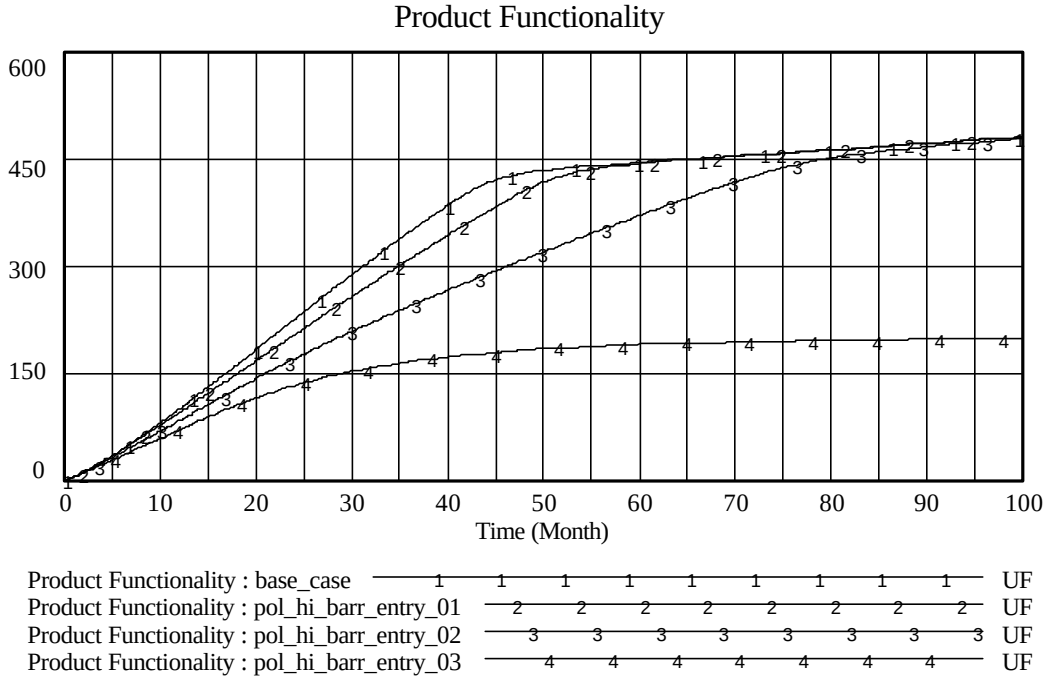


Figure 5.98. Product Functionality under Different Barriers to Entry Policy Settings

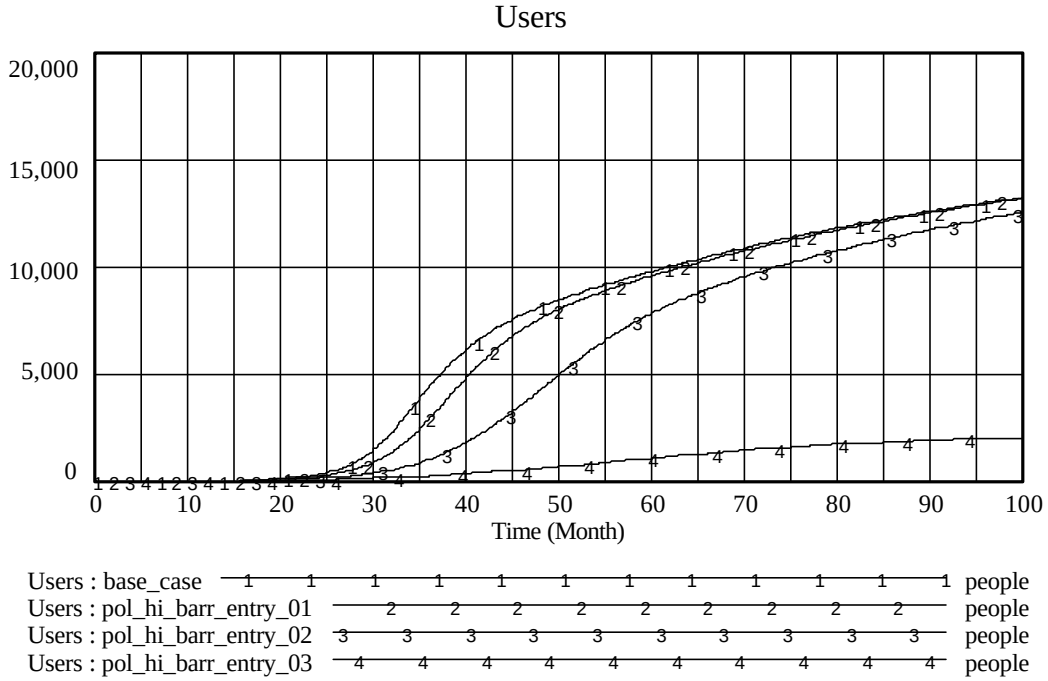


Figure 5.99. Users under Different Barriers to Entry Policy Settings

It is obvious from these figures that there has to be a trade-off in terms of product functionality and community growth whenever a higher barriers to entry policy is implemented to improve quality. The critical question then becomes what level of this policy would yield the most quality increase per decrease in the pace of functionality and community growth? Figure 5.100 shows that all three policy settings provided substantial increases in perceived product quality over the base case conditions. Furthermore, the differences between the levels of perceived product quality under the three policy settings were not large. Hence we may conclude that the first policy setting yields the greatest product quality payoff, while compromising relatively small in terms of functionality and community growth.

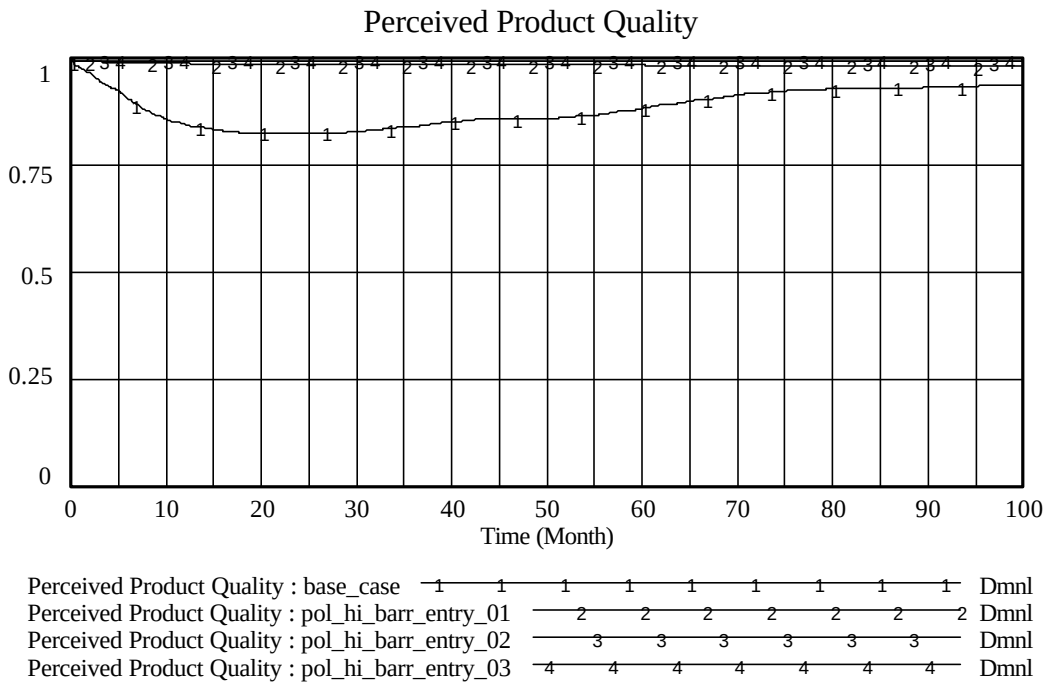


Figure 5.100. Perceived Product Quality under Different Barriers to Entry Policy Settings

Furthermore, the first policy setting provided a large increase in average developer talent. (See Figure 5.101.) While the two higher policy settings provided even higher developer talent levels, the marginal gains might not be deemed enough to justify the compromises in functionality and community growth.

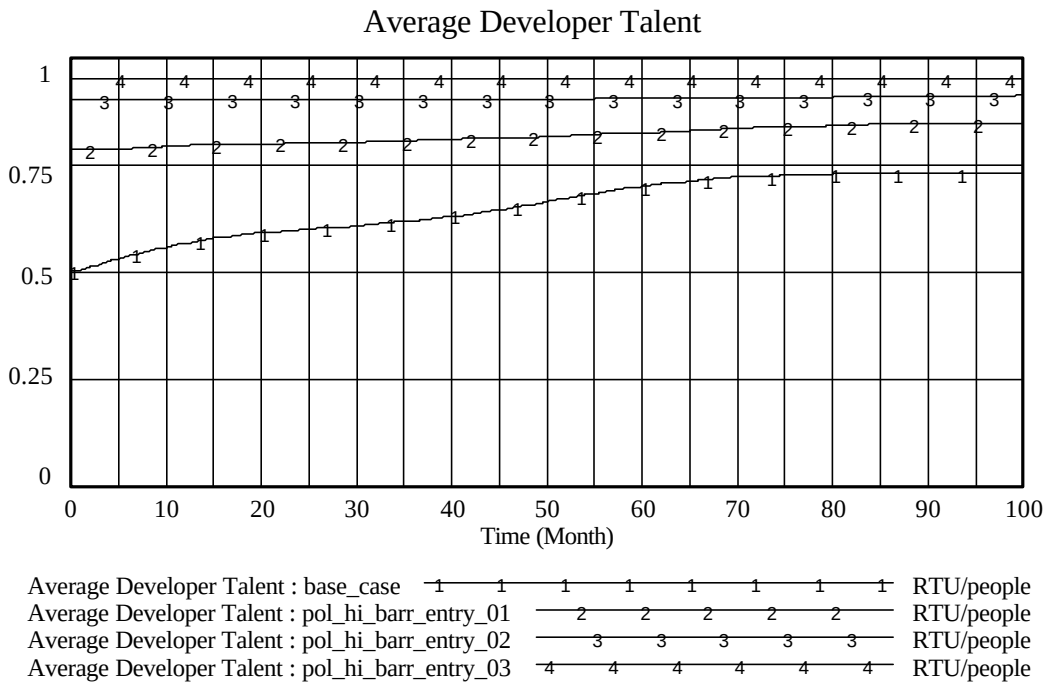


Figure 5.101. Average Developer Talent under Different Barriers to Entry Policy Settings

The barriers to entry policy option was introduced as the "Selecting New Inexperienced Authors" policy option to the subjects during the interview done with the members of the system dynamics K through 12 instructional material development community. (See Section 6.3.4.)

5.5.2. Higher Barriers to Contribution

"Barriers to contribution" was conceptualized as another important policy option for improving product quality, while maintaining functionality and community growth. The barriers to entry policy option was based on applying different rejection ratios to code produced by developers. Table 5.2 summarizes the conditions of the three policy runs along with the base case condition.

Table 5.2. Barriers to Contribution Policy Settings

Run	Rejection Ratio
Base Case	0.20
Higher Barriers to Contribution 1	0.40
Higher Barriers to Contribution 2	0.50
Higher Barriers to Contribution 3	0.60

The implications of the barriers to entry policy option were similar to the implications of the sensitivity runs done with different rejection ratios. Basically, higher barriers to contribution settings improved product quality by ensuring that the better portions of the code produced by developers were added to the overall code base, while low quality code was discarded. Hence higher barriers to contributions settings yielded initially better perceived product quality levels. However, the quality levels tended to decrease in the later stages of the project. (See Figure 5.102.) This was caused by the multiple effects of decreased developer participation. As discussed earlier in this chapter, the OSSD model assumes that developer participation decreases as rejection ratio increases due to decreased developer motivation. Decreased developer participation causes fewer developer hours available for debugging, which worsens the quality problem over time. Also, decreased participation limits developer talent growth, impeding the potential improvement in product quality.

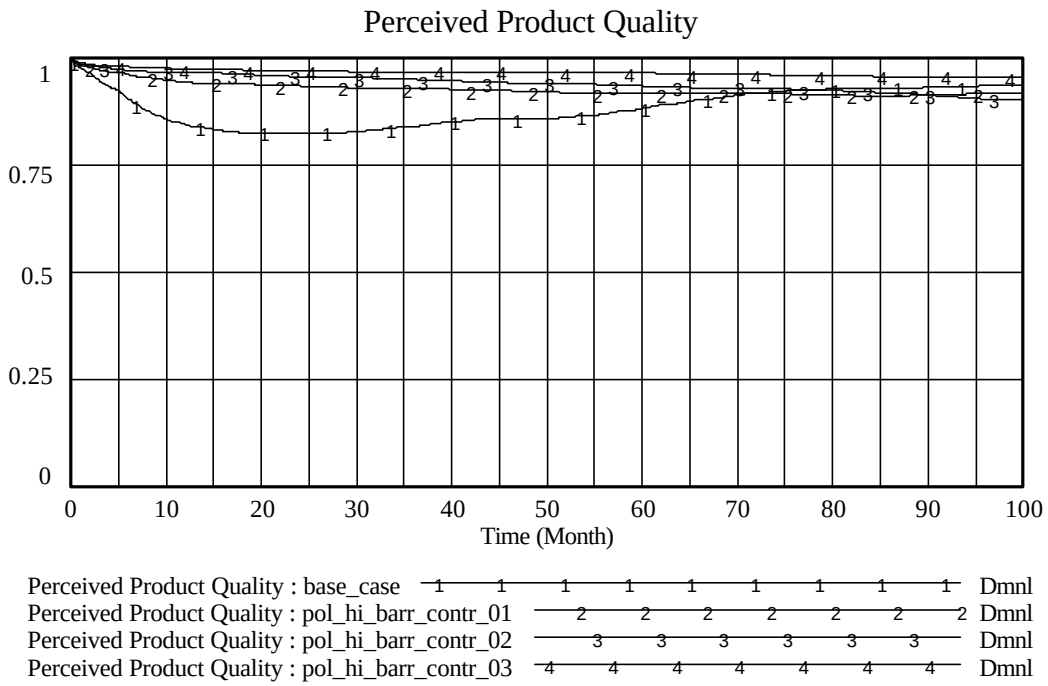


Figure 5.102. Perceived Product Quality under Different Barriers to Contribution Policy Settings

Another adverse effect of higher barriers to contribution through decreased developer participation is the decrease in the level of total production. (See Figure 5.103.) Product functionality grows more slowly as total production decreases. (See Figure 5.104.)

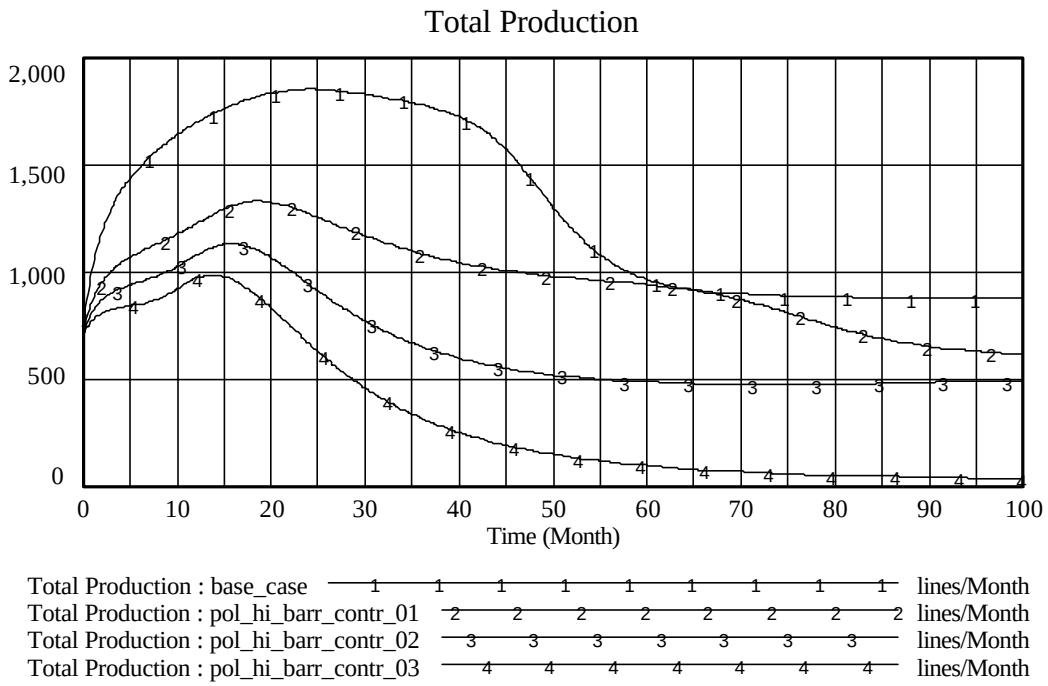


Figure 5.103. Total Production under Different Barriers to Contribution Policy Settings

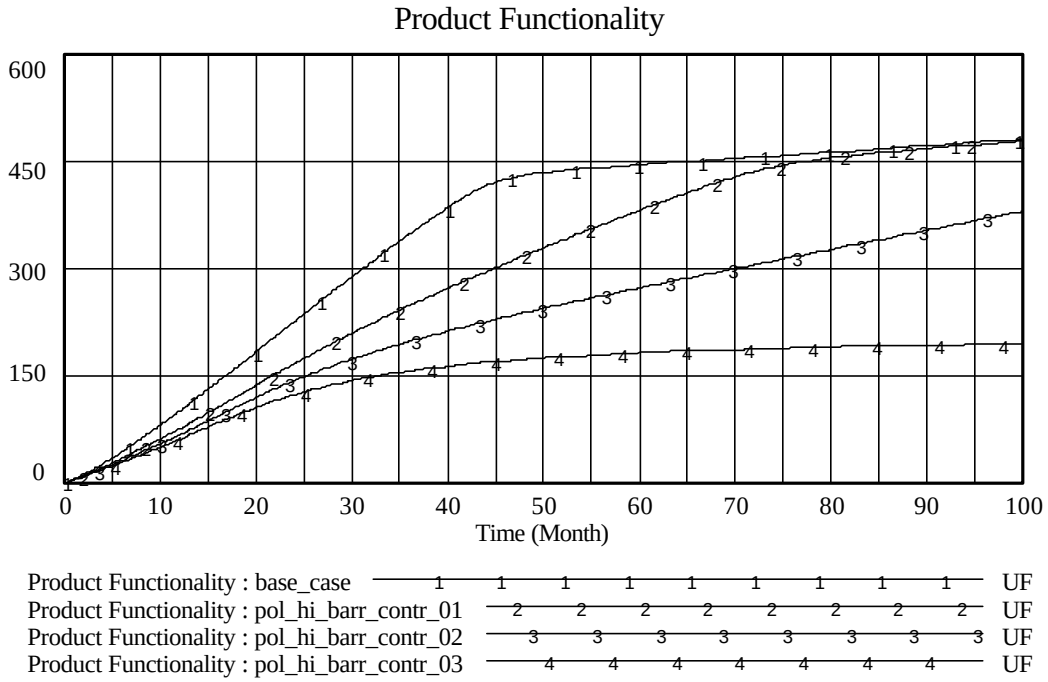


Figure 5.104. Product Functionality under Different Barriers to Contribution Policy Settings

User community growth follows functionality growth, and thus a slower functionality growth brings about a slower community growth. (See Figure 5.104.) Extremely high barriers to contribution may cause the community to fail to reach a viable product functionality level and to sustain itself in the long run, leading the community to extinction. Policy setting three, where the rejection rate was set to 0.60 is an example of such an extreme policy. Under that policy setting user and developer populations fail to reach sustainable levels (See Figure 5.105 and Figure 5.106.)

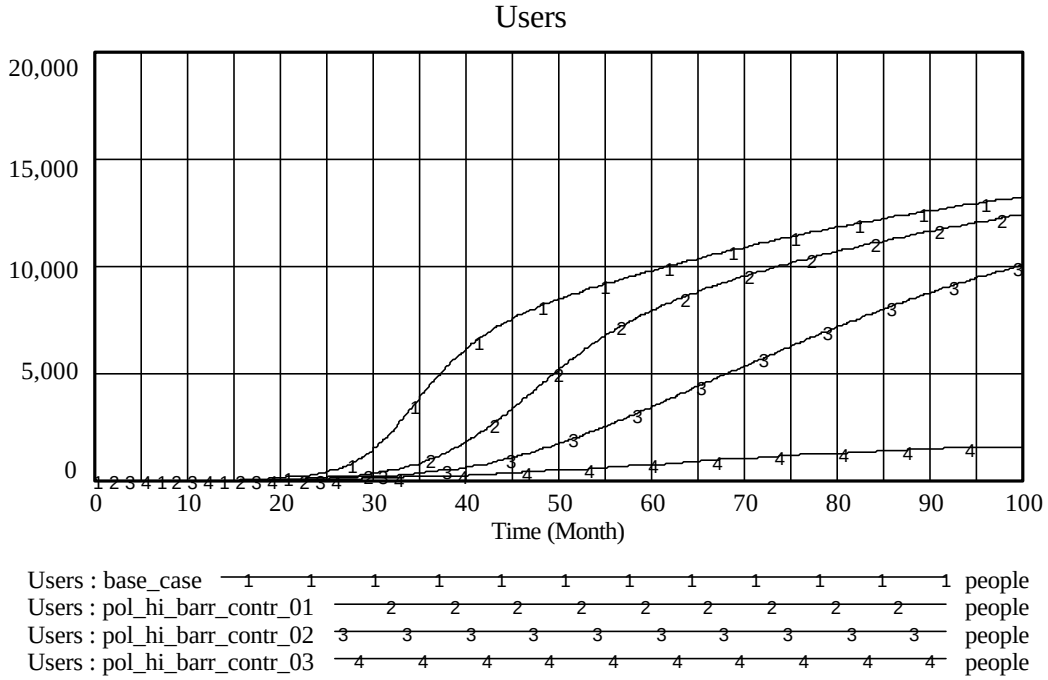


Figure 5.105. Users under Different Barriers to Contribution Policy Settings

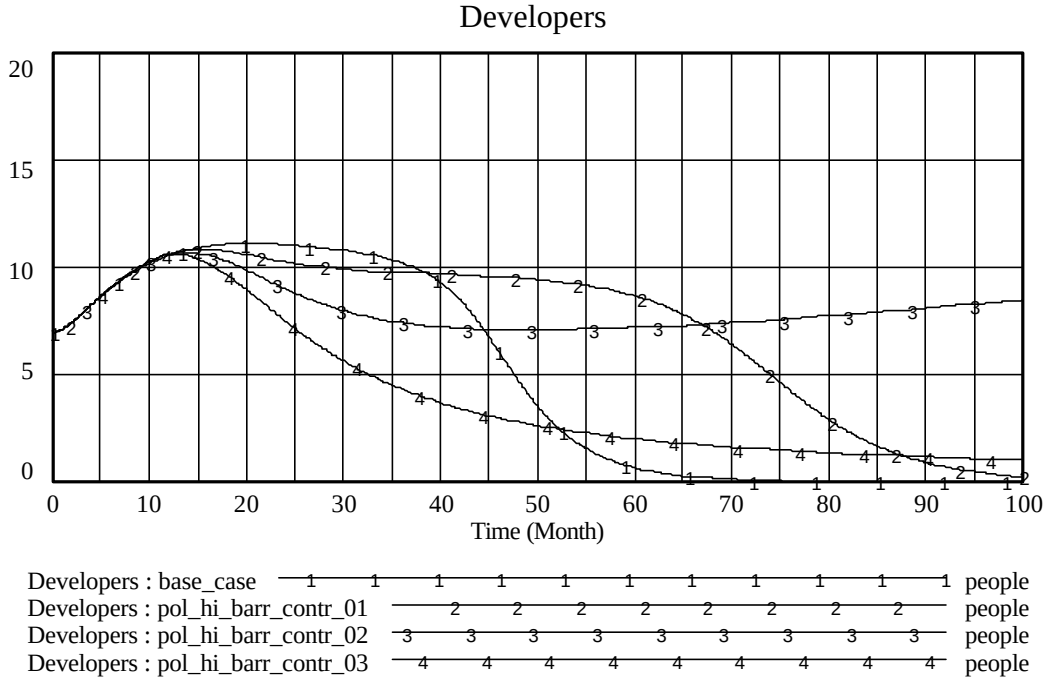


Figure 5.106. Developers under Different Barriers to Contribution Policy Settings

Once again, the fundamental question about the usefulness of this policy option was whether it provided a large enough quality improvement for a considerably small trade-off in terms of functionality and community growth. The answer to this question was not positive. The quality improvements in the policy runs were relatively small considering the substantial decrease in functionality and community growth rates. Furthermore, the quality improvement eroded after the initial improvement, thus rendering the policy totally unfavorable. A smaller, but nevertheless notable consequence of this policy option, which supported the unfavorable position, was the decreases in the rate of average developer talent growth due to decreased developer participation. (See Figure 5.107.)

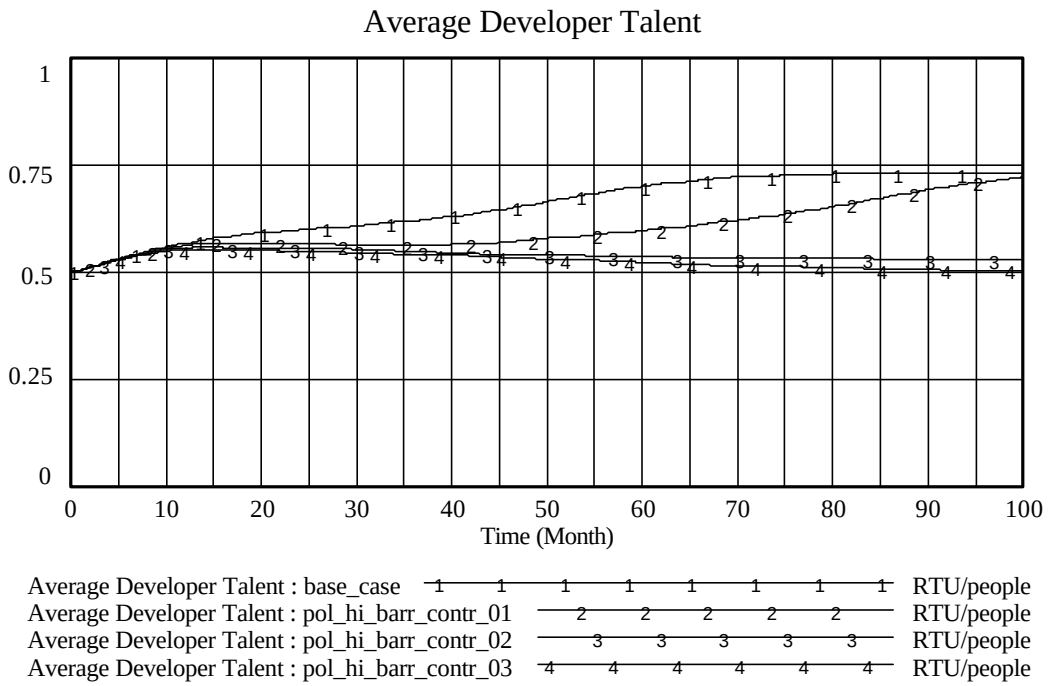


Figure 5.107. Average Developer Talent under Different Barriers to Contribution Policy Settings

Barriers to contribution was introduced as the “Filtering New Material” policy option to the interview subjects. (See Section 6.3.2.)

5.5.3. Higher Barriers to Entry and Contribution

Comparing the consequences of higher barriers to entry and higher barriers to contribution policy options revealed that higher barriers to entry option was the better choice between the two. Another policy run was done in order to test whether a combination of the two policies would yield better results than only the higher barriers to entry option. Table 5.3 summarizes the conditions of the two policy runs along with the base case condition.

Table 5.3. Barriers to Entry and Barriers to Entry and Contribution Policy Settings

Run	Refusal Ratio	Initial Number of Developers	Rejection Ratio
Base Case	0.10	7	0.20
Higher Barriers to Entry 1	0.35	5	0.20
Higher Barriers to Entry and Contribution 1	0.30	5	0.30

A comparison of the results of the two policy options revealed that the barriers to entry option performed better than the combination policy option in all main criteria. The barriers to entry option yielded a faster production growth (See Figure 5.108), which led to a faster user community growth (See Figure 5.109). This policy option also yielded a higher average developer talent (Figure 5.110.) and a better product quality, although the difference in product quality between the two policy options was very small. (See Figure 5. 111.)

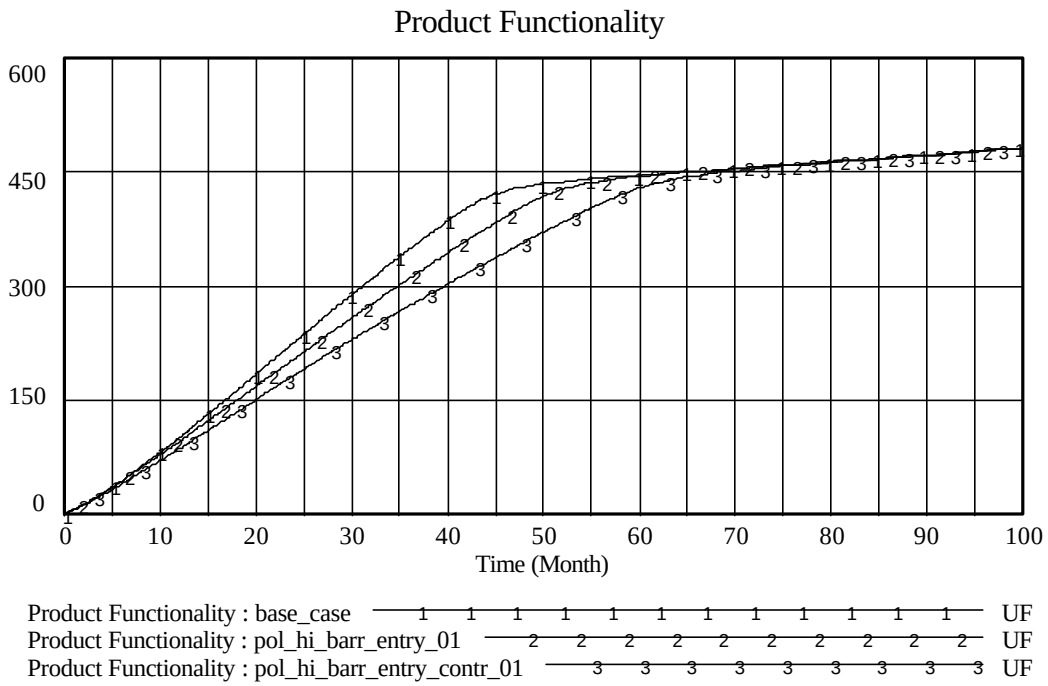


Figure 5.108. Product Functionality under Barriers to Entry and Combination Policy Settings

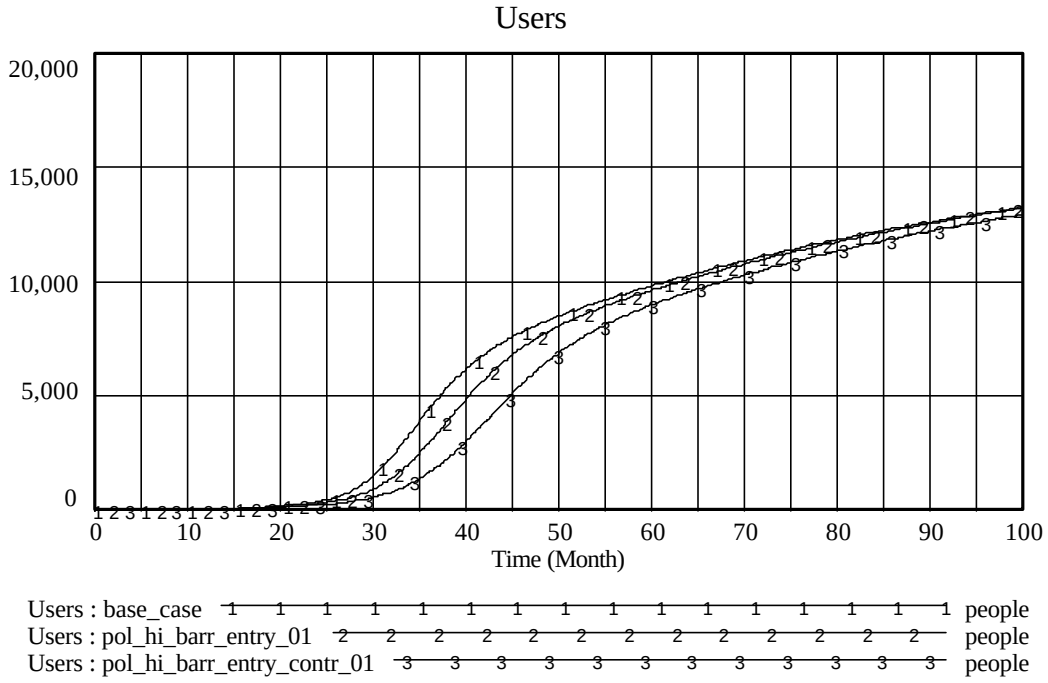


Figure 5.109. Users under Barriers to Entry and Combination Policy Settings

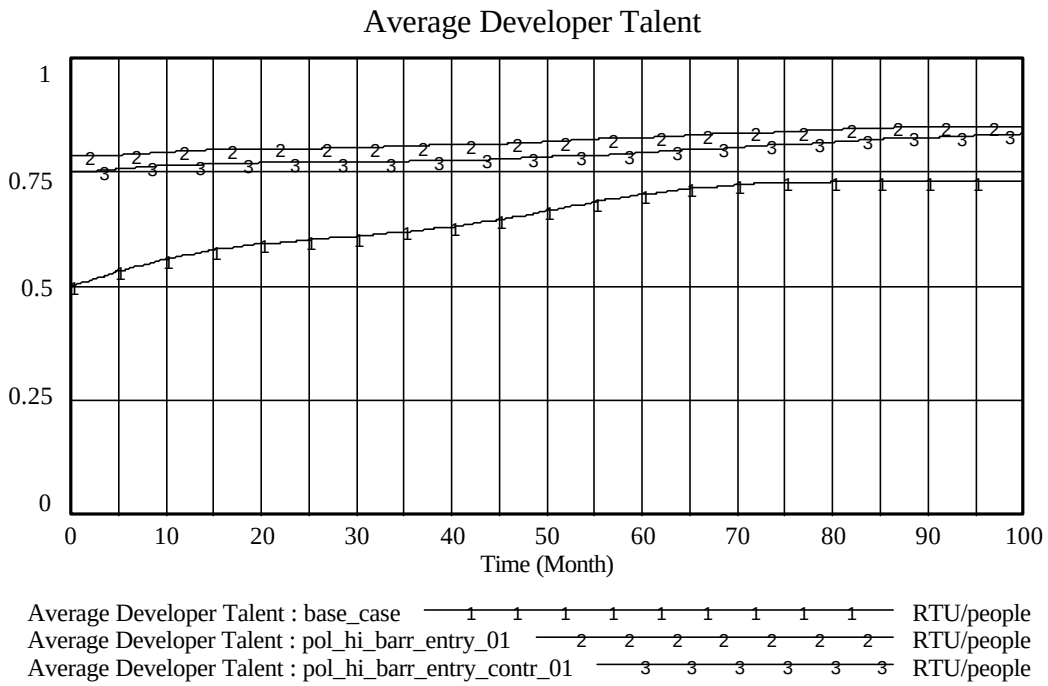


Figure 5.110. Average Developer Talent under Barriers to Entry and Combination Policy Settings

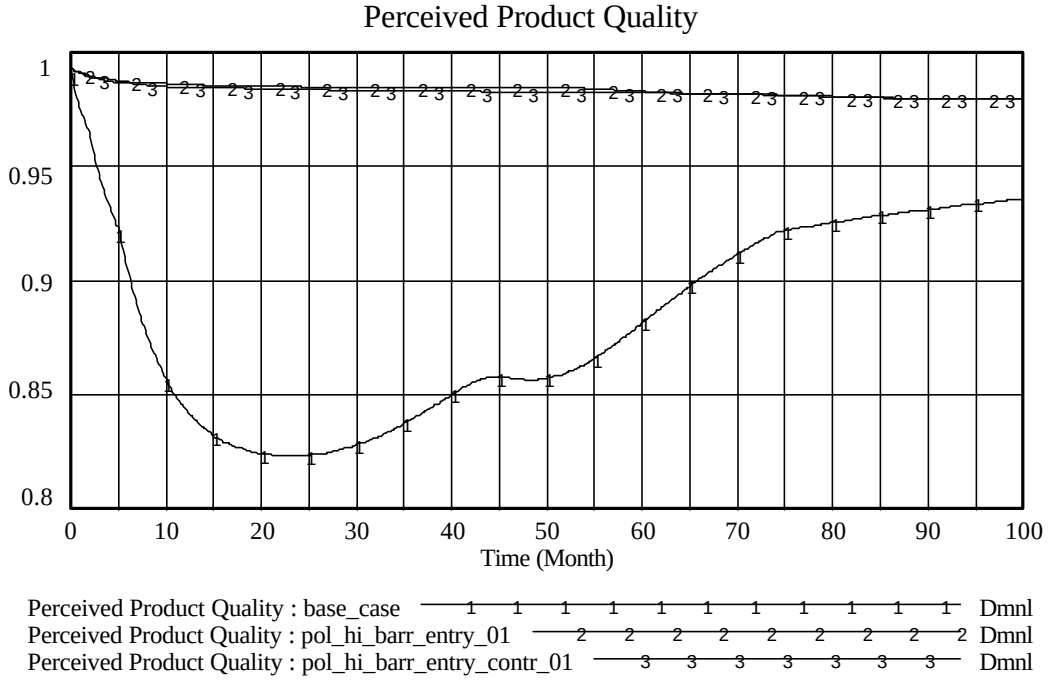


Figure 5.111. Perceived Product Quality under Barriers to Entry and Combination Policy Settings

5.5.4. Higher Debugging Emphasis

Another important policy option applied to the model was higher debugging emphasis. This option is conceptualized as increases in the relative pressures for bug detection and bug fixing within the community. As these relative pressures increase, the same number of known and/or unknown bugs generate relatively higher amounts of developer and leader time allocated for bug detection and bug fixing activities. Table 5.4 summarizes the conditions of the three policy runs as well as the base case conditions.

Table 5.4. Higher Debugging Emphasis Policy Settings

Run	**Pressure for Bug Detection**	**Pressure for Bug Fixing**
Base Case	Base Case Level*1	Base Case Level*1
Higher Debugging Emphasis 1	Base Case Level*5	Base Case Level*5
Higher Debugging Emphasis 2	Base Case Level*8	Base Case Level*8
Higher Debugging Emphasis 3	Base Case Level*10	Base Case Level*10

As expected, higher debugging emphasis yielded higher levels of perceived product quality. (See Figure 5.112) However, the marginal quality improvement by the second and third level policy settings did not yield as large a difference as the first policy setting yielded over the base case conditions. (See Figure 5.113.) This was noted as a potential limiting factor on the policy level, in case of a large functionality or community growth trade-off for higher levels of the policy.

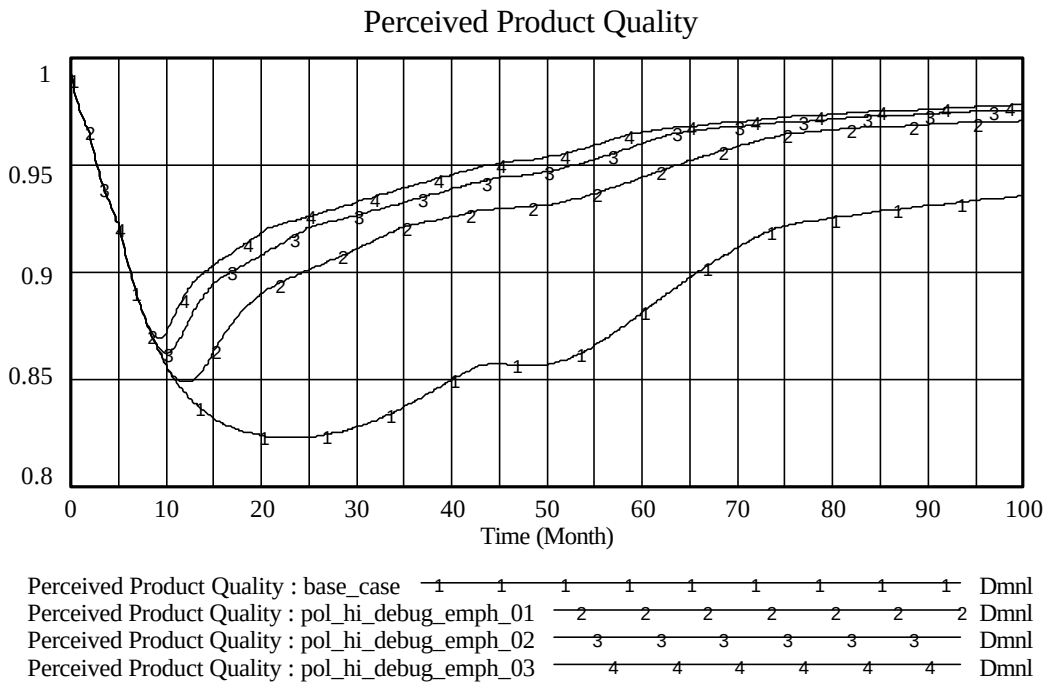


Figure 5.112. Perceived Product Quality under Different Debugging Emphasis Policy Settings

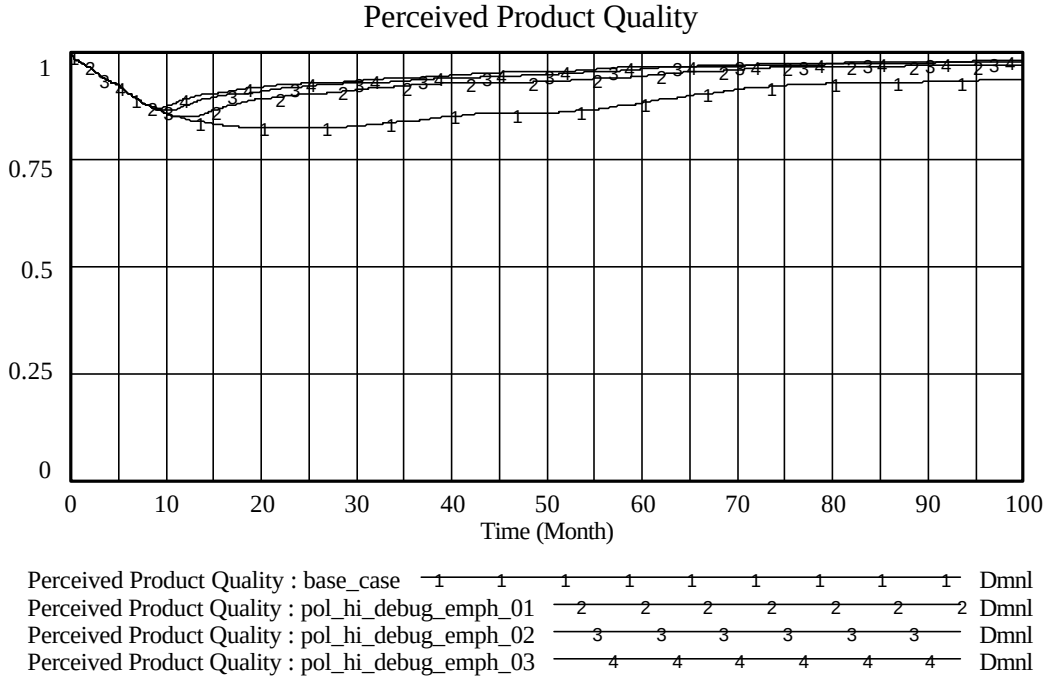


Figure 5.113. Perceived Product Quality under Different Debugging Emphasis Policy Settings

Further analysis revealed that substantially higher debugging emphasis did not necessitate large trade-offs in terms of product functionality and community growth or developer talent improvement. (See Figure 5.114 through Figure 5.116.) Therefore, it was concluded that higher levels of debugging emphasis would be favorable until further increases in the policy level yielded a negligibly small quality improvement.

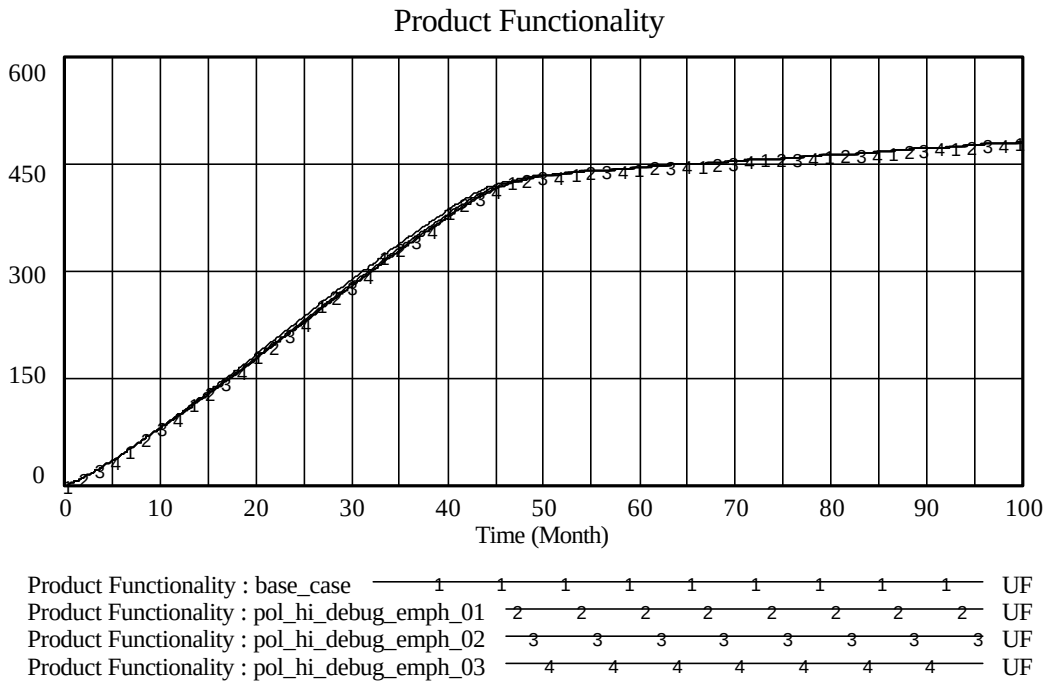


Figure 5.114. Product Functionality under Different Debugging Emphasis Policy Settings

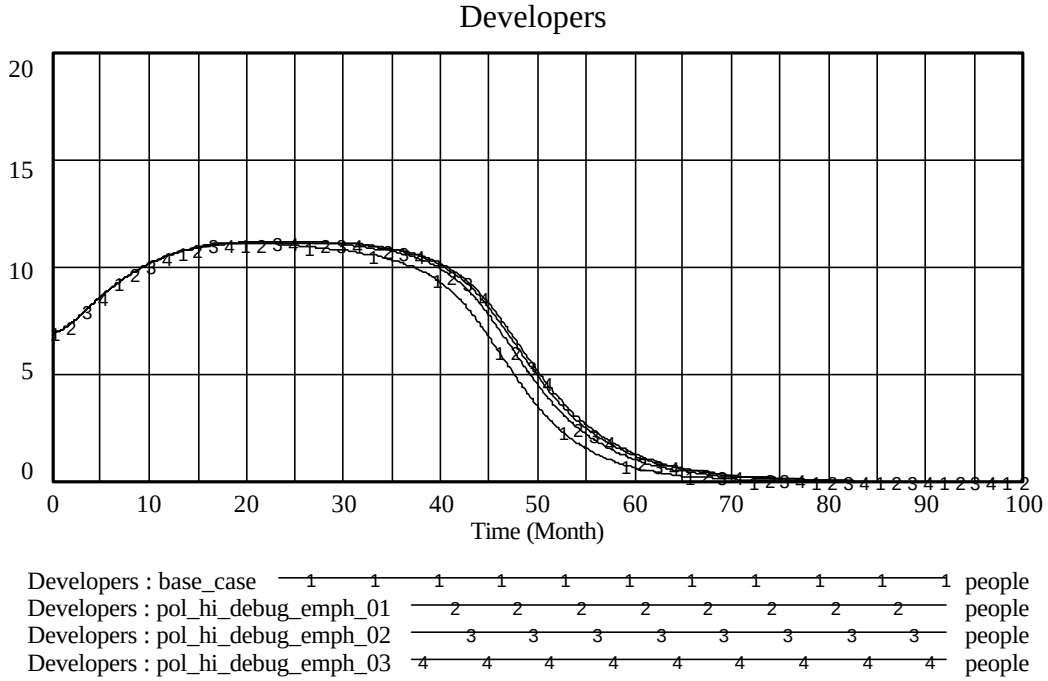


Figure 5.115. Developers under Different Debugging Emphasis Policy Settings

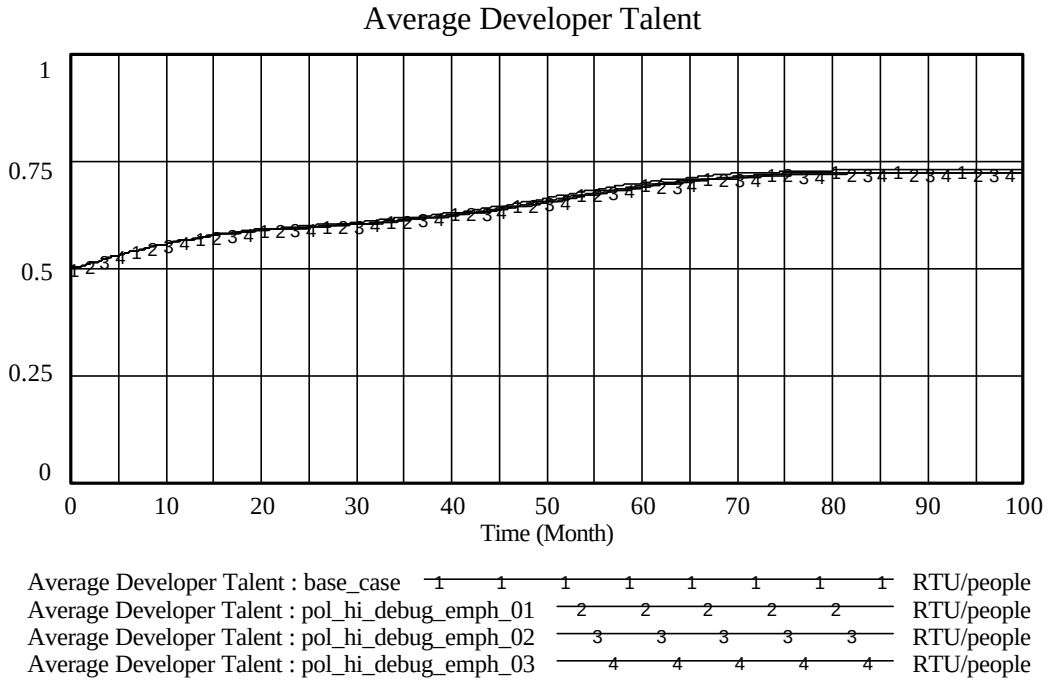


Figure 5.116. Average Developers Talent under Different Debugging Emphasis Policy Settings

Higher debugging emphasis option was introduced as the "Reviewing and Editing Existing Material" policy option to the interview subjects. (See Section 6.3.3.)

5.5.5. Higher Coaching Emphasis

Higher coaching emphasis was another policy option applied to the model. This option is conceptualized as increased levels of pressure for talent building. As this pressure increases, the same level of average developer talent generates a relatively higher amount of developer and leader time allocated for coaching. Table 5.5 summarizes the conditions of the three policy runs as well as the base case conditions.

Table 5.5. Higher Coaching Emphasis Policy Settings

Run	**Pressure for Talent Building**
Base Case	Base Case Level*1
Higher Coaching Emphasis 1	Base Case Level*2
Higher Coaching Emphasis 2	Base Case Level*3
Higher Coaching Emphasis 3	Base Case Level*4

Average developer talent increased faster and reached higher equilibriums for higher coaching emphasis levels. (See Figure 5.117.) The decrease in the pace of product functionality growth was not substantial for higher policy settings. (See Figure 5.118.)

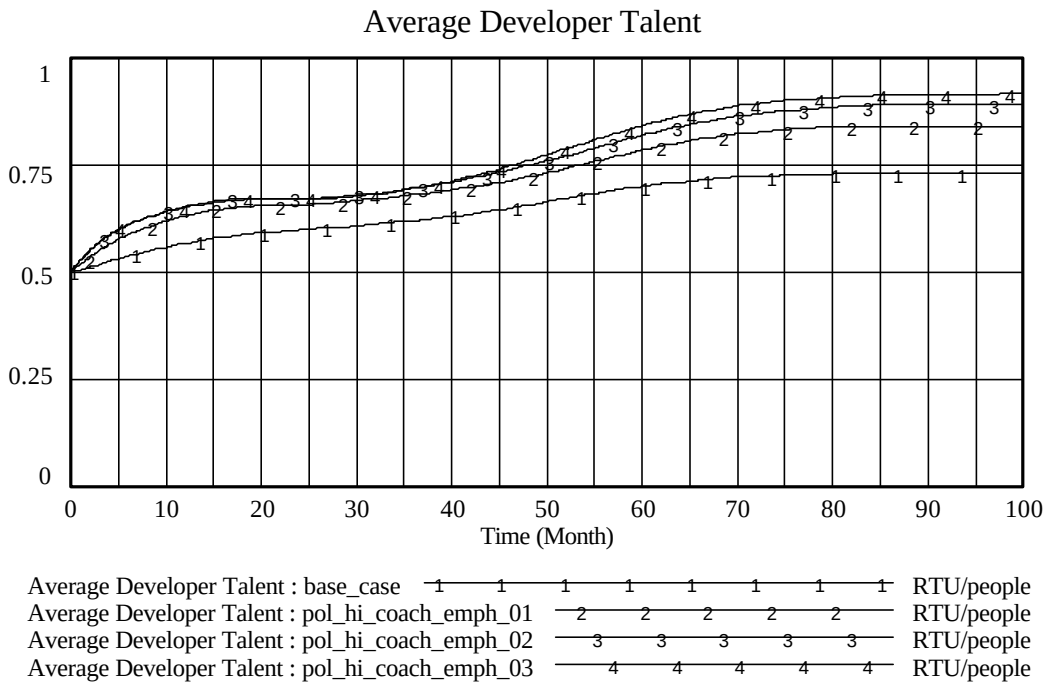


Figure 5.117. Average Developers Talent under Different Coaching Emphasis Policy Settings

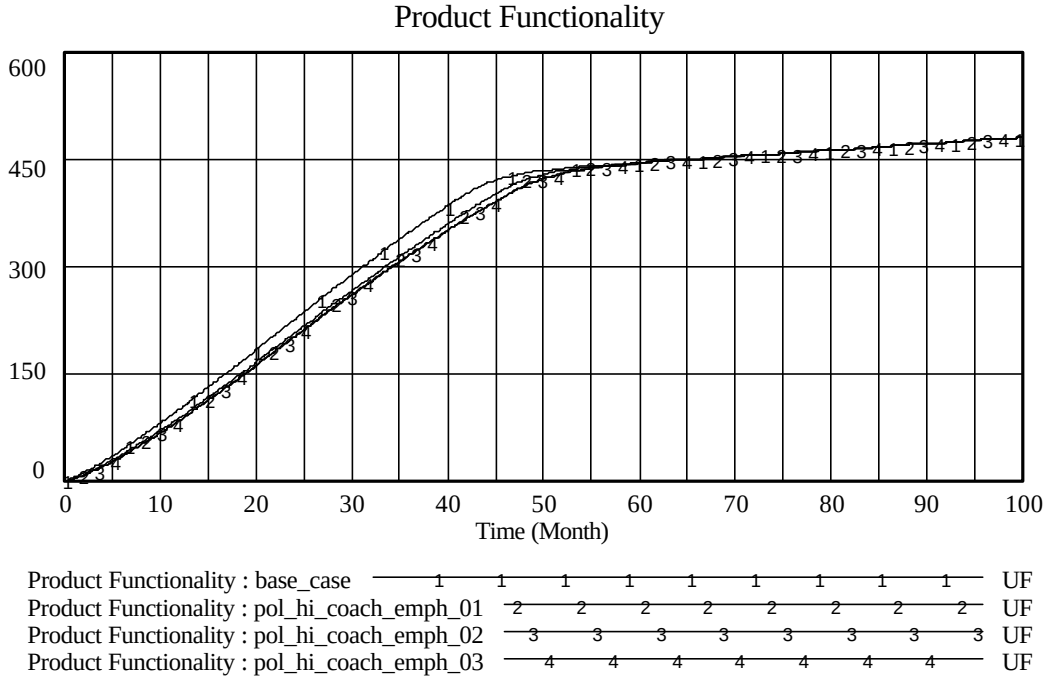


Figure 5.118. Product Functionality under Different Coaching Emphasis Policy Settings

However, the improvements in perceived product quality under higher coaching emphasis policy options were not satisfactory. (See Figure 5.119.)

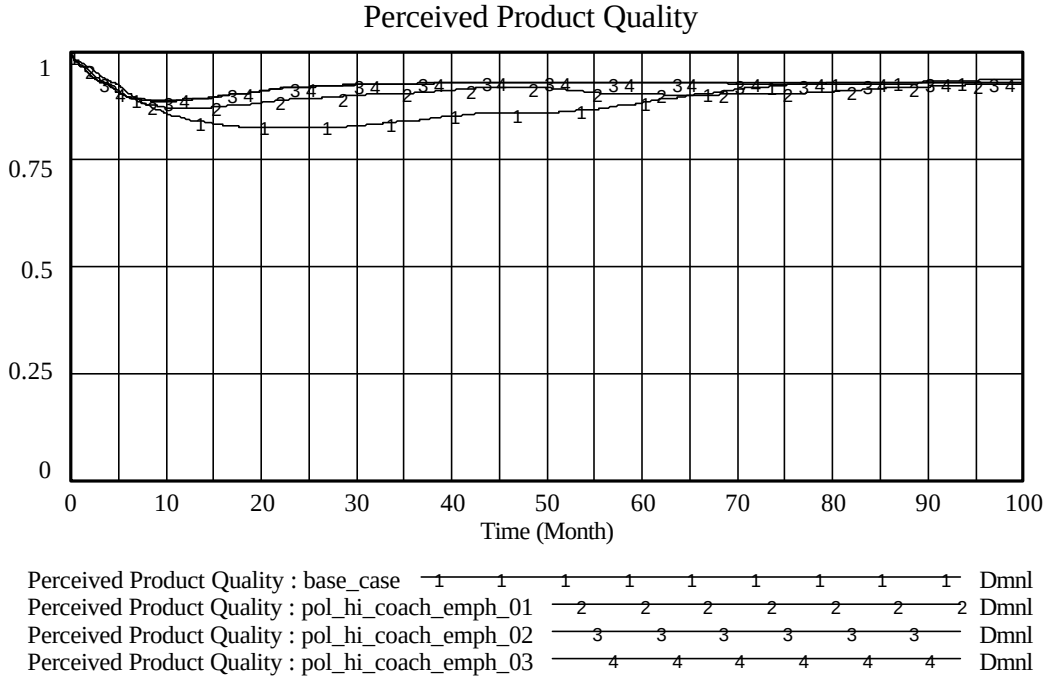


Figure 5.119. Perceived Product Quality under Different Coaching Emphasis Policy Settings

Further analysis identified the lower levels of pressures for bug detection and bug fixing as the causes behind the lack of sustained quality improvement under higher coaching emphasis policy options. Pressures for bug detection and bug fixing remained considerably lower under higher coaching policy conditions compared to the figures under the base case conditions. That was due to the initially fewer number of bugs in the product under higher coaching conditions. (See Figure 5.120 and Figure 5.121.) By the time debugging came into focus, the developers had left the community, and the lack of manpower slowed down the quality improvement process. (See Figure 5.122.)

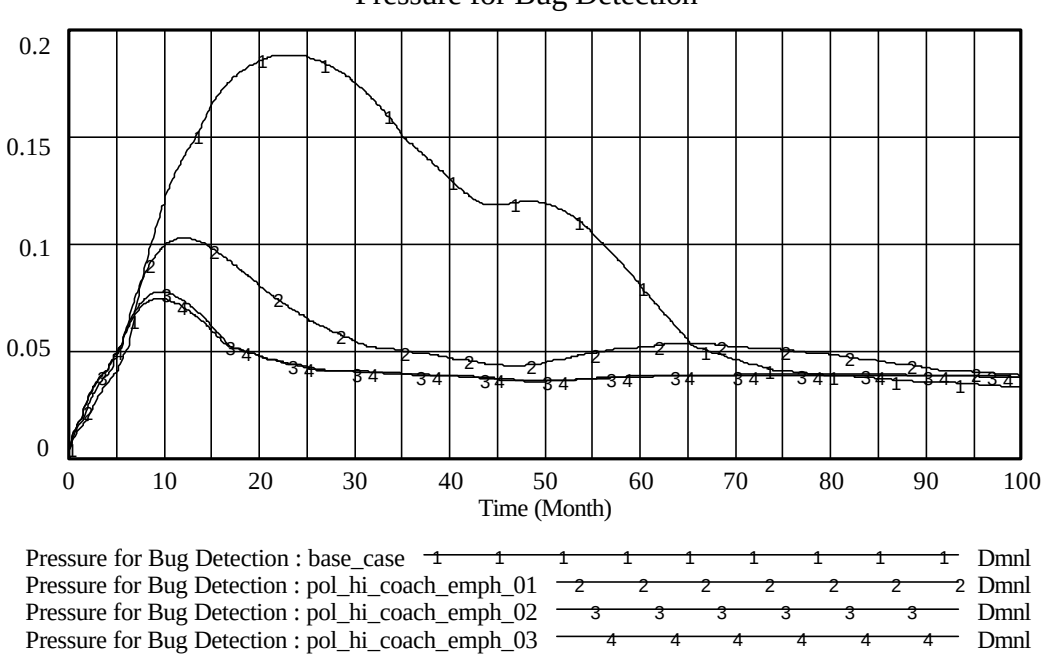


Figure 5.120. Pressure for Bug Detection under Different Coaching Emphasis Policy Settings

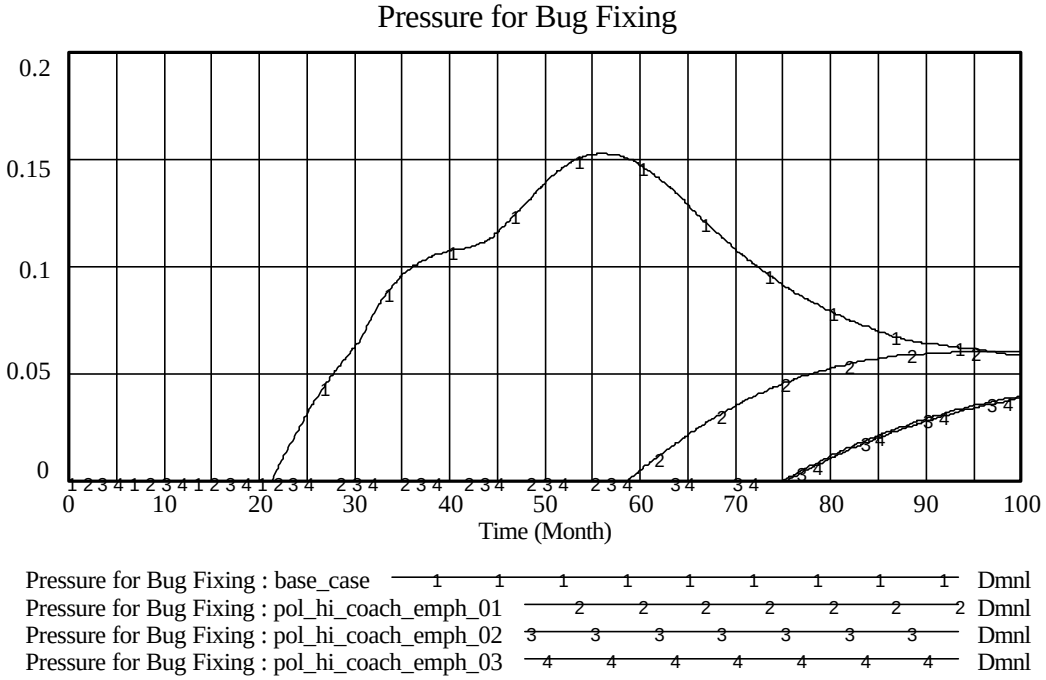


Figure 5.121. Pressure for Bug Fixing under Different Coaching Emphasis Policy Settings

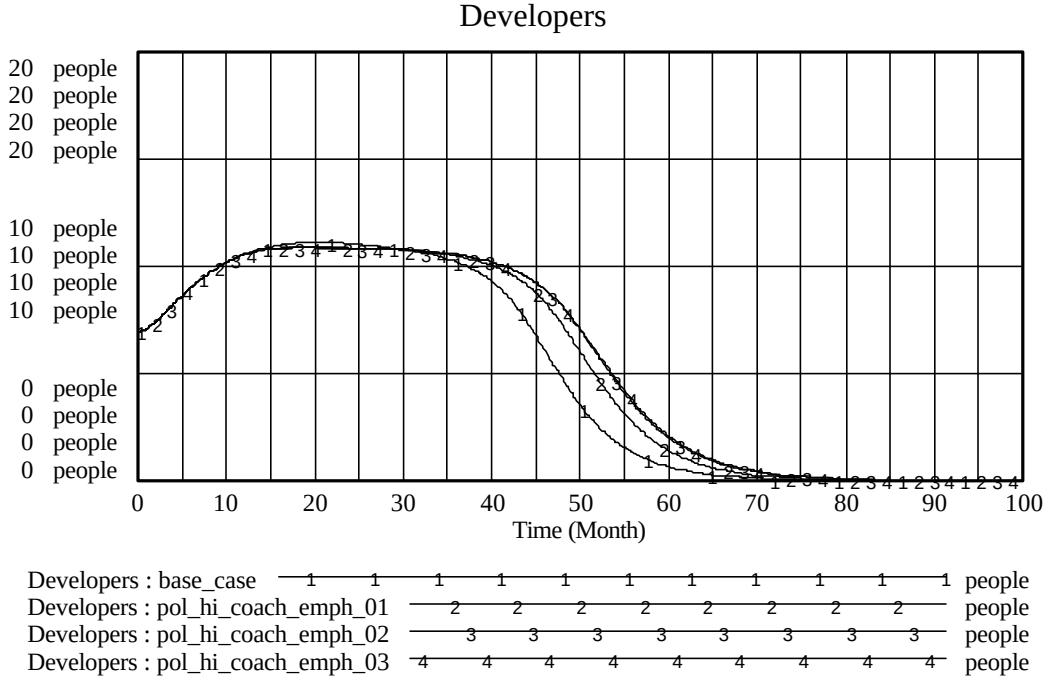


Figure 5.122. Developers under Different Coaching Emphasis Policy Settings

The higher coaching emphasis option was introduced as the “Coaching Existing Inexperienced Authors” policy option to the interview subjects. (See Section 6.3.5.)

5.5.6. Higher Debugging and Coaching Emphases

Another policy option was conceptualized after identifying the lack of debugging pressures as the main reason behind the unsatisfactory quality improvement under higher coaching emphasis policy option. The new policy option combined the higher debugging emphasis and higher coaching emphasis options. Table 5.6 summarizes the policy settings for the cases that were compared.

Table 5.6. Higher Debugging Emphasis, Higher Coaching Emphasis, and Higher Debugging and Coaching Emphases Policy Settings

Run	Pressure for Bug Detection	Pressure for Bug Fixing	Pressure for Talent Building
Base Case	Base Case Level*1	Base Case Level*1	Base Case Level*1
Higher Debugging Emphasis 3	Base Case Level*10	Base Case Level*10	Base Case Level*1
Higher Coaching Emphasis 3	Base Case Level*1	Base Case Level*1	Base Case Level*4
Higher Debugging and Coaching Emphases 1	Base Case Level*10	Base Case Level*10	Base Case Level*4

The combination policy option yielded a higher and more sustained quality improvement than those yielded by both the higher debugging emphasis and the higher coaching emphasis options. (See Figure 5.123.) Also, the increase in average developer talent under the combination policy was much higher than that under the higher debugging emphasis option, and very close to that under the higher coaching emphasis option. (See Figure 5.124.) Furthermore, the decrease in the pace of both product functionality growth and community growth (in terms of developers and users) was very small under the combination policy option. (See Figures 5.125 through 5.127.)

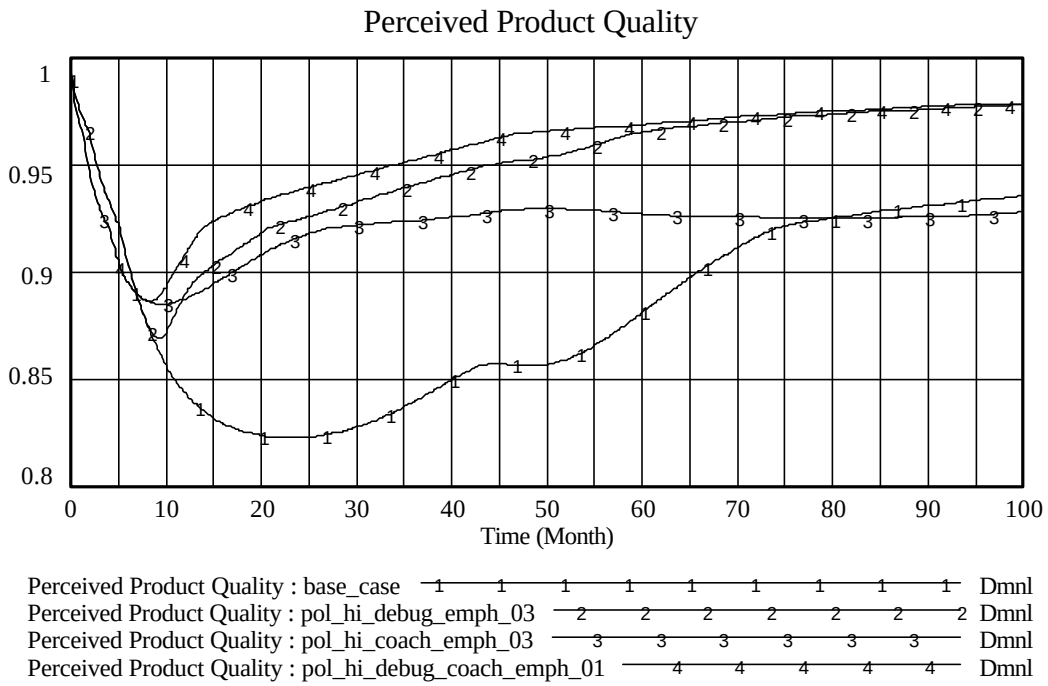


Figure 5.123. Perceived Product Quality under Higher Debugging Emphasis, Higher Coaching Emphasis, and Combination Policy Settings

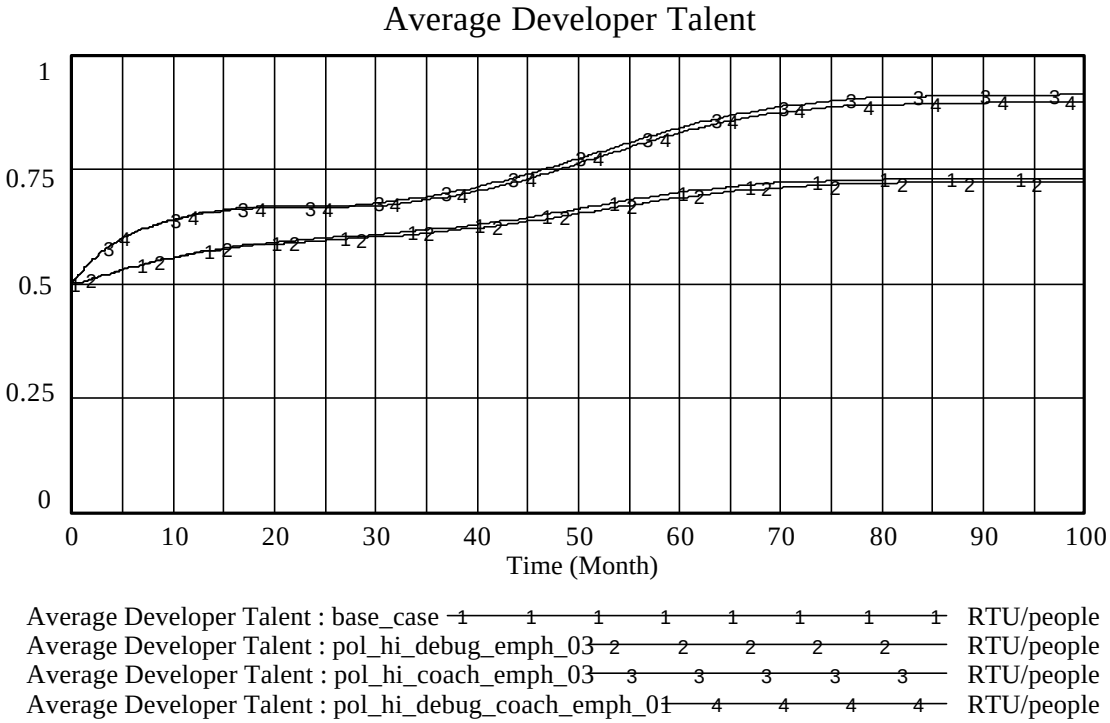


Figure 5.124. Average Developer Talent under Higher Debugging Emphasis, Higher Coaching Emphasis, and Combination Policy Settings

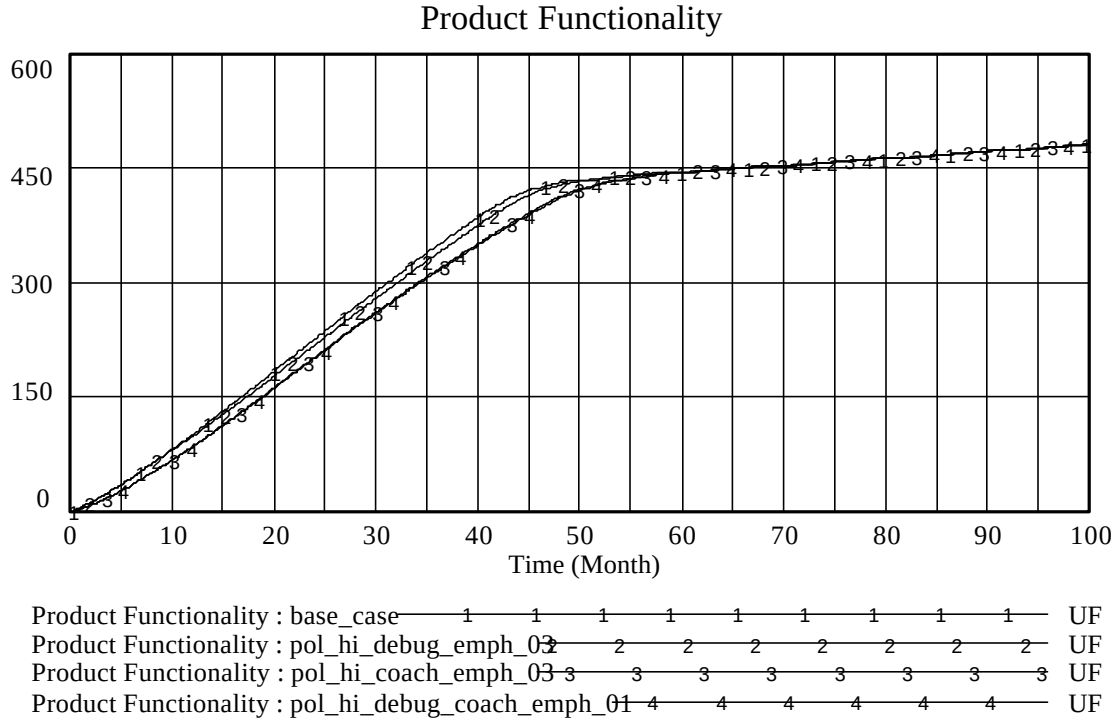


Figure 5.125. Product Functionality under Higher Debugging Emphasis, Higher Coaching Emphasis, and Combination Policy Settings

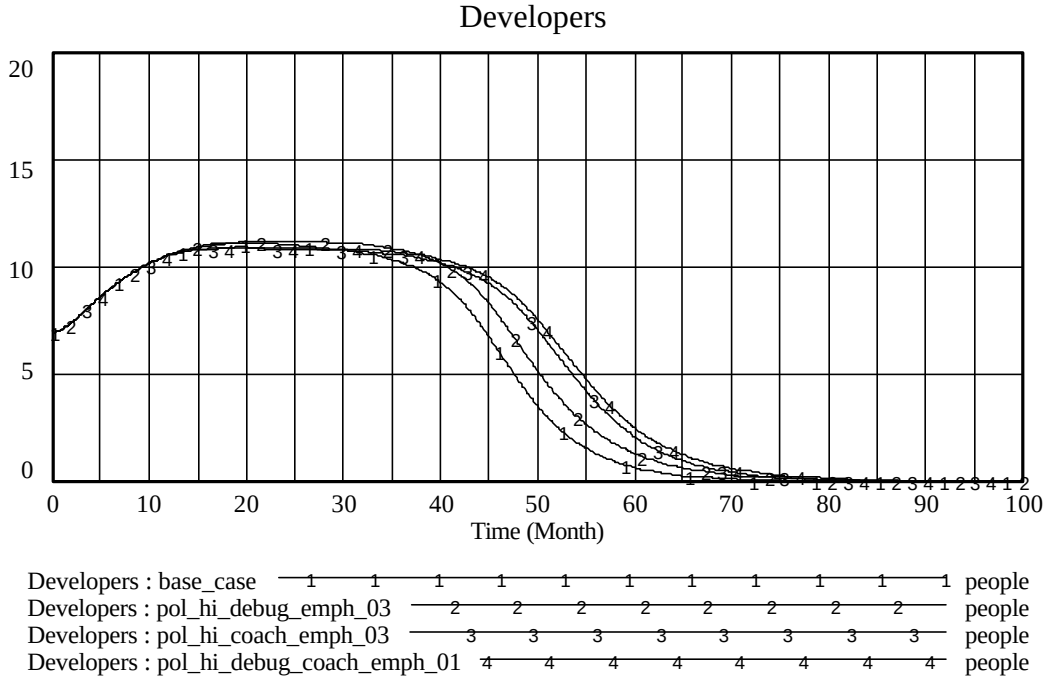


Figure 5.126. Developers under Higher Debugging Emphasis, Higher Coaching Emphasis, and Combination Policy Settings

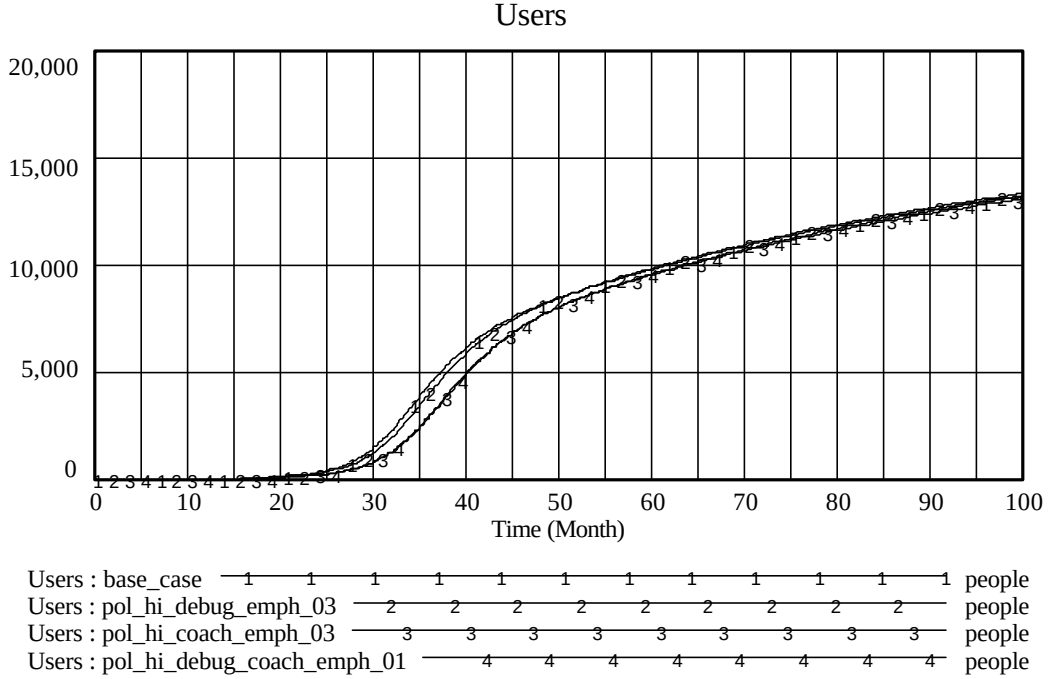


Figure 5.127. Users under Higher Debugging Emphasis, Higher Coaching Emphasis, and Combination Policy Settings

5.5.7. Higher Barriers to Entry, and Higher Debugging and Coaching Emphases

Higher barriers to entry and a combination of higher debugging and higher coaching emphases were found to be the two best policy options during the earlier policy runs. An overall combination policy run combining these two options was also tested on the model. Table 5.7 summarizes the policy settings for the compared runs.

Table 5.7. Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Combination Policy Settings

Run	Refusal Ratio	Initial Number of Developers	Pressure for Bug Detection	Pressure for Bug Fixing	Pressure for Talent Building
Base Case	0.10	7	Base Case Level*1	Base Case Level*1	Base Case Level*1
Higher Barriers to Entry 1	0.35	5	Base Case Level*1	Base Case Level*1	Base Case Level*1
Higher Debugging and Coaching Emphases 1	0.10	7	Base Case Level*10	Base Case Level*10	Base Case Level*4
Higher Barriers to Entry and Higher Debugging and Coaching Emphases 1	0.35	5	Base Case Level*10	Base Case Level*10	Base Case Level*4

The analysis of the model behaviors under these three policy runs demonstrated that the overall combination policy yielded higher improvements in both perceived product quality and average developer talent than the two alternatives. (See Figure 5.128 and Figure 5.129.) However, the product functionality growth and community growth became much slower under the overall combination policy conditions. (See Figures 5.130 through 5.132.)

Another finding of the comparison of these three policy options was that the specific higher barriers to entry, and higher debugging and coaching emphases policy

settings caused almost the same amount of loss in product functionality and community growth. However, while higher barriers to entry yielded a faster and larger quality improvement, higher debugging and coaching emphases yielded a higher average developer talent in the long run. Furthermore, higher debugging and coaching emphases achieved the same quality level with higher barriers to entry toward the end of the simulation horizon of 100 months.

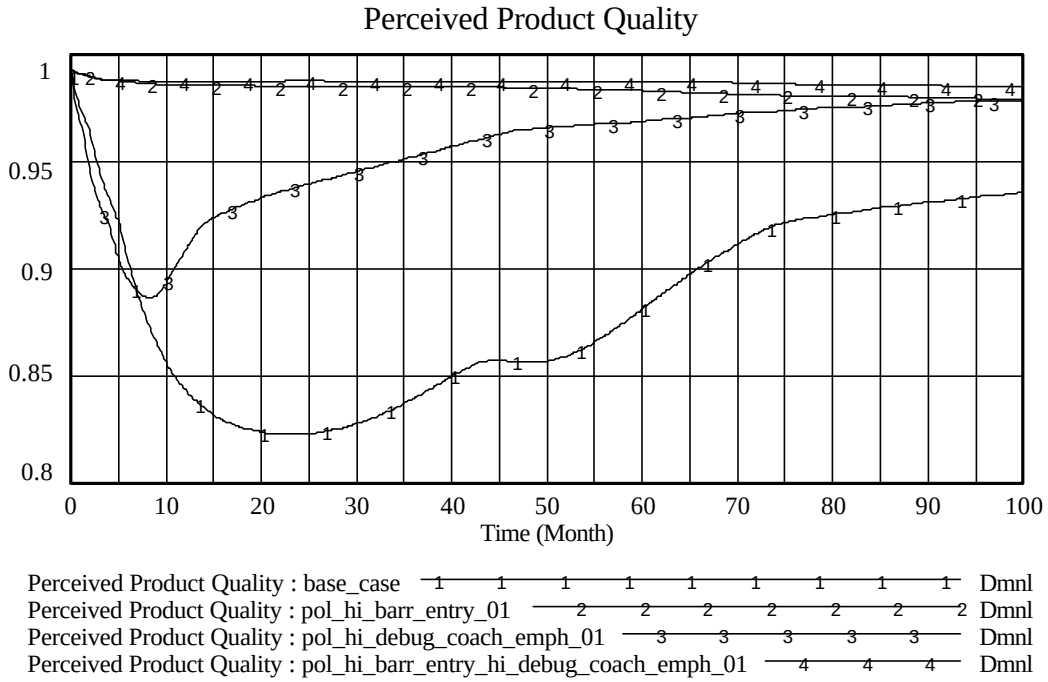


Figure 5.128. Perceived Product Quality under Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Overall Combination Policy Settings

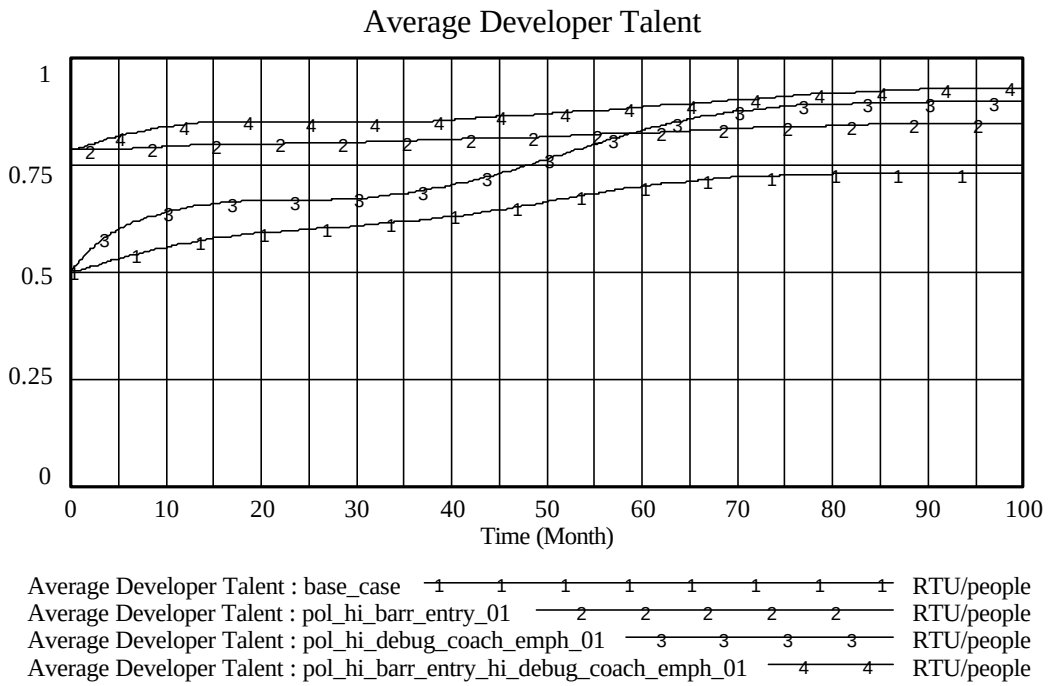


Figure 5.129. Average Developer Talent under Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Overall Combination Policy Settings

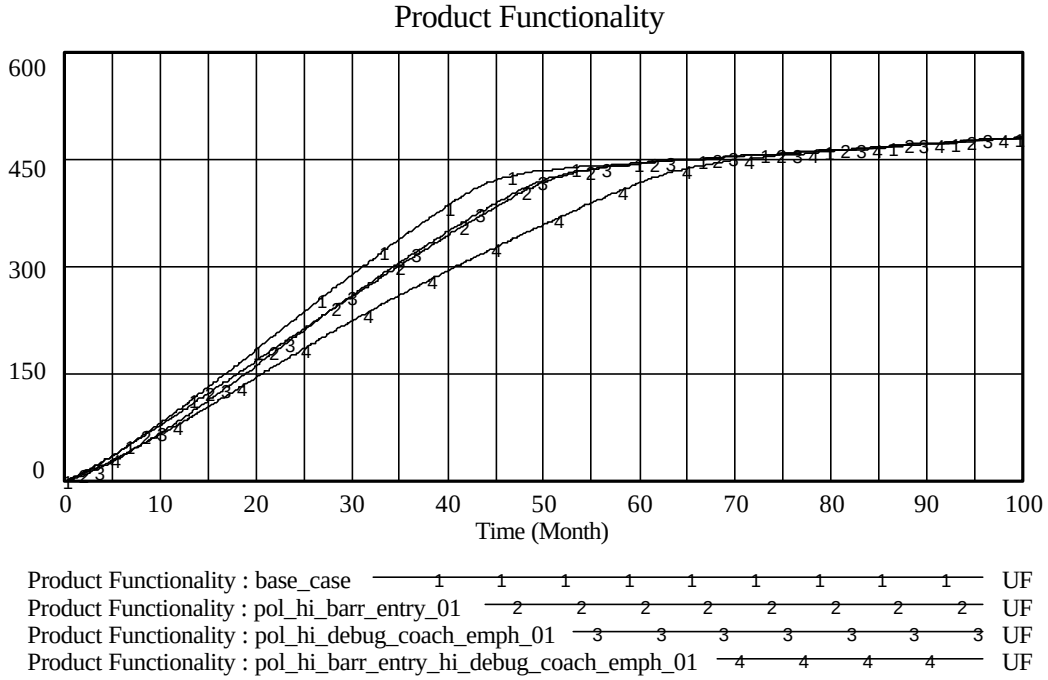


Figure 5.130. Product Functionality under Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Overall Combination Policy Settings

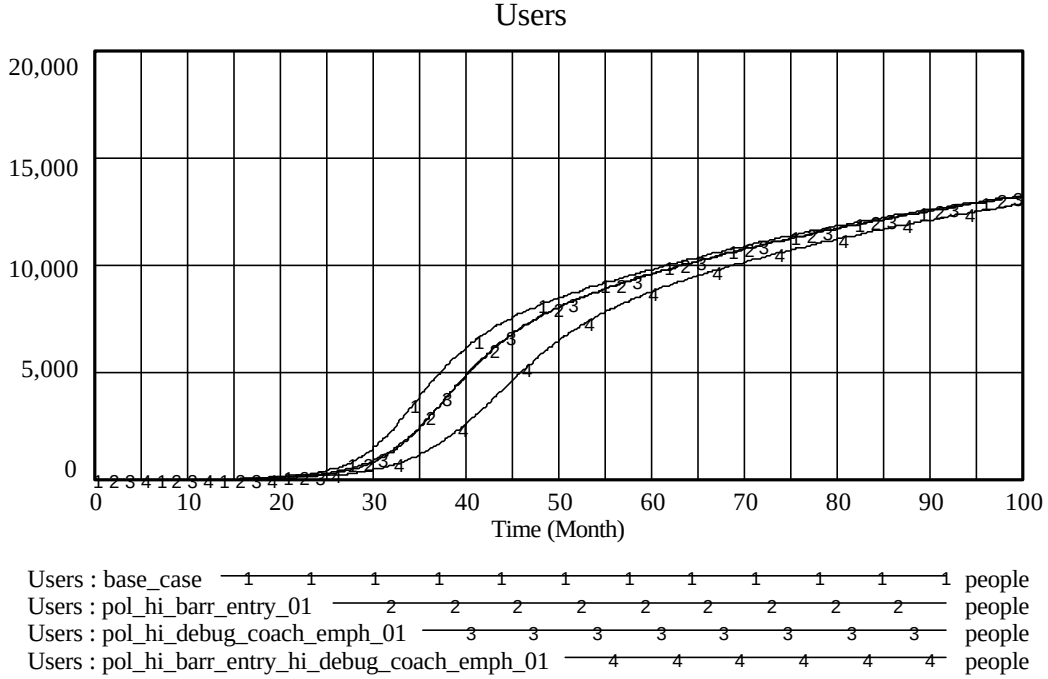


Figure 5.131. Users under Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Overall Combination Policy Settings

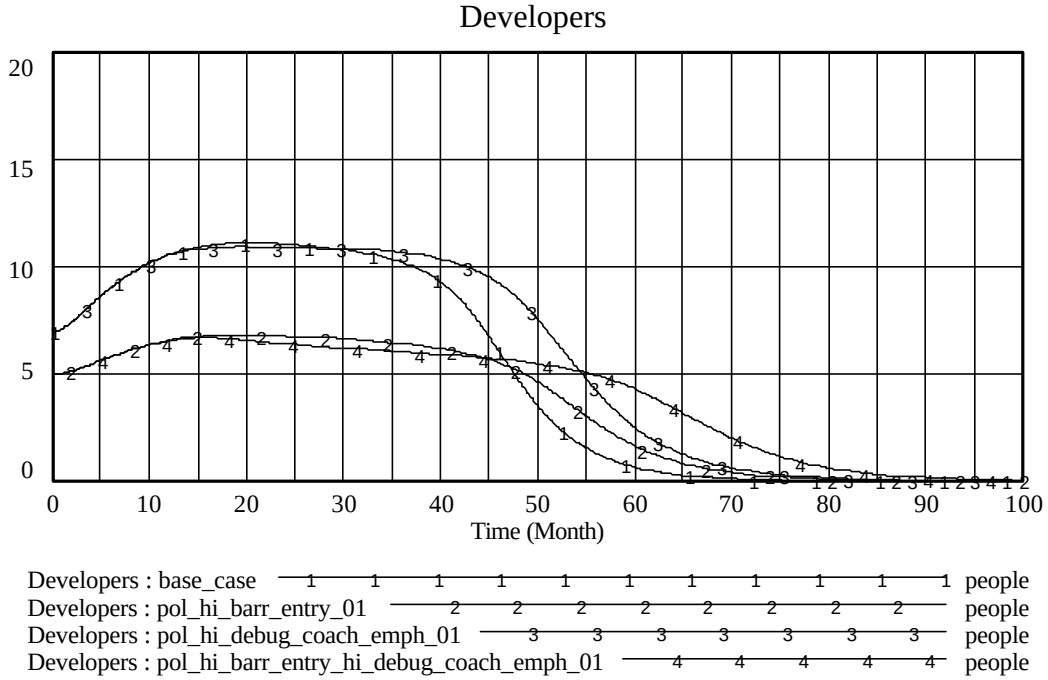


Figure 5.132. Developers under Higher Barriers to Entry, Higher Debugging and Coaching Emphases, and Overall Combination Policy Settings

5.5.8. Higher Barriers to Contribution and Higher Debugging Emphasis

During one of the interviews done with the members of the system dynamics K through 12 community, the interviewee argued that a combination of the higher barriers to contribution and the higher debugging emphasis options would be the most beneficial policy. This combination policy option, which was not in the original policy run set, was then performed on the OSSD model. (See Section 6.3.6 for an analysis of policy comparisons by the interviewees.) Table 5.8 summarizes the policy settings for the compared runs.

Table 5.8. Higher Barriers to Contribution and Higher Debugging Emphasis Policy Settings

Run	**Rejection Ratio**	**Pressure for Bug Detection**	**Pressure for Bug Fixing**
Base Case	0.20	Base Case Level*1	Base Case Level*1
Higher Barriers to Contribution 1	0.40	Base Case Level*1	Base Case Level*1
Higher Debugging Emphasis 1	0.20	Base Case Level*10	Base Case Level*10
Higher Barriers to Contribution and Higher Debugging Emphasis 1	0.40	Base Case Level*10	Base Case Level*10

The combination of higher barriers to contribution and higher debugging emphasis yielded a faster product quality improvement than both of the pure policy options. (See Figure 5.133.) Higher debugging emphasis caught the combination policy in terms of product quality improvement by month 75, or in other words, by the three quarters of the simulation horizon.

On the other hand, the combination policy yielded the slowest product functionality growth among the three policy options. (See Figure 5.134) While the difference between the behaviors of product functionality under the combination policy and the pure barriers to contribution policy was not too large, the combination policy performed much worse than the pure higher debugging emphasis option in terms of product functionality. Community growth under the combination policy was also much slower than that under the pure higher debugging emphasis option. (See Figure 5.135.) The combination policy option yielded a much slower average developer talent growth than that under the pure higher debugging emphasis option, as well. (See Figure 5.136.) However, the behaviors of average developer talent under the combination and the pure higher barriers to contribution options were not too different.

The overall comparison of the three policy runs revealed that a pure higher debugging emphasis policy would yield better overall results than a combination of higher debugging emphasis and higher barriers to entry. On the other hand, the combination policy might be more favorable than a pure higher barriers to contribution policy, since it yields a substantially faster product quality improvement with relatively small marginal losses in functionality and community growth on top of the losses caused by the pure higher barriers to contribution option.

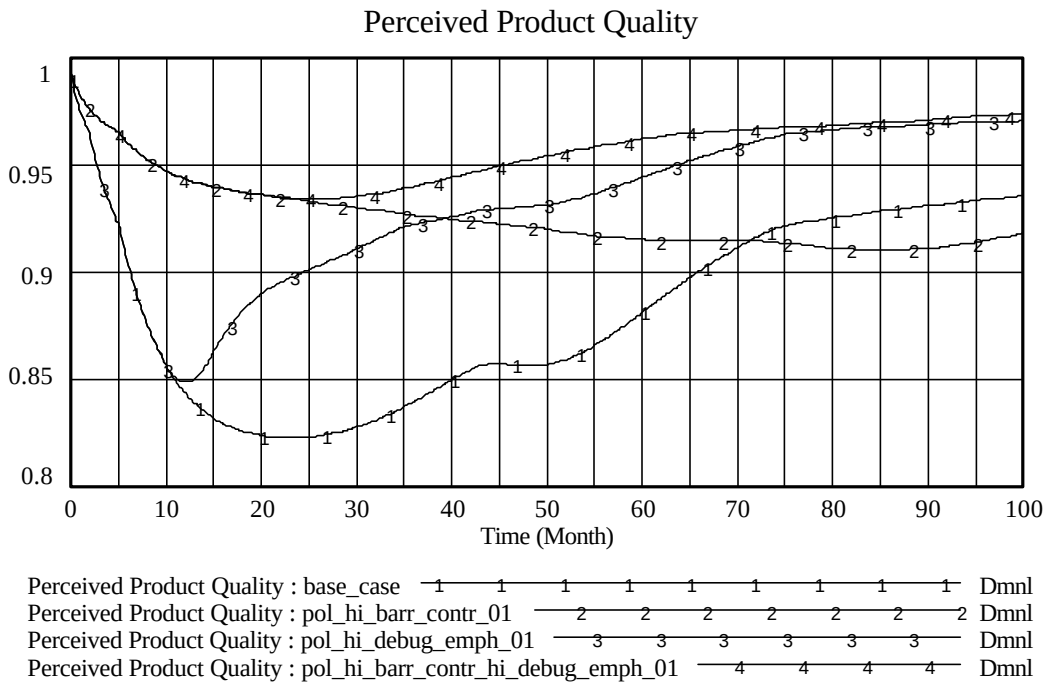


Figure 5.133. Perceived Product Quality under Higher Barriers to Contribution, Higher Debugging Emphasis, and Combination Policy Settings

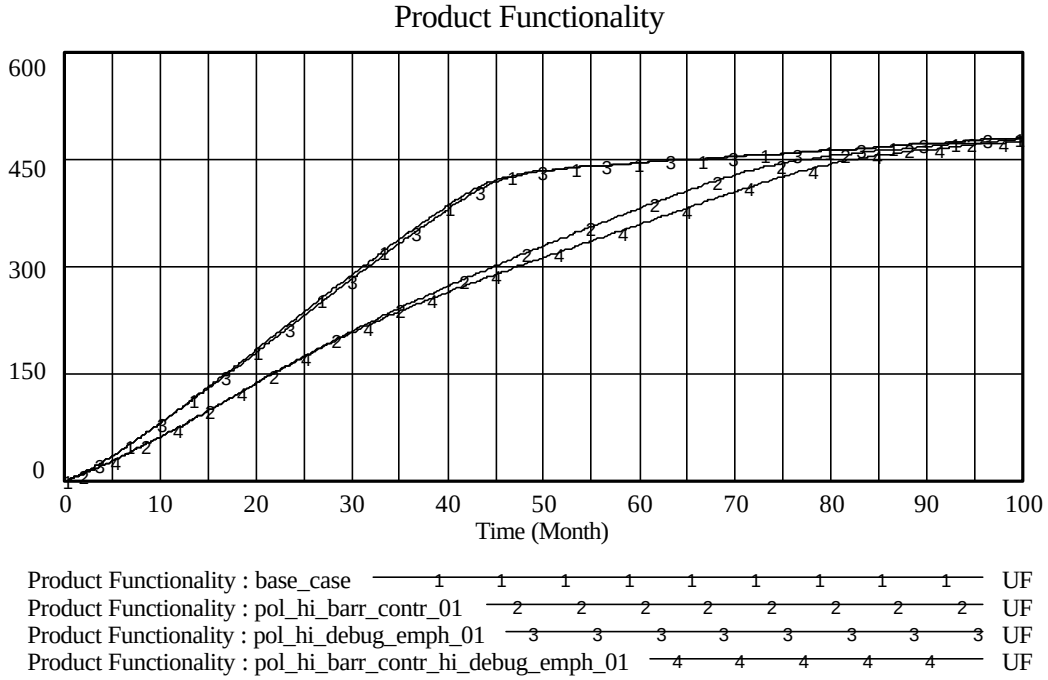


Figure 5.134. Product Functionality under Higher Barriers to Contribution, Higher Debugging Emphasis, and Combination Policy Settings

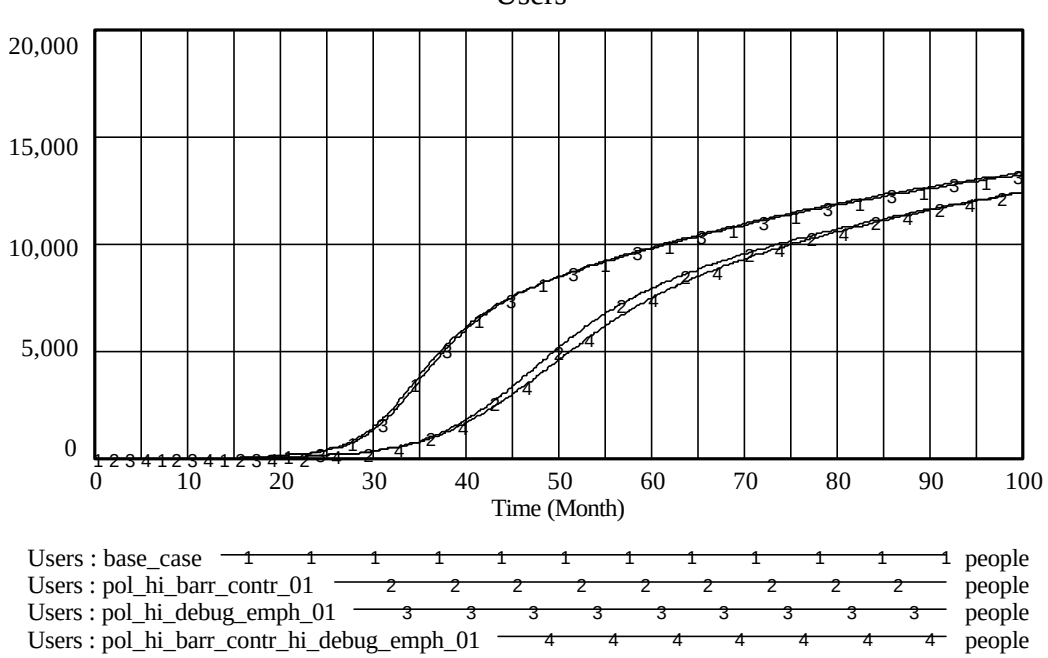


Figure 5.135. Users under Higher Barriers to Contribution, Higher Debugging Emphasis, and Combination Policy Settings

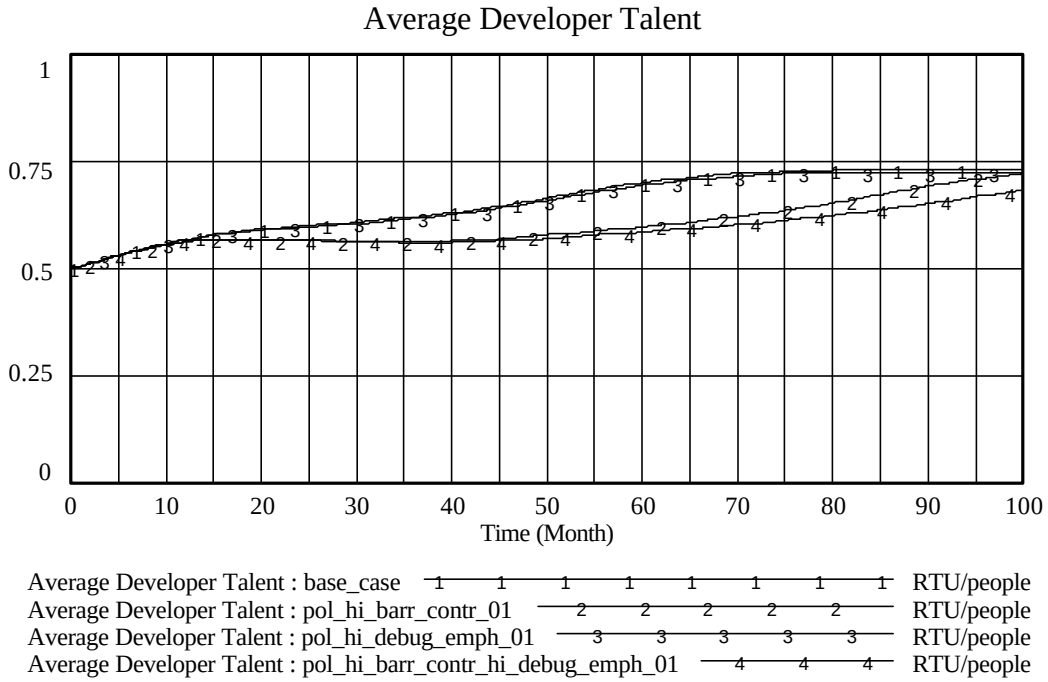


Figure 5.136. Average Developer Talent under Higher Barriers to Contribution, Higher Debugging Emphasis, and Combination Policy Settings

5.5.9. Implications of the Policy Runs

The policy runs demonstrated that the OSSD model has the potential to replicate a variety of behaviors within expected limits under different policy conditions. In that sense, the policy runs helped build confidence in the model, from both internal validity and usefulness perspectives.

The policy runs also provided insight about the effectiveness of different policy options under the existing structure and parameters of the OSSD model. As a general finding, the policy runs showed that any quality improvement policy has the potential of slowing product functionality and community growth beyond a certain level. Furthermore, the marginal quality improvement may decrease substantially as the policy level increases. These two findings together imply smaller quality gains at expense of larger functionality losses as the policy level increases.

Specifically, the two pure-policy runs focusing on barriers to entry and barriers to contribution showed that any quality increase that is gained through these policy options would come at the expense of functionality growth. Furthermore, while the quality gains for relatively lower levels of these policy options are substantial and thus justify the functionality and community growth losses, marginal quality gains for higher policy levels are very small.

The barriers to contribution runs showed that there is a critical level for that policy, where the functionality loss becomes so substantial that the community fails to sustain itself in the long run. Although the barriers to entry runs did not show such a critical level, higher levels of that policy combined with conditions such as low developer

participation, or low productivity may also cause large functionality loses which would fail the community in the long run.

The comparison of the pure-policy options showed that the barriers to entry policy yielded higher and more sustained quality gains for lower functionality loses than the barriers to contribution option. A comparison of the pure barriers to entry policy with a combined barriers to entry and contribution policy showed that the pure barriers to entry policy performed better both in terms of quality gain and functionality loss. However, it should be pointed out that the performances of these three policy options were not dramatically different, and barriers to contribution policy appeared to be an acceptable policy for communities that cannot implement other quality improvement policies for a variety of reasons.

Policy runs under higher debugging emphasis yielded substantial quality improvements with very small losses in product functionality, developer talent and community growth. Although very high levels of this policy option did not impede community growth substantially, marginal improvements by higher levels became very small beyond a point.

Another set of policy runs under higher coaching emphasis conditions provided substantial improvements in average developer talent. However, these runs did not yield the expected levels of product quality improvement, and the limited improvements were not sustained throughout the runs. The cause for limited quality improvement under higher coaching emphasis was found to be a lack of debugging emphasis that would couple the increase in coaching emphasis. It was as if the large improvements in

developer talent achieved in these runs were not being put to use due to low debugging emphasis.

Based on the finding that higher debugging and higher coaching emphases improve the system in different ways, a combination of these two policy options was also put to the test with the expectation that substantial improvements would be achieved both in product quality and average developer talent. As expected, the combination policy option provided better overall results than both pure policy options. It yielded a quality improvement higher than that under the higher debugging emphasis option, and a developer talent improvement almost as high as that under the higher coaching emphasis option. Furthermore, the losses in product functionality and community growths were not critically different than those under the two pure policy options. Thus, the combination policy proved to be a better choice than the two pure policy options.

Another set of policy runs was performed under an overall combination of the two best policy options of the earlier runs: higher barriers to entry, and higher debugging and coaching emphases. The product quality and developer talent improvements under the overall combination policy were higher than those under the two alternative options. However, the product functionality and community growth losses were also greater under the combination policy conditions. Furthermore, comparing the pure higher barriers to entry option with the combined higher debugging and coaching emphases option revealed that the first option yielded a faster and larger quality improvement, while the second yielded a higher average developer talent in the long run.

The final policy run combining higher barriers to contribution and higher debugging emphasis policies yielded a very fast product quality improvement, but caused

the product functionality and community growths slow down substantially. The combination policy option was found to be more favorable than the pure higher barriers to contribution option, since it yielded a much faster and larger product quality improvement in expense of a relatively small additional loss in product functionality and community growth. However, the overall performance of the combination policy was not better than that of the pure higher debugging emphasis policy, since the marginal improvement in product quality was not high enough to justify the marginal loss in product functionality and community growth.

These findings clearly showed that an open source software community has to consider the trade-off between building functionality and improving quality while developing policies. Based on these findings, this study defines the underlying policy problem in an open source software development community as the tension between building product functionality and improving product quality while sustaining community growth. Furthermore, there are several ways to achieve quality improvement, including policies such as setting barriers to entry or contribution, putting more emphasis on debugging or coaching, or a combination of these and other policies.

5.6. Analysis of Bifurcation Behavior

An important observation during the sensitivity and policy runs was the existence of behavioral bifurcation points that separated successful and unsuccessful cases under different parametric conditions. For example, when average developer participation was set to values below a certain point, the community could not sustain itself in the long run. The same behavior was observed for values of average developer productivity below a certain point. (See Section 5.4.1 and Section 5.4.2.) Also, policy runs such as those for

higher barriers to contribution indicated the existence of bifurcation points for some policy options. (See Section 5.5.2.) These observations indicated an underlying cause that drives the community to failure under a set of parametric conditions.

An analysis of the model structure revealed that the cause behind the bifurcation behavior is the patience factor. As discussed in Section 4.3, the OSSD model assumes a general level of patience that determines the expectations of the users and the developers related to product functionality. Patience runs out as time passes, and thus the expectation about the functionality of the product increases. When the real functionality achievement is below the expected level, the attractiveness of the community for both users and developers decreases. On the other hand, a functionality achievement above the expected level attracts users and developers more.

As a starting point, a set of sensitivity runs was done with different values of normal time to lose patience -- the rate with which patience diminishes. The results reveled that the model is sensitive to changes in the value of this variable, especially if the value is below 25 months. A decreased normal time to lose patience causes the expectations about product functionality to increase faster (See Figure 5.137.) When the achieved level of functionality cannot match the fast increase in expectations, a large number of developers lose their motivations and leave the community. (See Figure 5.138.) This further decreases the community's ability to achieve a functionality level that can match the expectations. As a consequence, product functionality stagnates, and this decreases the number of new users, slowing down community growth. (See Figure 5.139 and Figure 5.140.) For values of normal time to lose patience that are below a certain level community fails to sustain product functionality and community growth, and

disintegrates. Further analysis indicated that the critical value is between 12 and 13 months. Perceived product quality and average developer talent were also lower for lower values of normal time to lose patience. (See Figure 5.141 and Figure 5.142.)

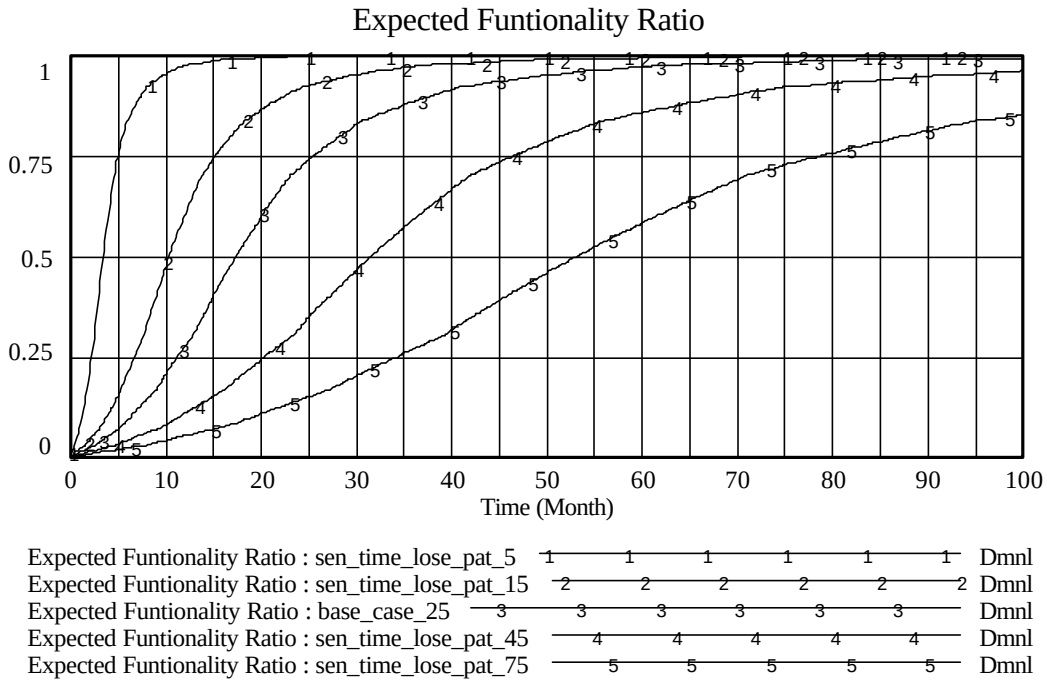


Figure 5.137. Expected Product Functionality Ratio for Different Values of Normal Time to Lose Patience

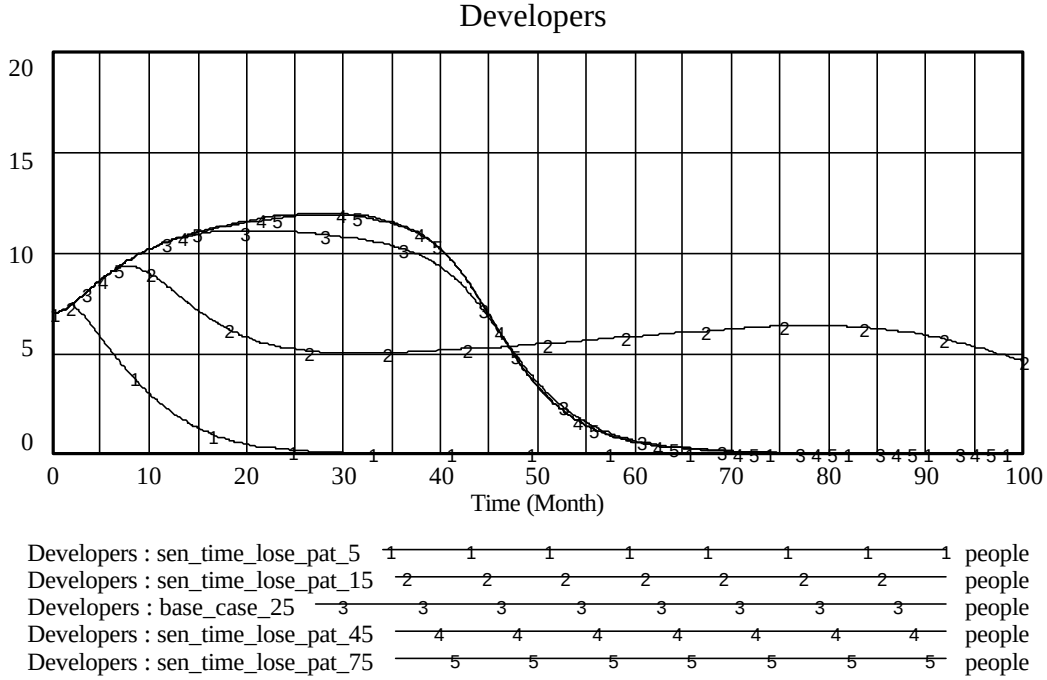


Figure 5.138. Developers for Different Values of Normal Time to Lose Patience

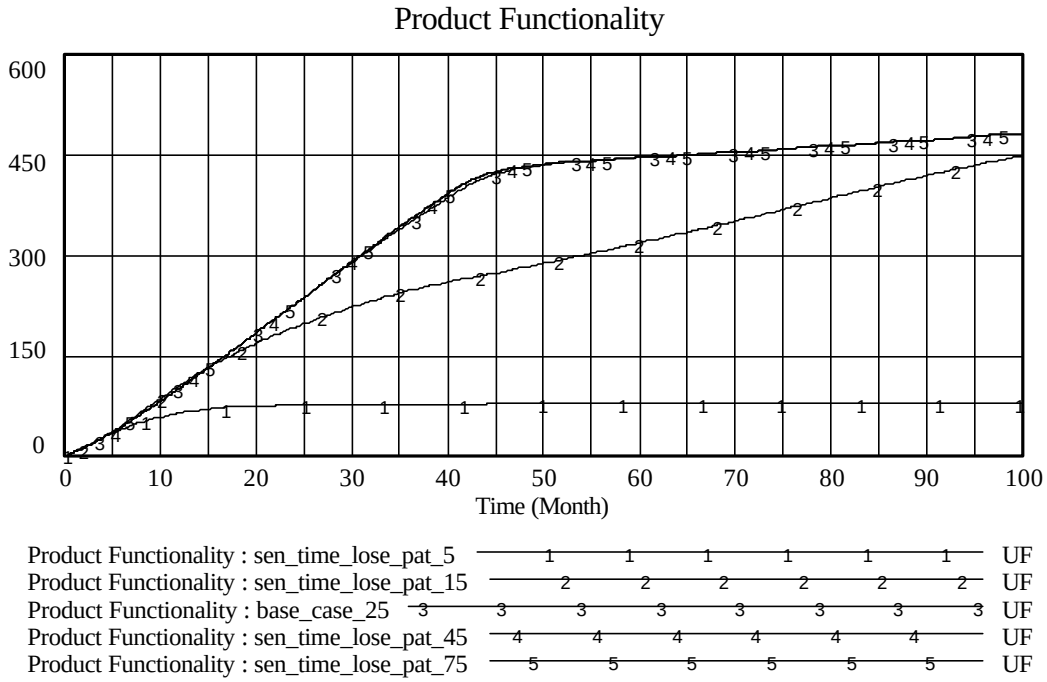


Figure 5.139. Product Functionality for Different Values of Normal Time to Lose Patience

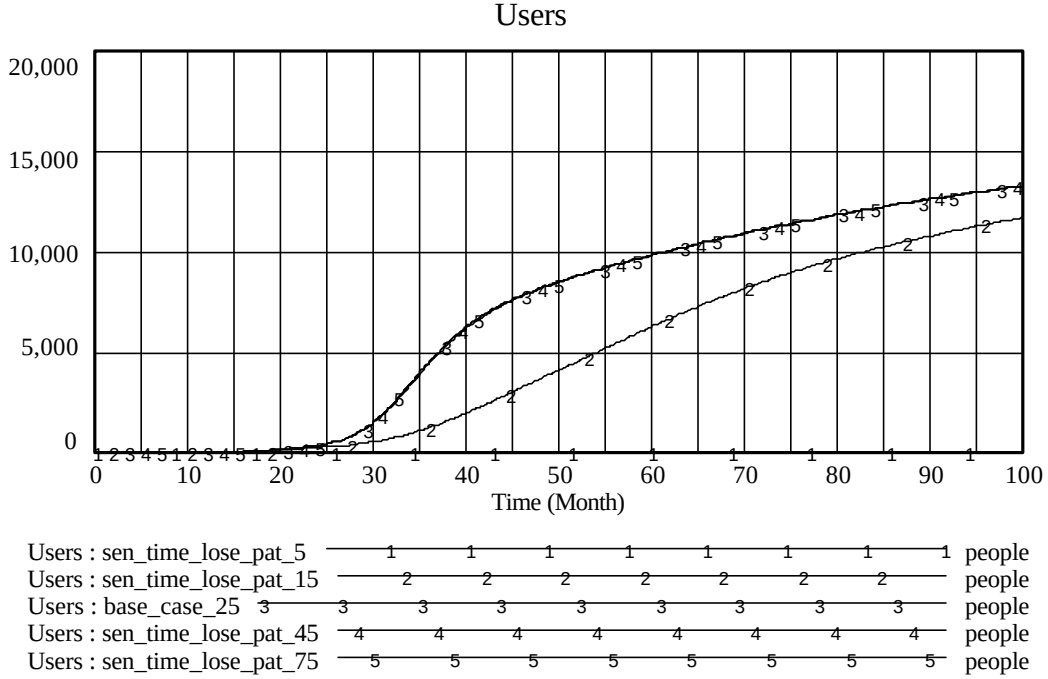


Figure 5.140. Users for Different Values of Normal Time to Lose Patience

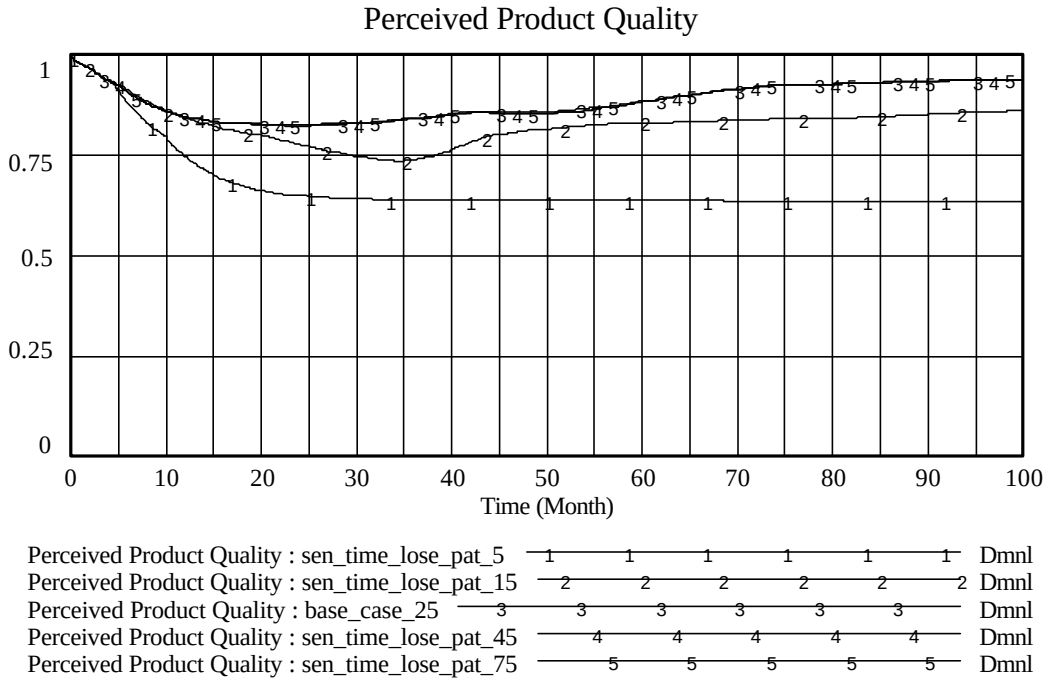


Figure 5.141. Perceived Product Quality for Different Values of Normal Time to Lose Patience

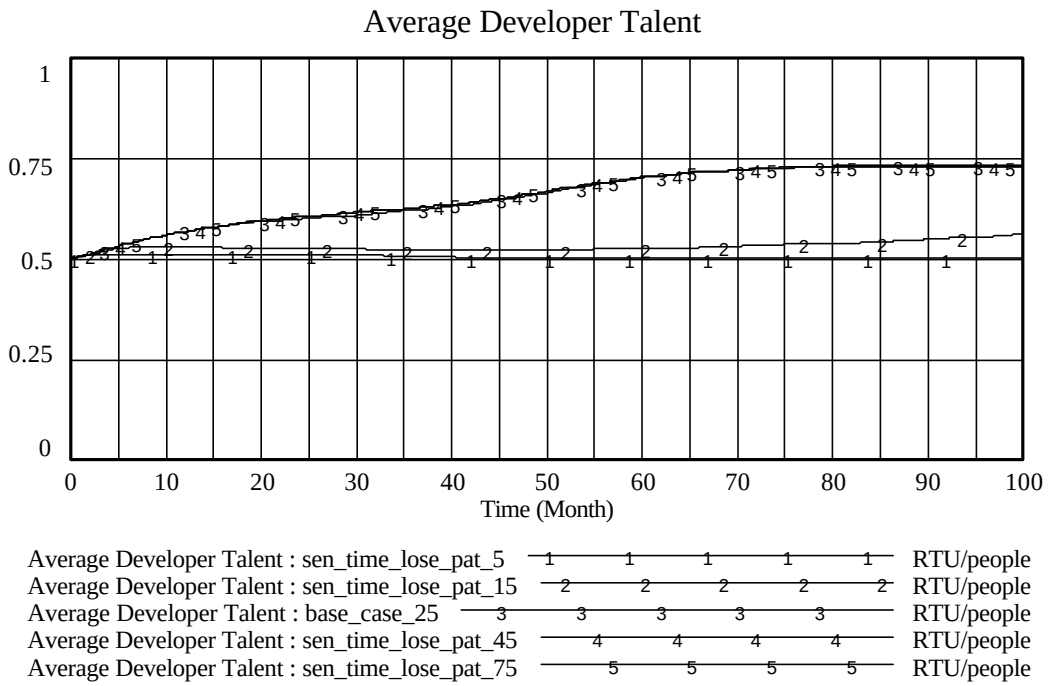


Figure 5.142. Average Developer Talent for Different Values of Normal Time to Lose Patience

These results revealed that the bifurcation is caused fundamentally by the discrepancy between the expectations about functionality growth and the actual growth in functionality. If functionality growth cannot measure up to expectations due to low participation, low productivity or a similar factor, or if the expectations grow far faster than the actual functionality growth the community fails to sustain itself and disintegrates.

Several additional sensitivity and policy runs were made to analyze the importance of the patience factor within the overall model structure, and its effects on model behavior under different parametric conditions and policy settings. These runs revealed that the patience factor is indeed an important determinant of model behavior, and that it has a large effect on the outcomes of policy options.

As a starting point the base case was run under the condition of infinite patience. For this run, normal time to lose patience was set to a very high number, which kept the patience level constant throughout the run. There behaviors of product functionality and the number of users were almost indentical to their behaviors in the original base case run. (See Figure 5.143 and Figure 5.144.)

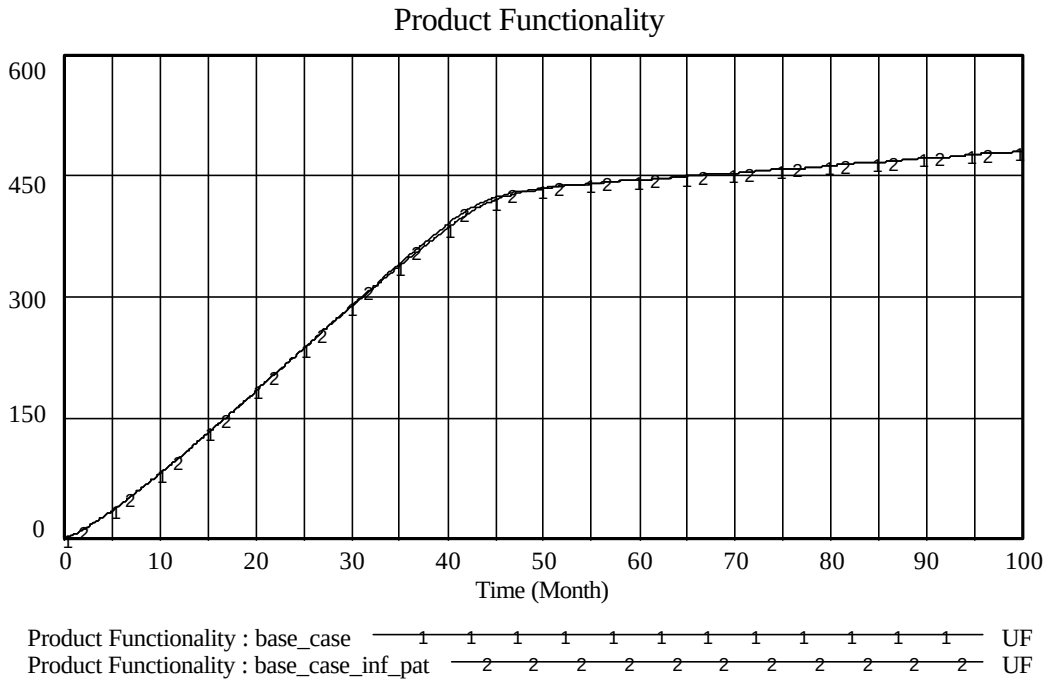


Figure 5.143. Product Functionality under Base Case Conditions and under Infinite Patience Assumption

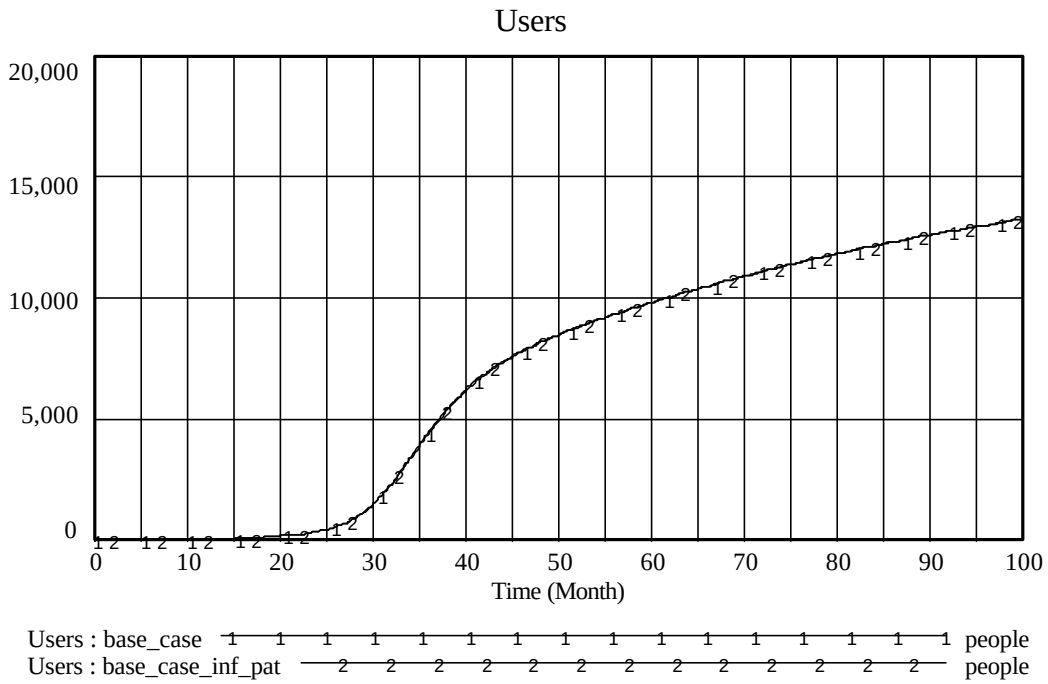


Figure 5.144. Users under Base Case Conditions and under Infinite Patience Assumption

The behavior of the number of developers was slightly different than its behavior in the original base case run. (See Figure 5.145.) This is attributable to the change in the behavior of attractiveness of product for developers due to achieved functionality, which, in turn, was caused by the change in the behavior of operative functionality versus expected functionality. (See Figure 5.146 and Figure 5.147.)

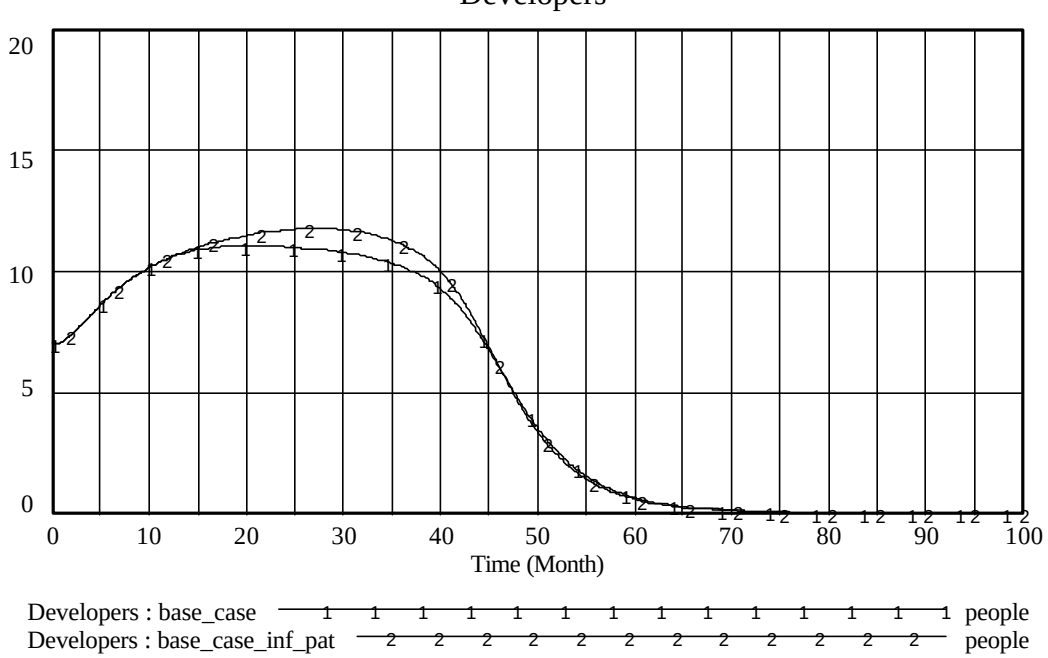


Figure 5.145. Developers under Base Case Conditions and under Infinite Patience Assumption

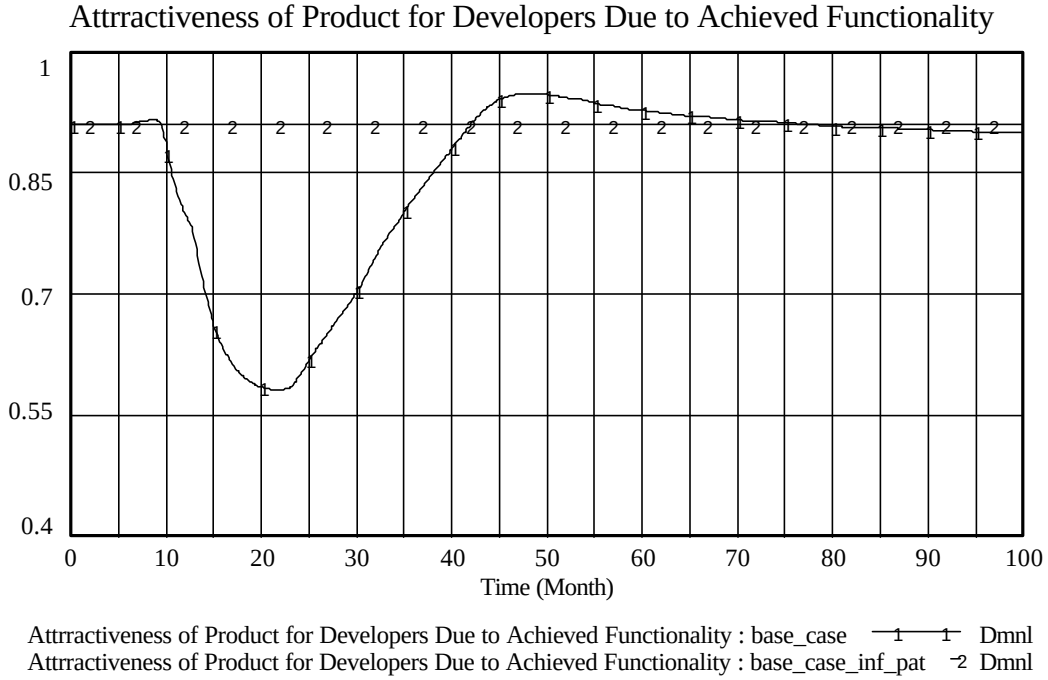


Figure 5.146. Attractiveness of Product for Developers Due to Achieved Functionality under Base Case Conditions and under Infinite Patience Assumption

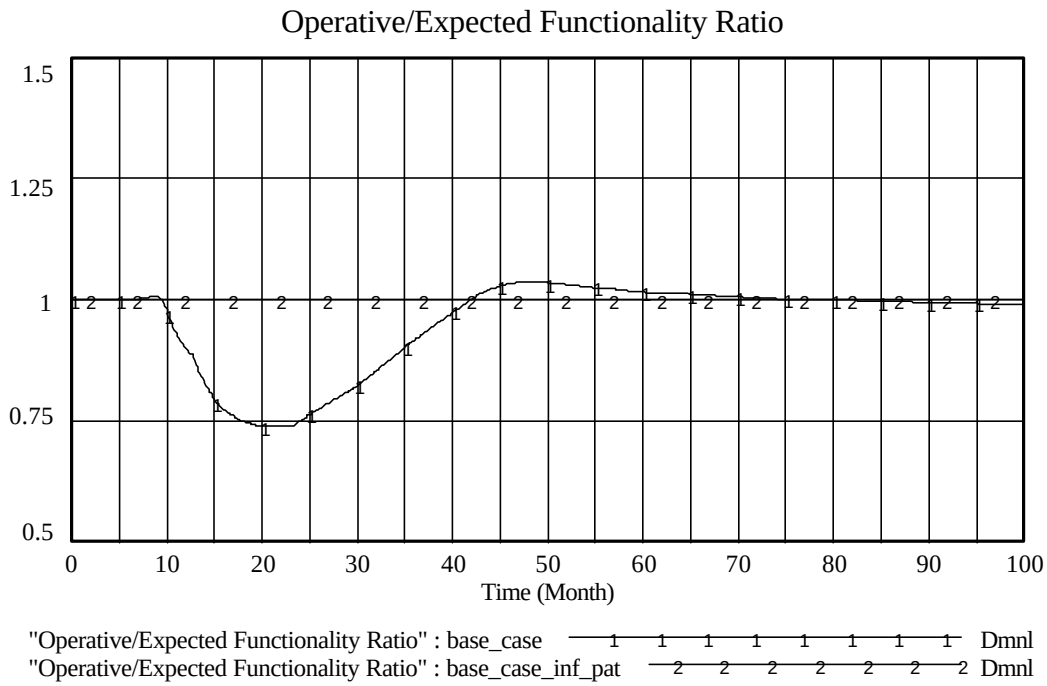


Figure 5.147. Operative/Expected Functionality Ratio under Base Case Conditions and under Infinite Patience Assumption

A number of the runs involved the replication of the sensitivity runs that indicated bifurcation points. One group of such runs was done for different values of average developer participation. The sensitivity runs under the original diminishing patience assumption of the model indicated that there is a bifurcation point somewhere between 10 to 11 hours per month average developer participation. (See Section 5.4.1.) Consequently, the original sensitivity runs with values of average developer participation below 11 hours/month portrayed behaviors where the community failed to sustain itself and disintegrated. On the other hand, the sensitivity runs for different values of average developer participation under the “infinite patience” assumption rendered a completely different picture. As Figure 5.148 shows, product functionality grew and approached the limit on product functionality for even very low values of average developer

participation. Product functionality grew considerably slower for lower values of average developer participation, but the community was able to sustain the functionality growth.

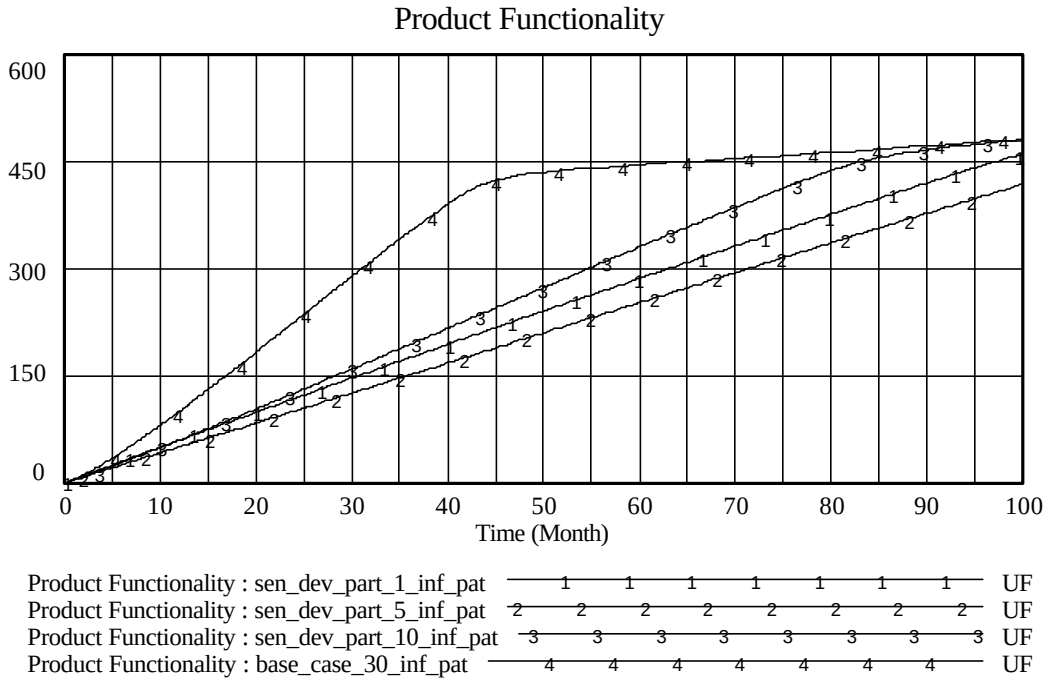


Figure 5.148. Product Functionality for Different Values of Average Developer Participation under Infinite Patience Assumption

Community growth could also be sustained for even extremely low values of average developer participation. Figure 5.149 shows that although the number of users grew slower for lower values of average developer participation, the growth could be sustained in all of the runs.

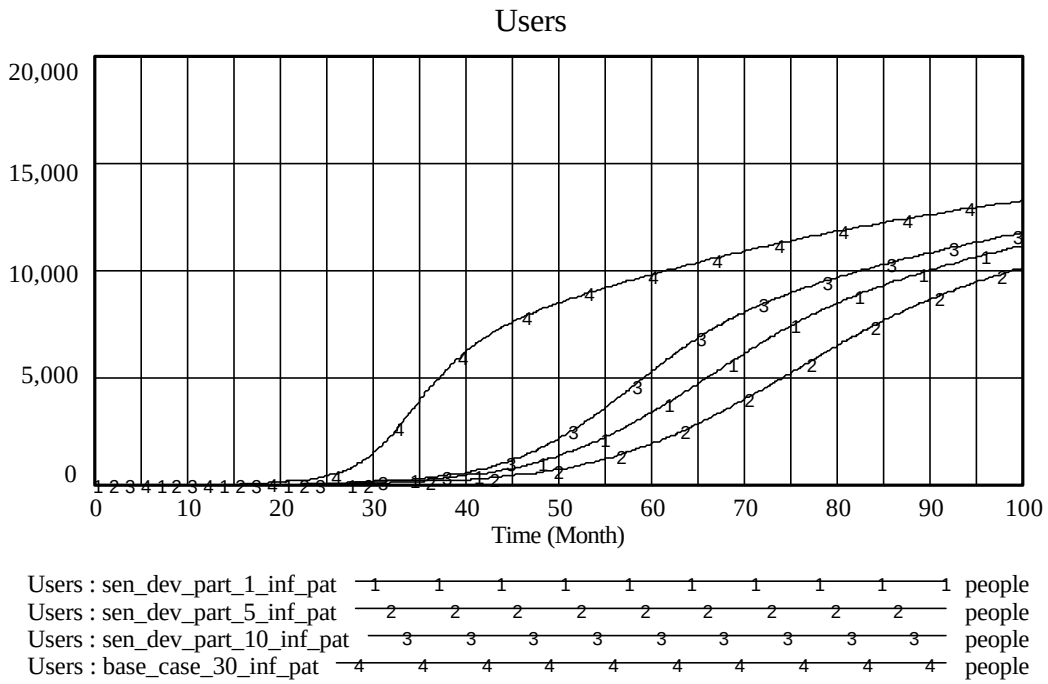


Figure 5.149. Users for Different Values of Average Developer Participation under Infinite Patience Assumption

Average developer productivity was another variable, the lower values of which led the community to fail in the original sensitivity runs. The bifurcation point for this variable was somewhere between 1.6 and 1.7 lines/hour. (See Section 5.4.2.) Under the infinite patience assumption, no bifurcation was observed for this variable, as well. Figure 5.150 and Figure 5.151 show that product functionality and community growth could be sustained for even very low values of average developer productivity. Again, product functionality and community growth were slower for lower values of average developer productivity, as expected.

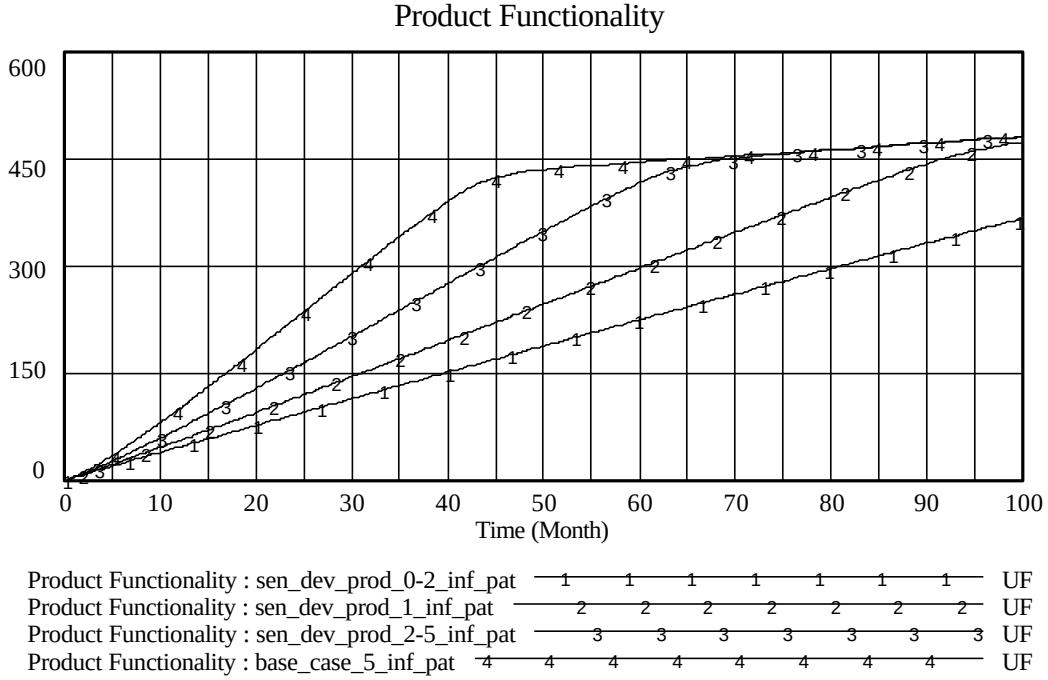


Figure 5.150. Product Functionality for Different Values of Average Developer Productivity under Infinite Patience Assumption

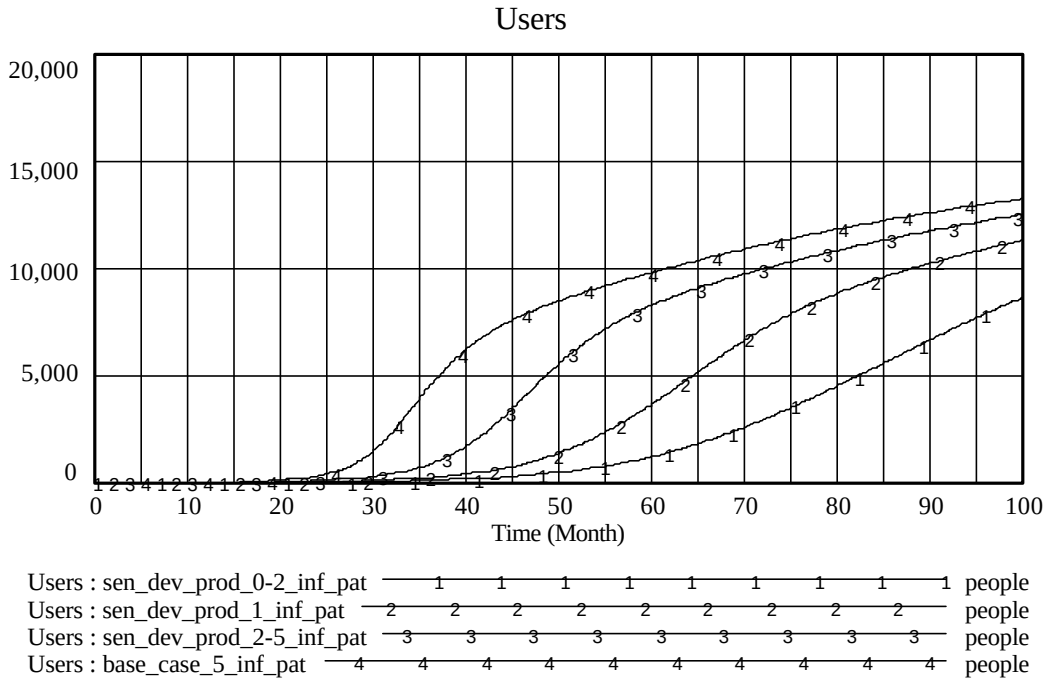


Figure 5.151. Users for Different Values of Average Developer Productivity under Infinite Patience Assumption

Some of the policy runs were replicated under the infinite patience assumption, as well. One such policy runs was higher barriers to entry. In the original set of policy runs, the community failed to sustain its growth under very high level of he barriers to entry option. (See Section 5.5.1.) However, under the infinite patience assumption, product functionality and community growth could be sustained even for very high levels of the barriers to entry option. (See Figure 5.152 and Figure 5.153.) The behaviors of perceived product quality and average developer talent were not different than those in the original set of higher barriers to entry policy runs. (See Figure 5.154 and Figure 5.155.)

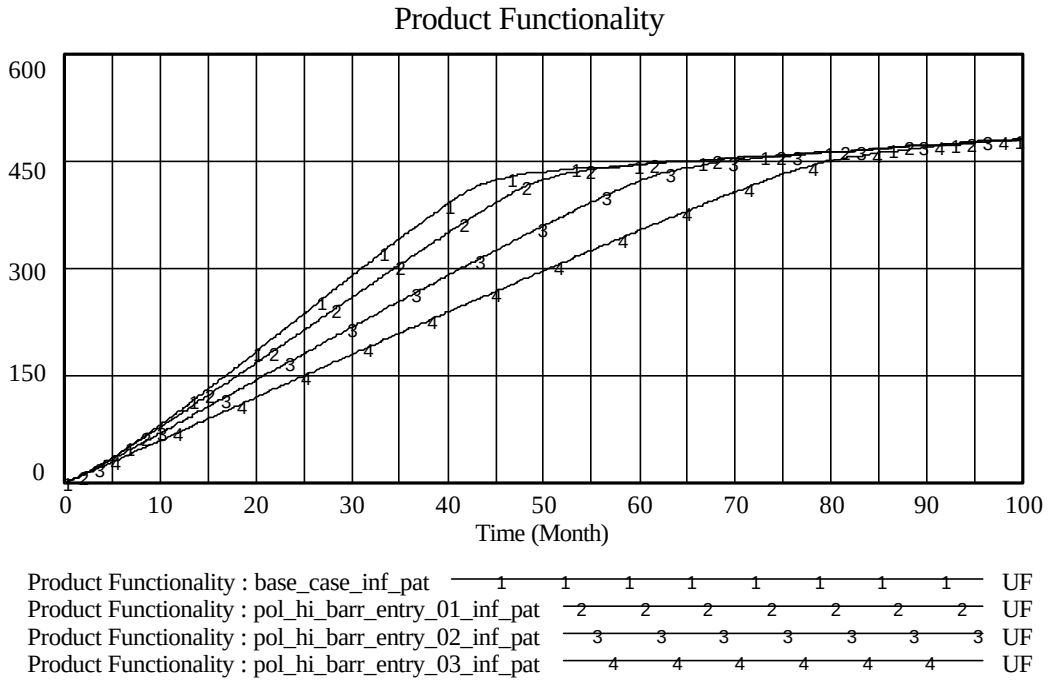


Figure 5.152. Product Functionality for Different Barriers to Entry Policy Settings under Infinite Patience Assumption

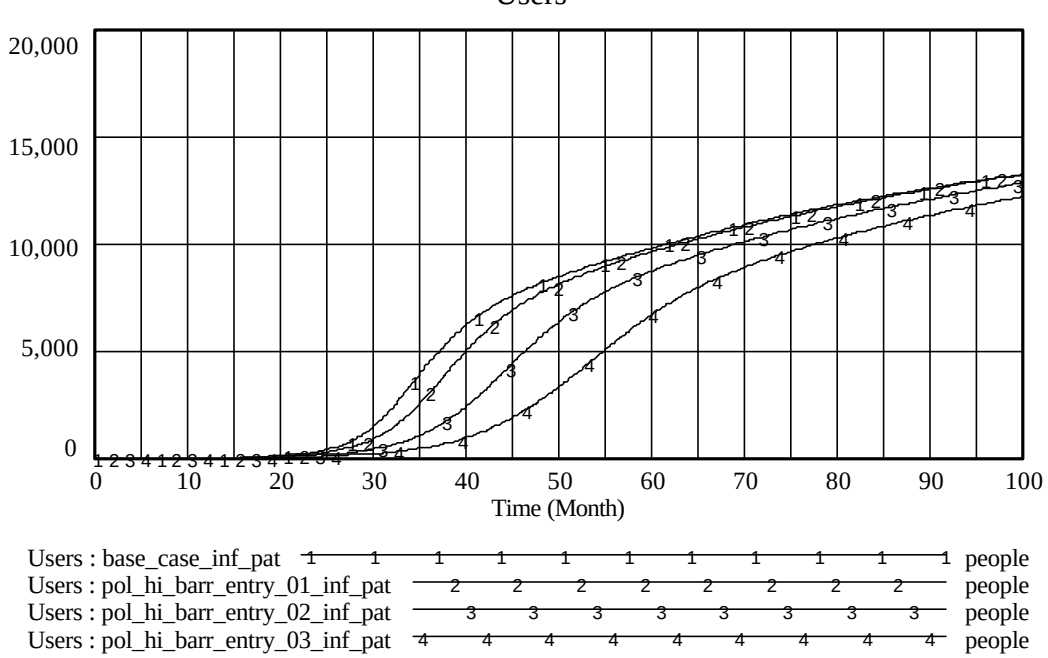


Figure 5.153. Users for Different Barriers to Entry Policy Settings under Infinite Patience Assumption

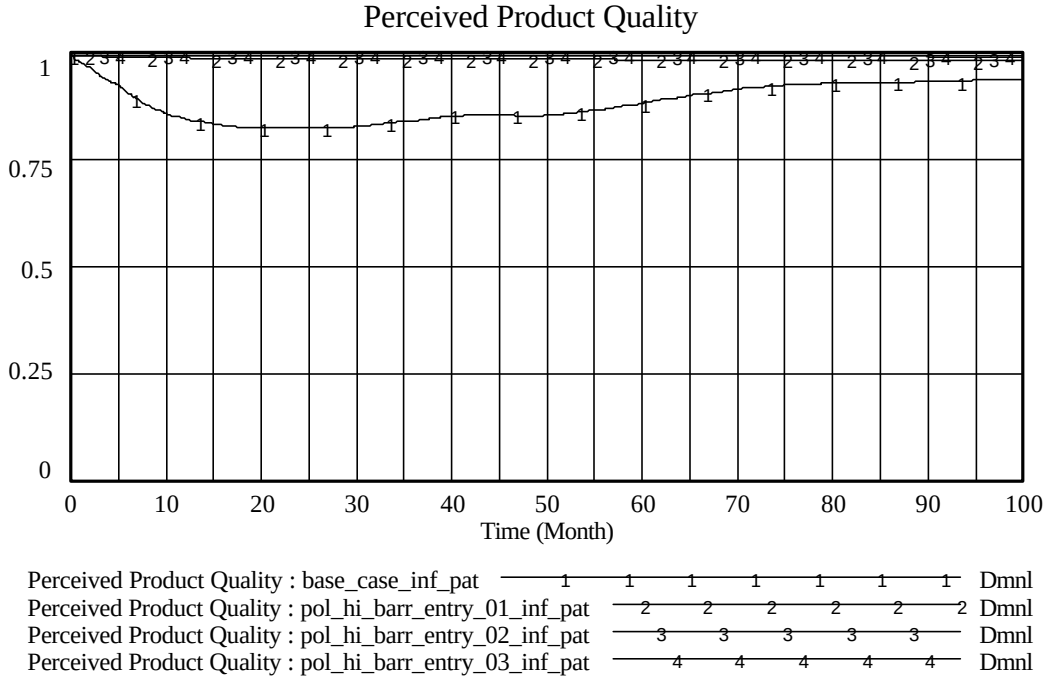


Figure 5.154. Perceived Product Quality for Different Barriers to Entry Policy Settings under Infinite Patience Assumption

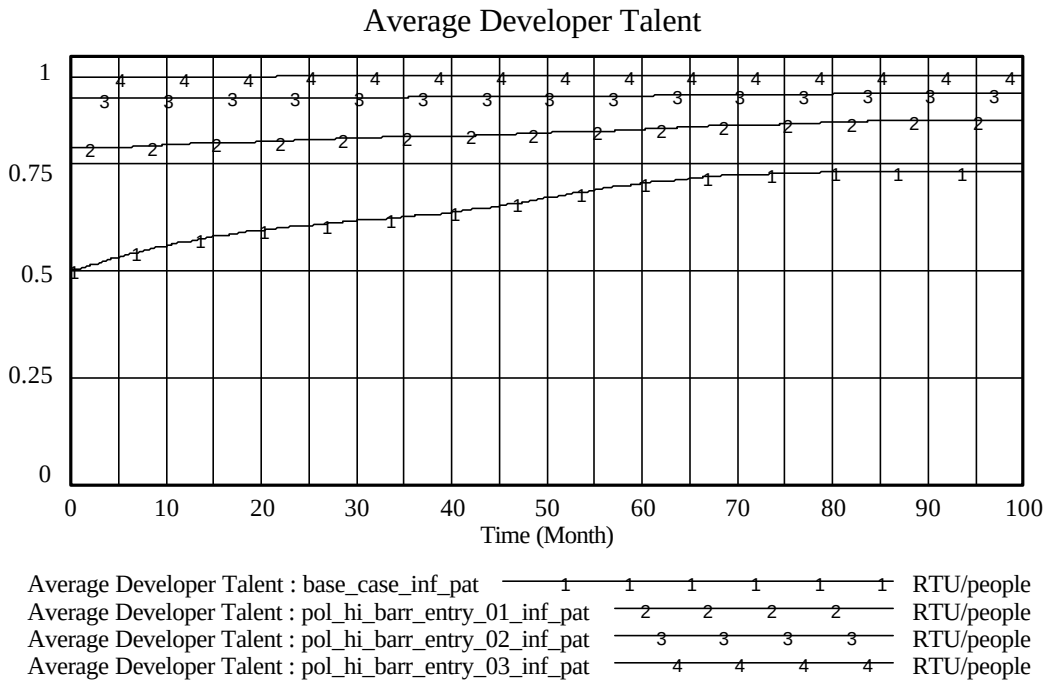


Figure 5.155. Average Developer Talent for Different Barriers to Entry Policy Settings under Infinite Patience Assumption

Higher barriers to contribution option was another policy, very high levels of which caused the community to fail to sustain itself in the long run under the original diminishing patience assumption. (See Section 5.5.2.) Higher barriers to contribution policy did not cause such a failure under the infinite patience assumption. Even for the highest setting of this policy option the community could sustain product functionality and community growth. (See Figure 5.156 and Figure 5.157.) The behavior of perceived product quality was not critically different than that in the original set of higher barriers to contribution policy runs. (See Figure 5.158.) Average developer talent was higher for the same level of higher barriers to contribution under the infinite patience assumption than its level under the original assumption, due to the decreased number of leaving developers under the infinite patience assumption. A smaller number of leaving

developers decreases the talent loss, and thus yields a higher average developer talent. (See Figure 5.159.)

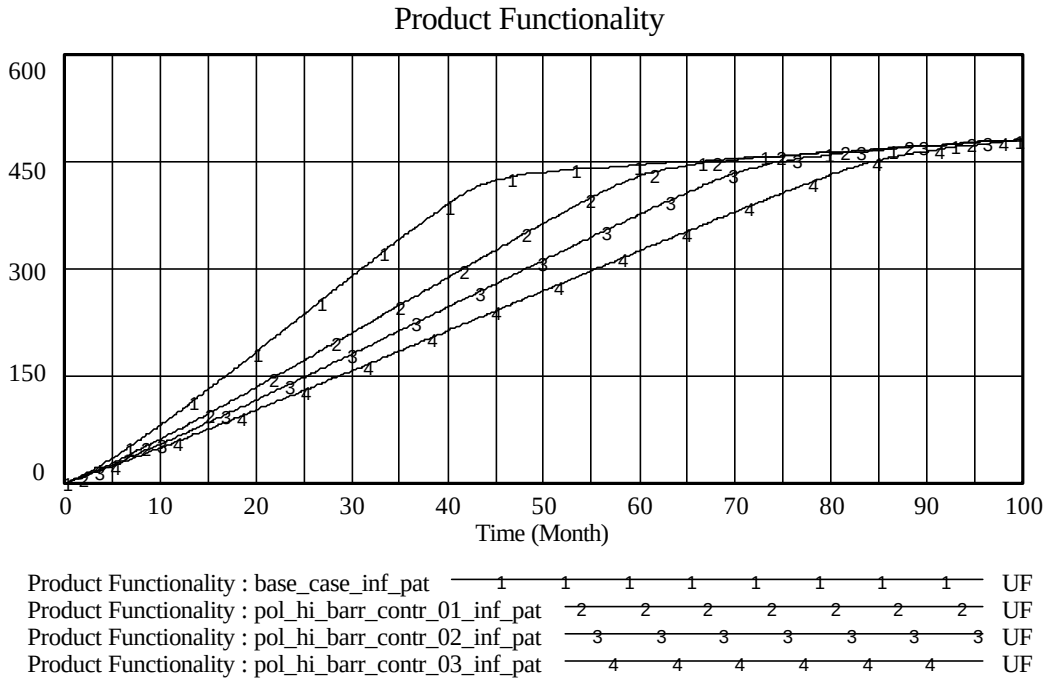


Figure 5.156. Product Functionality for Different Barriers to Contribution Policy Settings under Infinite Patience Assumption

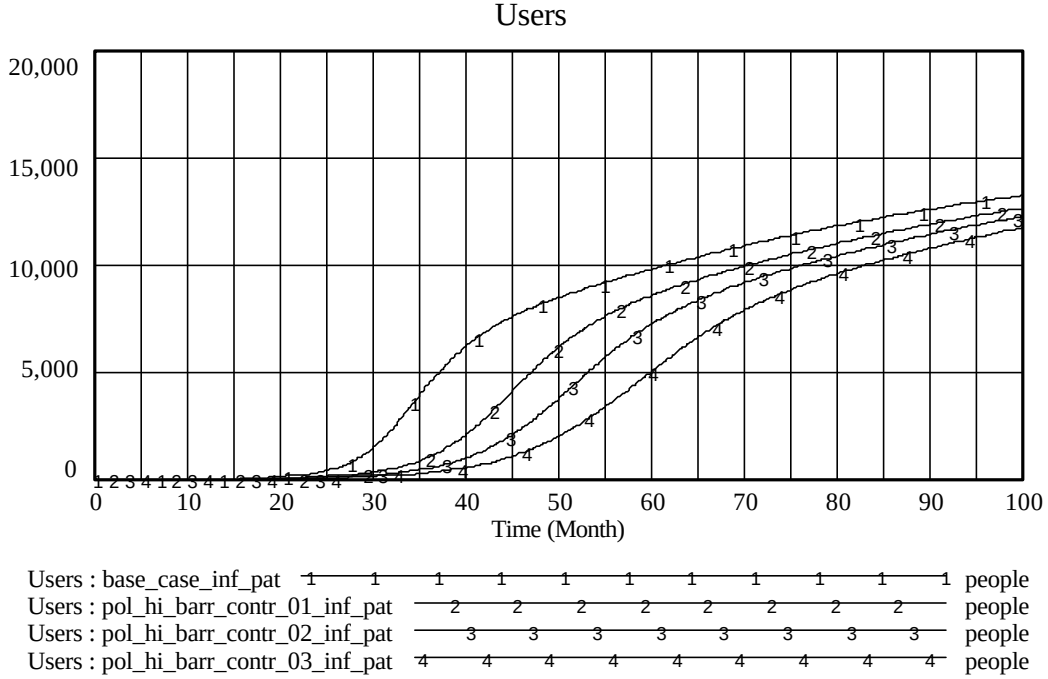


Figure 5.157. Users for Different Barriers to Contribution Policy Settings under Infinite Patience Assumption

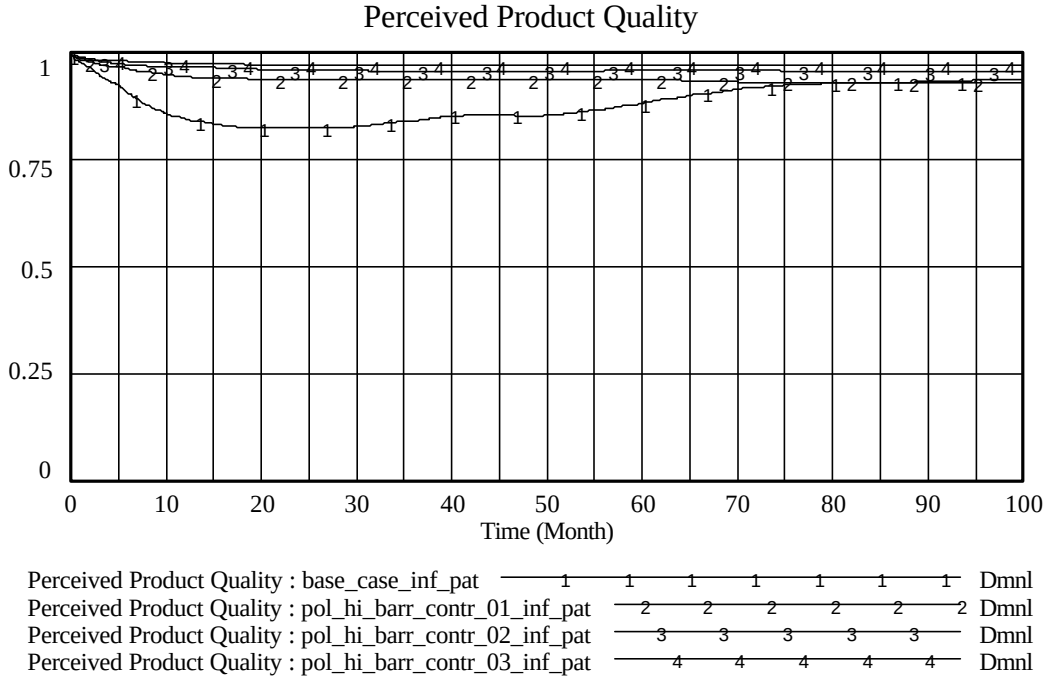


Figure 5.158. Perceived Product Quality for Different Barriers to Contribution Policy Settings under Infinite Patience Assumption

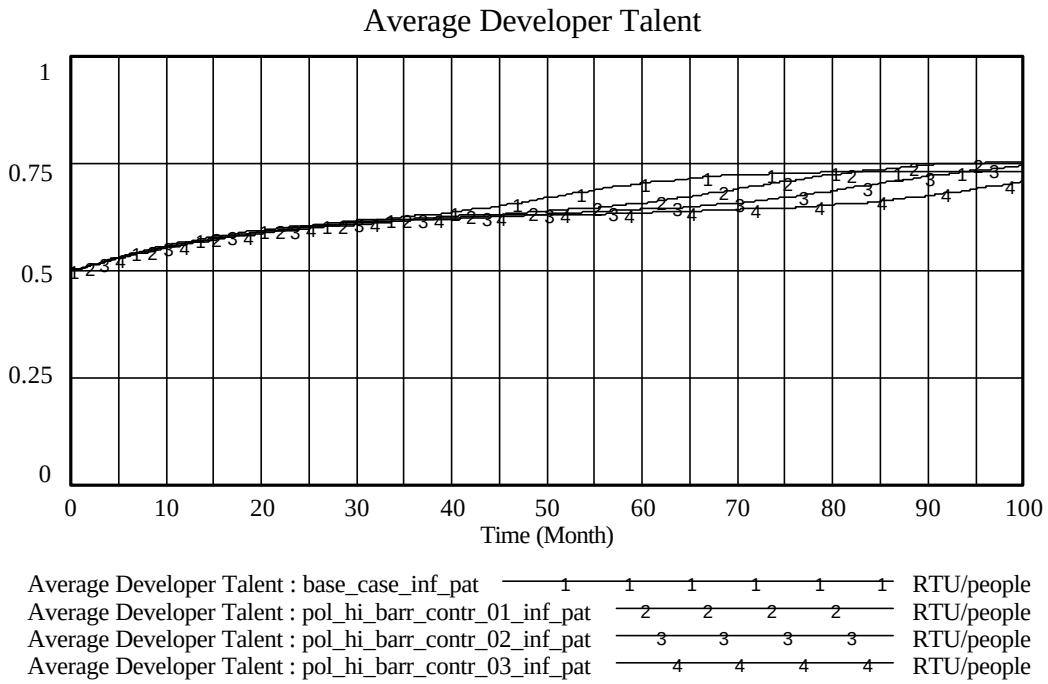


Figure 5.159. Average Developer Talent for Different Barriers to Contribution Policy Settings under Infinite Patience Assumption

The original policy analyses under the diminishing patience assumption included a comparison of the pure higher barriers to entry option with a combination of higher barriers to entry and higher barriers to contribution options. (See Section 5.5.3.) Replicating those runs under the infinite patience assumption yielded findings similar to those under the original diminishing patience assumption. Here again, the pure higher barriers to entry option performed better than the combination policy in terms of product functionality, community and average developer talent growth. (See Figures 5.160 through 5.162.) Once again, there was virtually no difference between the quality improvements yielded by these two policy options. (See Figure 5.163.)

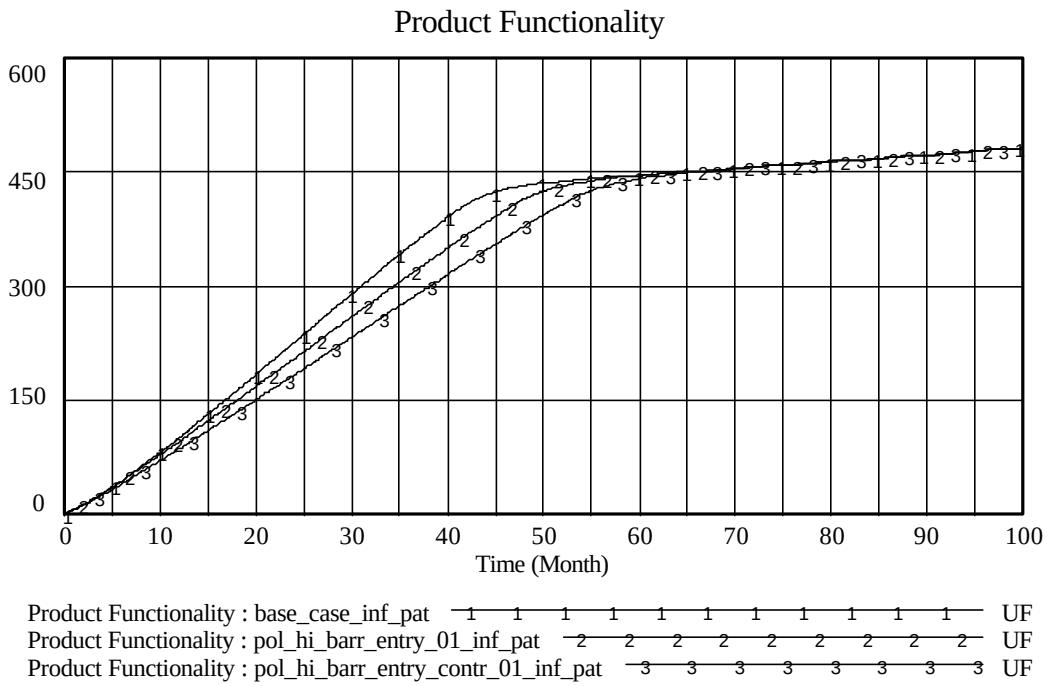


Figure 5.160. Product Functionality for Barriers to Entry and Combination Policy Settings under Infinite Patience Assumption

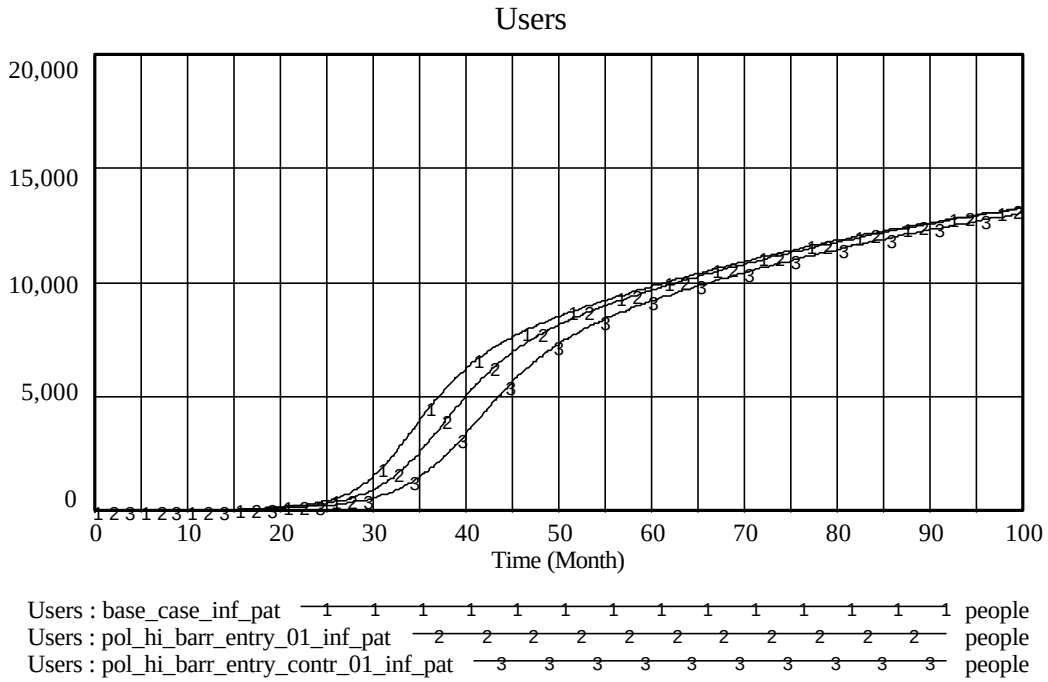


Figure 5.161. Users for Barriers to Entry and Combination Policy Settings under Infinite Patience Assumption

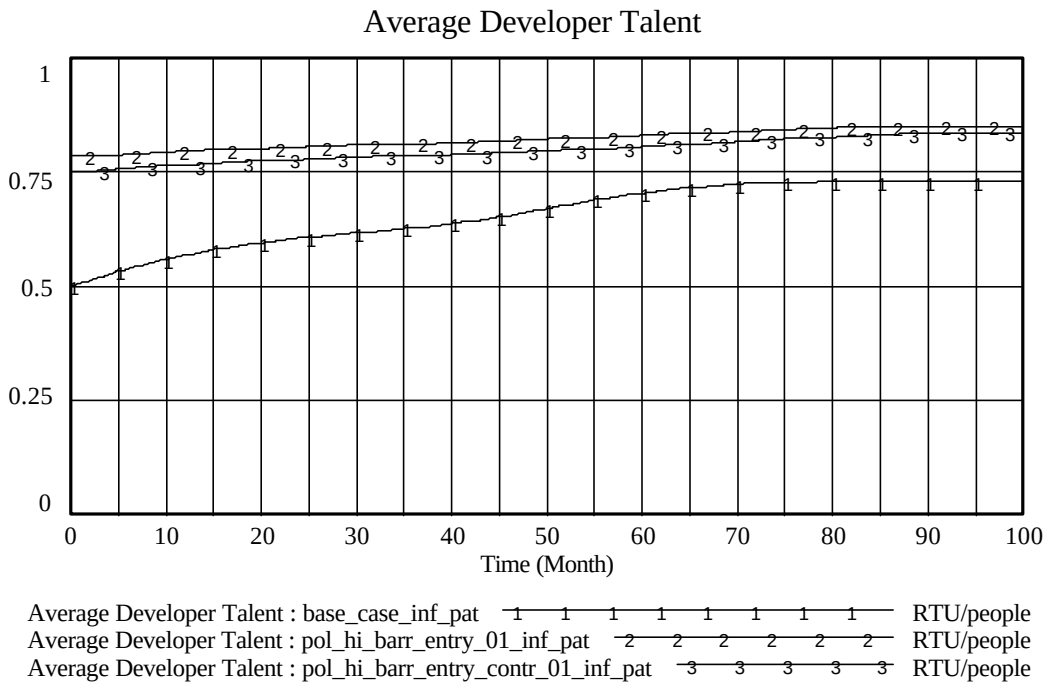


Figure 5.162. Average Developer Talent for Barriers to Entry and Combination Policy Settings under Infinite Patience Assumption

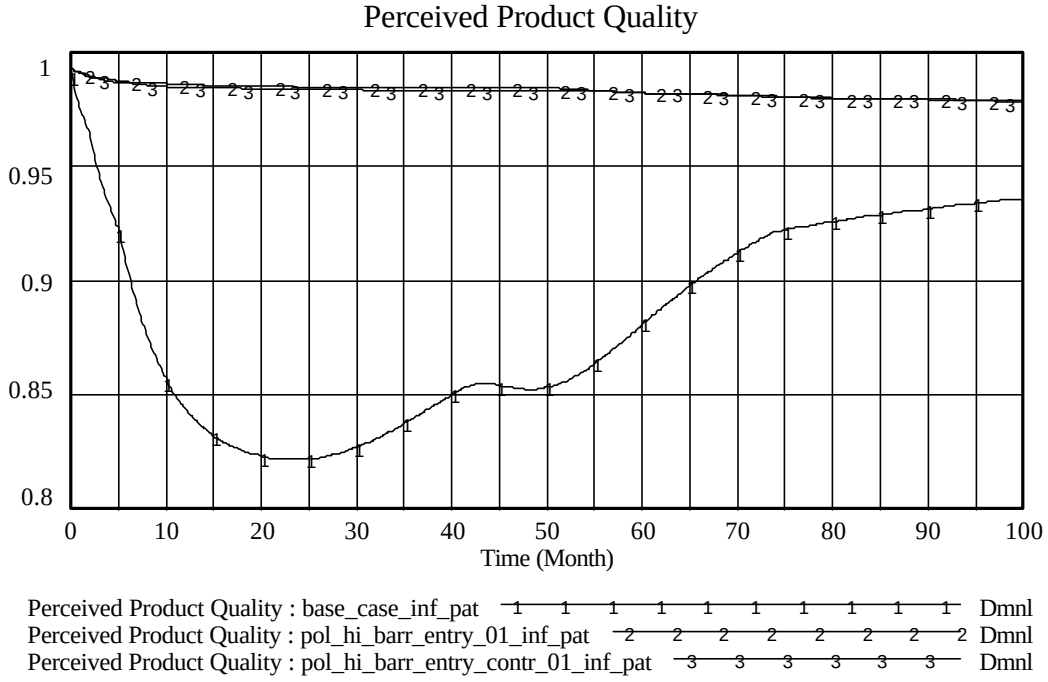


Figure 5.163. Perceived Product Quality for Barriers to Entry and Combination Policy Settings under Infinite Patience Assumption

Another combination policy option compared with its pure counterparts under the original diminishing patience assumption was the combination of higher debugging and higher coaching emphases options. (See Section 5.5.6.) Comparing the pure higher debugging and higher coaching emphases options with the combination option under infinite patience assumption yielded results that were similar to those under the original diminishing patience assumption. Once again, the combination policy performed better than the two pure options in the overall. The behaviors of the key variables were not critically different than those under the original diminishing patience assumption. (See Figures 5.164 through 5.167.)

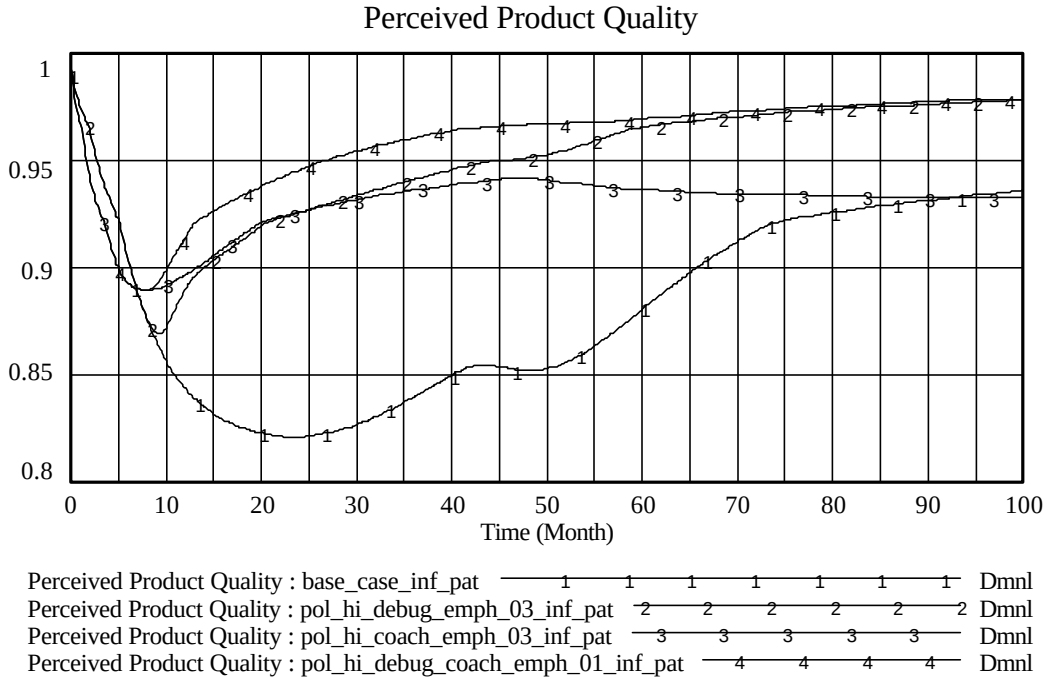


Figure 5.164. Perceived Product Quality for Higher Debugging, Higher Coaching, and Combination Policy Settings under Infinite Patience Assumption

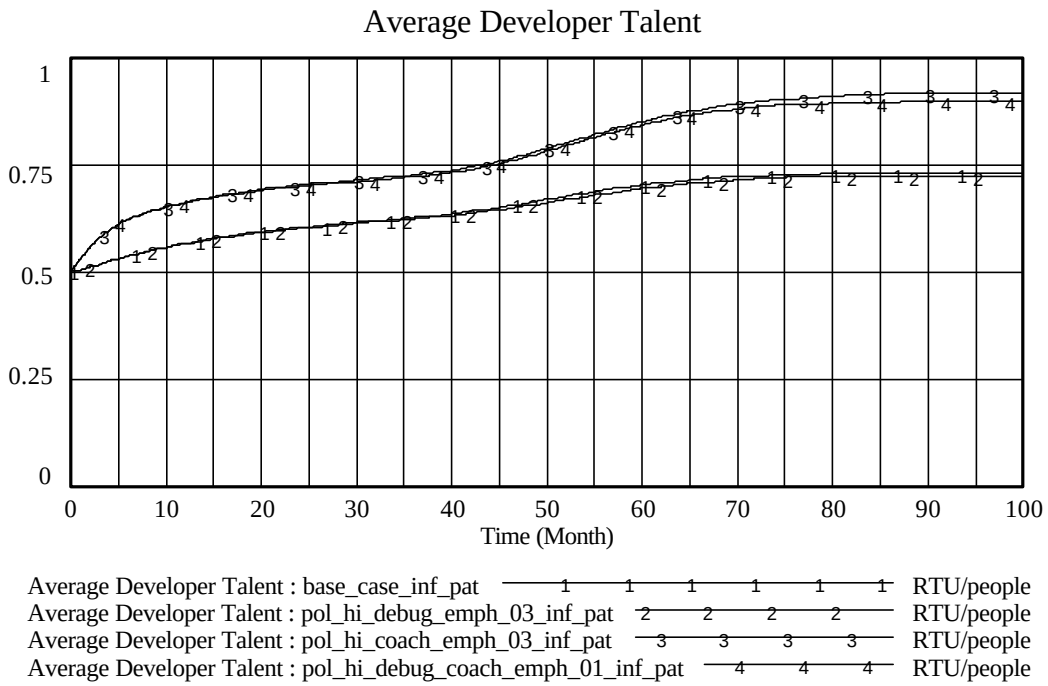


Figure 5.165. Average Developer Talent for Higher Debugging, Higher Coaching, and Combination Policy Settings under Infinite Patience Assumption

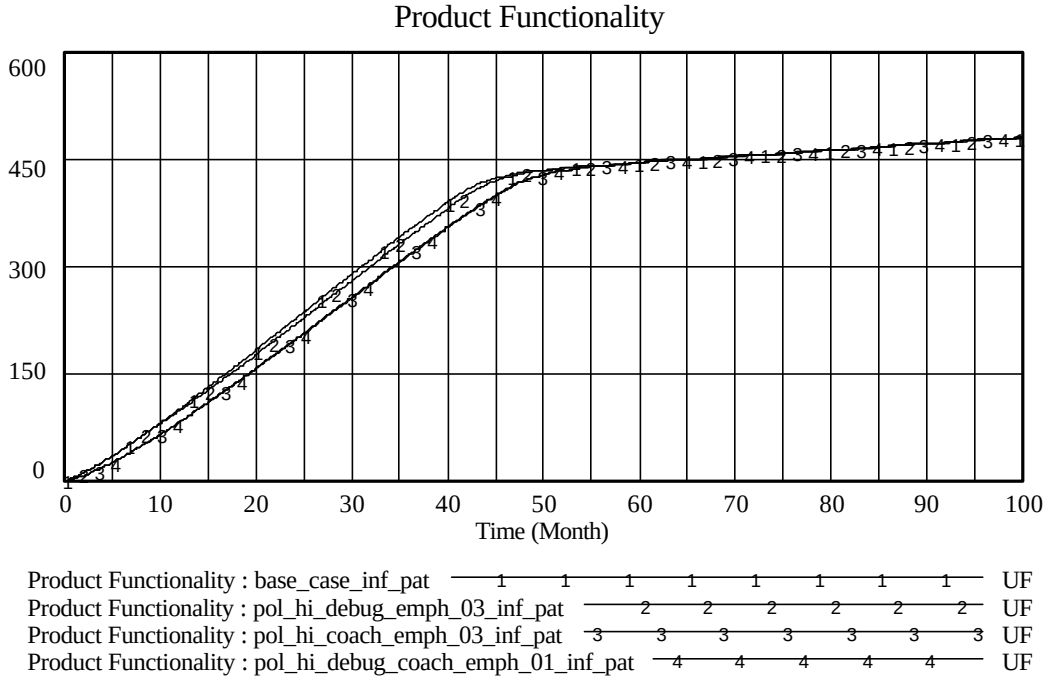


Figure 5.166. Product Functionality for Higher Debugging, Higher Coaching, and Combination Policy Settings under Infinite Patience Assumption

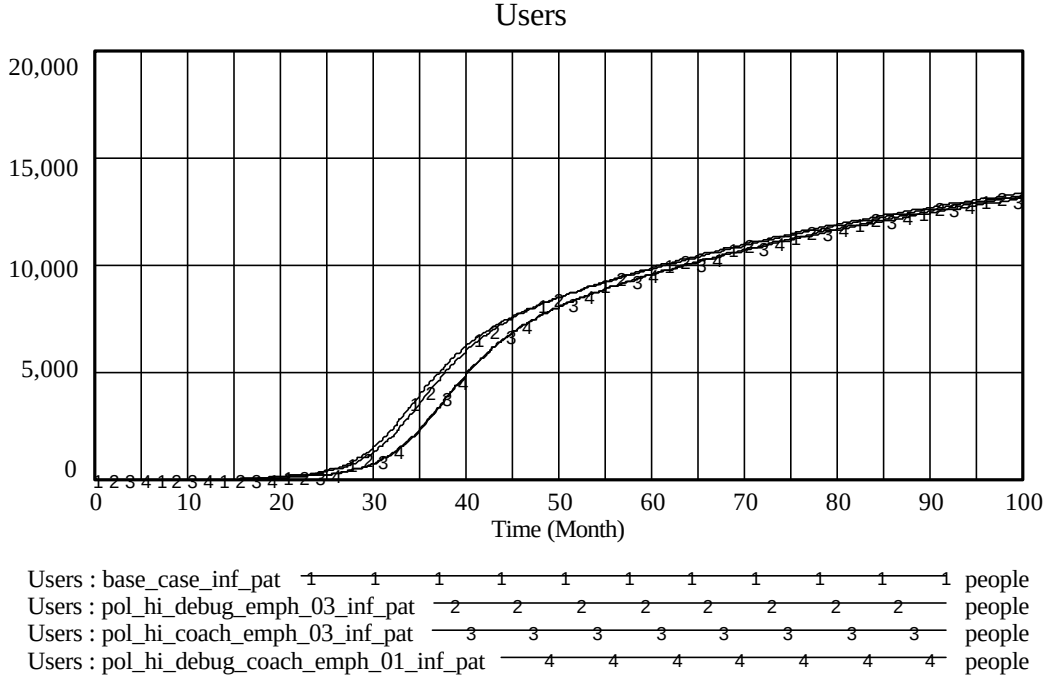


Figure 5.167. Users for Higher Debugging, Higher Coaching, and Combination Policy Settings under Infinite Patience Assumption

Just like under the original diminishing patience assumption, the best policy alternatives under the infinite patience assumption were the pure higher barriers to entry and the combination of higher debugging and higher coaching emphases options. (See Section 5.5.7.) The combination of these two policy options was also replicated under infinite patience assumption. Once again, the overall combination policy yielded better results in terms of both perceived product quality and average developer talent than those of its components. (See Figure 5.168 and Figure 5.169.) The product functionality and community growth were slower under the combination policy, just like they were under the original diminishing patience assumption. (See Figure 5.170 and Figure 5.171.)

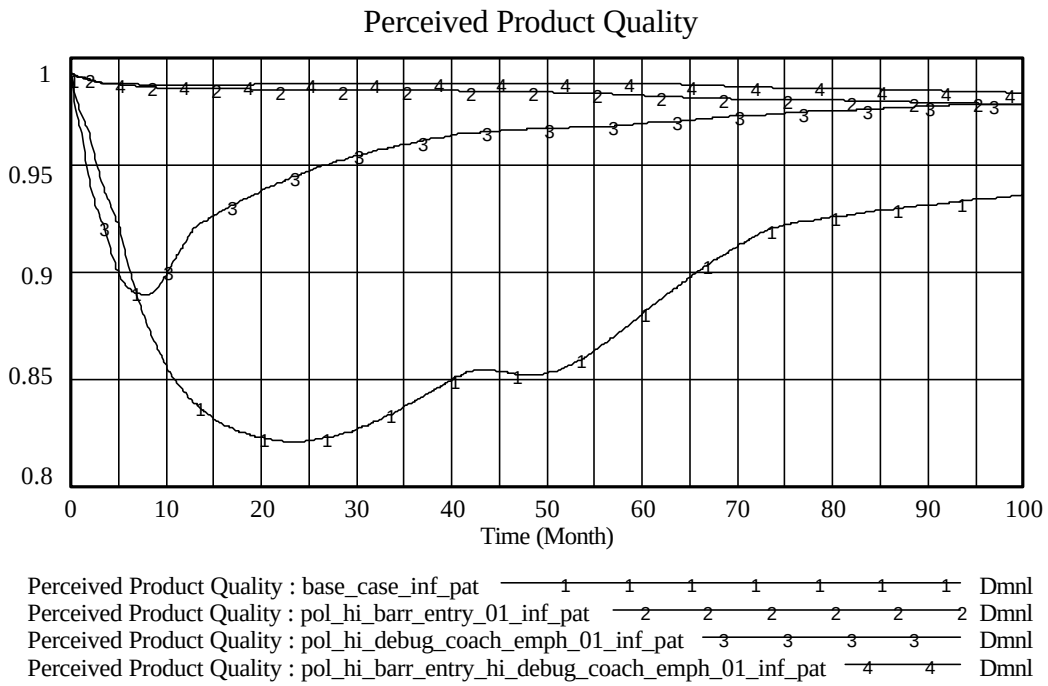


Figure 5.168. Perceived Product Quality for Barriers to Entry, Debugging and Coaching, and Combination Policy Settings under Infinite Patience Assumption

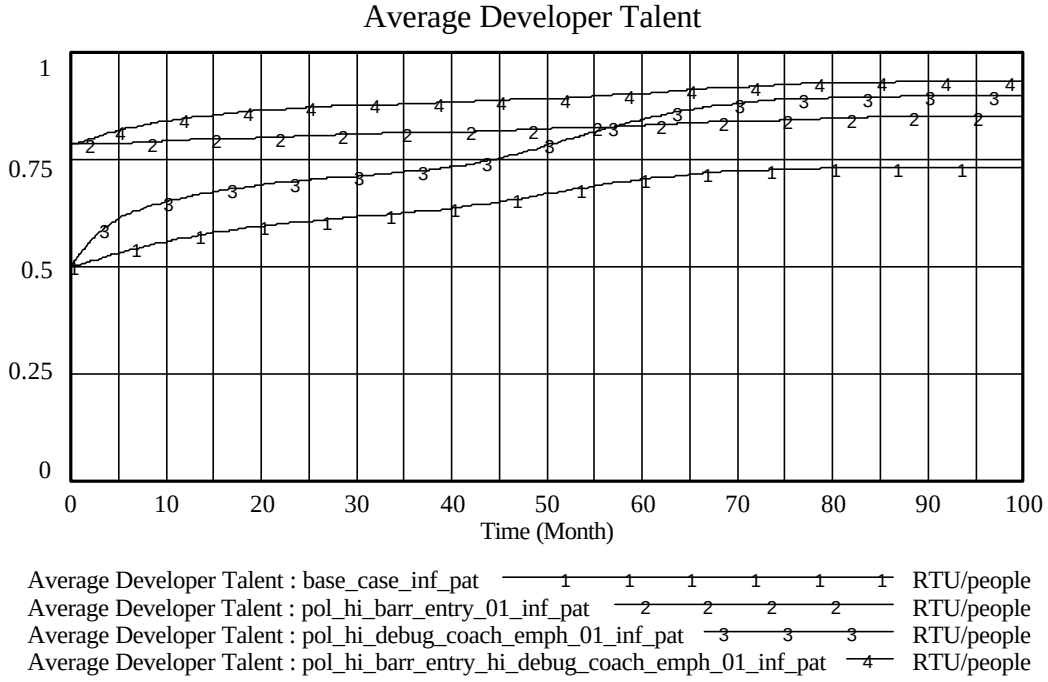


Figure 5.169. Average Developer Talent for Barriers to Entry, Debugging and Coaching, and Combination Policy Settings under Infinite Patience Assumption

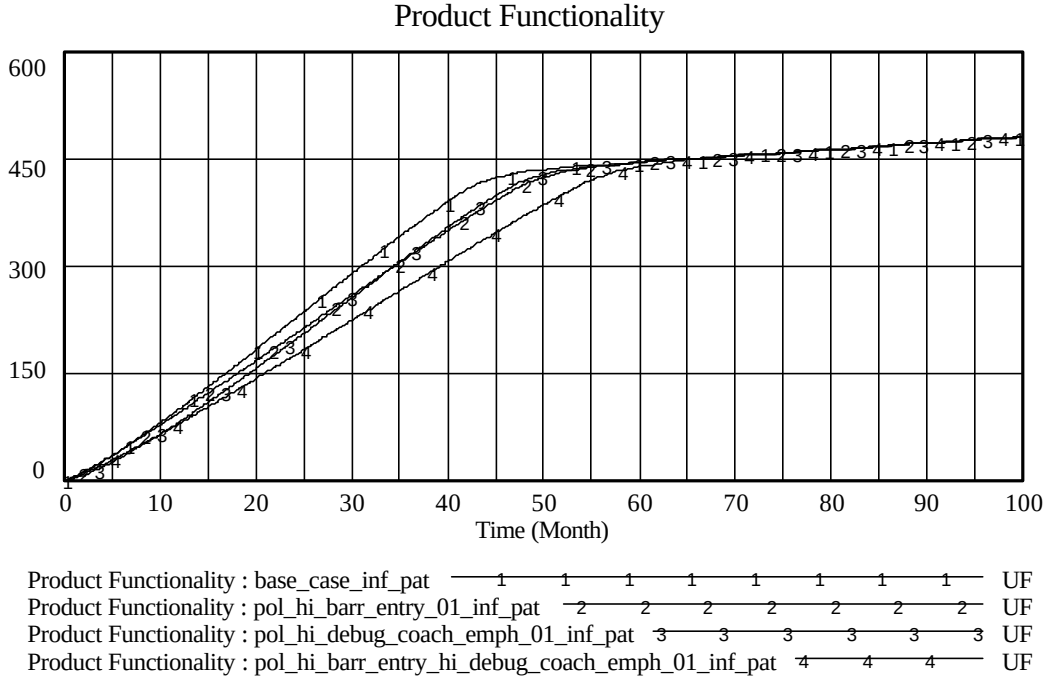


Figure 5.170. Product Functionality for Barriers to Entry, Debugging and Coaching, and Combination Policy Settings under Infinite Patience Assumption

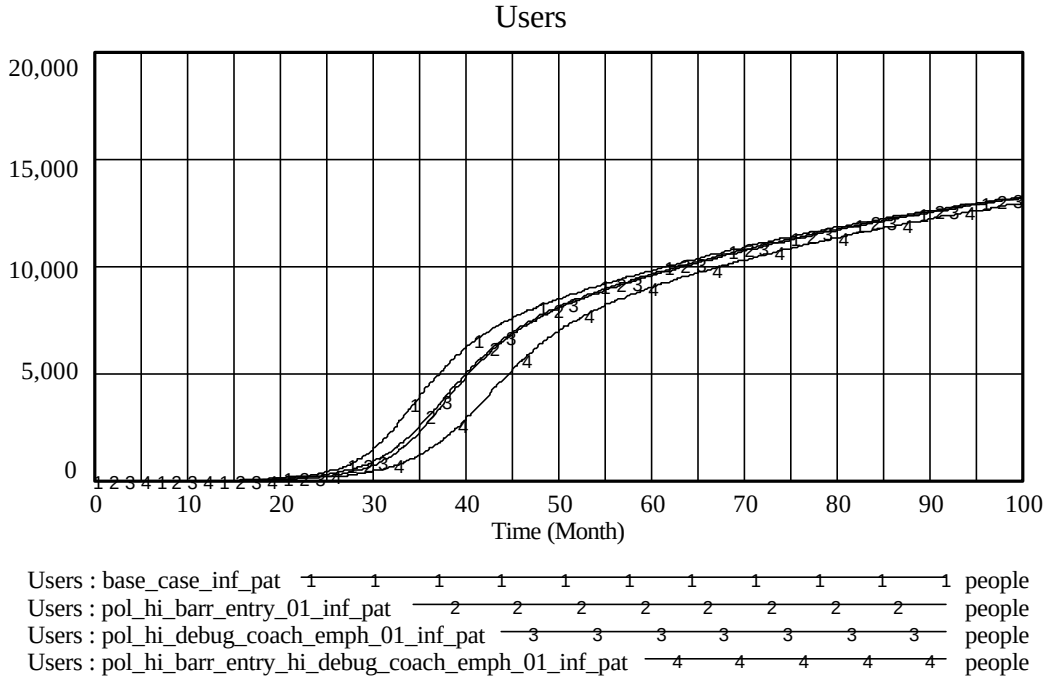


Figure 5.171. Users for Barriers to Entry, Debugging and Coaching, and Combination Policy Settings under Infinite Patience Assumption

The results of the comparisons among the policy options under infinite patience assumption were not different than those under the original diminishing patience assumption. The best policy options were higher barriers to entry, combination of higher debugging and coaching emphases, and an overall combination of these two policy options. However, the implications about certain individual policy options were rather different than those under the original assumption. The bifurcation observed for higher levels of barriers to entry and barriers to contribution under the original assumption was not observed at all under infinite patience assumption. This finding indicated that the assumption about the patience factor could affect the outcomes of the policy interventions.

All of these findings indicate that the existence of a diminishing patience level, which drives the expectations about product functionality, is a key assumption of the OSSD model. Furthermore, the values of the parameters that drive the patience factor, such as normal time to lose patience may affect how the model behaves under different parametric conditions and different policy settings. This leads to the conclusion that the values of such parameters should be estimated very accurately in order to achieve an acceptable level of confidence in the model. This would be a crucial antecedent to drawing implications for real life applications from the findings of the model, especially from the policy runs. The challenge of estimating an accurate value for such parameters in the OSSD model was noted as a potential future research topic.

Another important implication of the analyses on the patience factor is that managing patience and expectations in an open online collaboration community can provide considerable leverage as a policy. The leaders of such communities can sustain

the attractiveness of their communities for exiting and potential contributors and users by maintaining a healthy level of expectations, which is neither too high not too low compared to the realities of the community. While unrealistically high expectation would cause disappointments among the members of the community and lead them to leave the community, low expectations would decrease the attractiveness of the community for potential members, and may have a decreasing effect on the motivation of the existing contributors.

On the other hand, rivals of such communities can employ tactics that would decrease the patience level within the community and increase or decrease expectations beyond realistic limits in order to impede the growth of the community and hurt its ability to develop products. The software development world have witnessed allegations about proprietary software companies trying to impede the growth of open source software development communities (Valloppillil, Cohen and Raymond (annotations) 1998, Valloppillil and Raymond (annotations) 1998). Although it would be very interesting and insightful, a detailed study of the implications discussed in this and the previous paragraphs are obviously beyond the scope of this dissertation. However, such a study was noted as a potential topic for future research, as well.

The following chapter summarizes the findings of a series of interviews carried out in order to test whether the structure of the OSSD model and the policy implications discussed in this chapter can be applied to actual open online collaboration communities. The policy options tested on the model were introduced to the interviewees as pure policy options only, and not in combination with one another. Barriers to entry policy option was introduced to the interviewees as “Selecting New Inexperienced Authors”. Barriers

to contribution was introduced as “Filtering New Material”, higher debugging emphasis as “Reviewing and Editing Existing Material”, and higher coaching emphasis as “Coaching Existing Inexperienced Authors.”

The interviewees were then asked whether they observed similar polices implemented in their community, and if so what the consequences of such policies were, or if not, what they thought the potential consequences of such policies would be in case they were implemented. The interviewees also compared the policy options based on their potential positive and negative consequences. At that stage, the interviewees discussed about combination policy options, as well as the pure policy options.

CHAPTER 6 -- INSTRUCTIONAL MATERIAL DEVELOPMENT - THE CASE OF SYSTEM DYNAMICS K THROUGH 12 COMMUNITY

6.1. Analysis of the Interviews

In this study, the main function of the interviews was to test the applicability of the hypothetical open source software development (OSSD) model to the case of a specific instructional material development community. Accordingly, the interviews were analyzed in order to see whether the personal observations and mental models of the interviewees supported or refuted the assumptions and the structure of the model. The interviews were analyzed in the order in which the subjects were interviewed. The analysis involved testing the main reinforcing and balancing (limiting) loops in the model, the assumption of the underlying policy problem for the community, and the policy options that had the potential of addressing that policy problem. Please refer to Appendix A, Sections 5 through 7 for the worksheets and the diagrams used during the interviews, and the complete interview protocol.

The interviews tested the main reinforcing and balancing loops, which are discussed in Chapter 4. Loops of secondary importance, namely the Reinforcing Loop 1 ("Positive Network Externalities Effect Attracts More Users"), and the Balancing Loop 3 ("More Functionality Makes It Harder to Add Further Functionality") were omitted from the informed portion of the interview. This was done in order to simplify the communication about the model between the interviewer and the interviewees and to help interviewees comprehend the model within the short span of time the interviews allowed.

Another tactic used in order to simplify communication and increase comprehension was the omission of certain outflows associated with the main stocks of

the model. “Leaving Authors” and “Leaving Users” were the two outflows omitted in the diagrams presented to the interviewees. (See Appendix A, Section 6 for the diagrams.) Another reason for omitting these outflows was to elicit interviewees’ observations and mental models with the least possible amount of interference caused by exposing them to an existing model. As discussed elsewhere in this chapter, several interviewees suggested the existence of the omitted outflows. This provides stronger support to those components of the model than having the interviewees approve them after being exposed to diagrams that include those components.

Omitting the outflows of leaving authors and leaving users kept several reinforcing and balancing loops that work through these outflows out of the diagrams shown to the interviewees. Namely, Reinforcing Loop 4 (“More Functionality Retains More Existing Developers”), Reinforcing Loop 5 (“More Functionality Retains More Existing Users, and That Attracts More New Developers”), and Balancing Loop 2 (“Fewer Opportunities for Contribution Retain Fewer Existing Developers”) were excluded due to omitting the outflows. However, each of these three loops, which work through outflows, has a symmetrical loop that works through a corresponding inflow. Accordingly, the dynamic effects delivered to the corresponding stocks by these omitted loops were tested in an indirect way, via the corresponding loops that work through the inflows. Omitting these loops provided a way to elicit the interviewees’ observations and mental models with the least amount of interference, in addition to simplifying communication and facilitating comprehension. Once again, several interviewees suggested the existence of the omitted loops, albeit sometimes slightly different than they

were originally conceptualized, thus providing support to those specific components of the model.

Another step of the testing process was to ask the interviewees about the underlying policy problem of the community with respect to developing instructional materials. The underlying policy problem of the hypothetical OSSD model was identified as a tension between producing content and maintaining quality. The interviewees were asked to elaborate on whether they have observed that problem in the community, and to what extent.

The last step involved the testing of the policy options outlined in the model. At this stage, the interviewees were exposed to four series of diagrams about the four policy options (See Appendix A.6.), and were asked to comment on whether they observed the application of those policy options within the community and the consequences of the policy options that were applied. In cases where an interviewee suggested that certain policy options had not been applied to the community he or she was asked about the possible consequences of those policy options if they were applied.

6.2. Analysis of the Loops

6.2.1. Reinforcing Loop 3 ("More Functionality Attracts More Authors")

The first loop discussed with the interviewees was Reinforcing Loop 3. Figure 6.1 displays this loop as it was shown to the interviewees.[4] The explanation that accompanied the sketch for Balancing Loop 3 was as follows:

[4] The loops were introduced to the interviewees as a series of diagrams building on top of each other. In this chapter only the final (complete) diagram for each loop is shown. Please refer to Appendix A, Section 6 for the full set of diagrams. Also, the references to partial diagrams in the explanations that accompanied

"Here, participating authors produce content in the form of documents, models, visuals, etc. and thus add new functionality to the teaching materials collection. Here, functionality means a general level of usefulness of the materials for teaching purposes. As new functionality is added, functionality of the materials approaches the level expected by possible users, and thus functionality achievement increases. Increased functionality achievement increases the attractiveness of participation for authors, and thus new authors become active in the community faster."

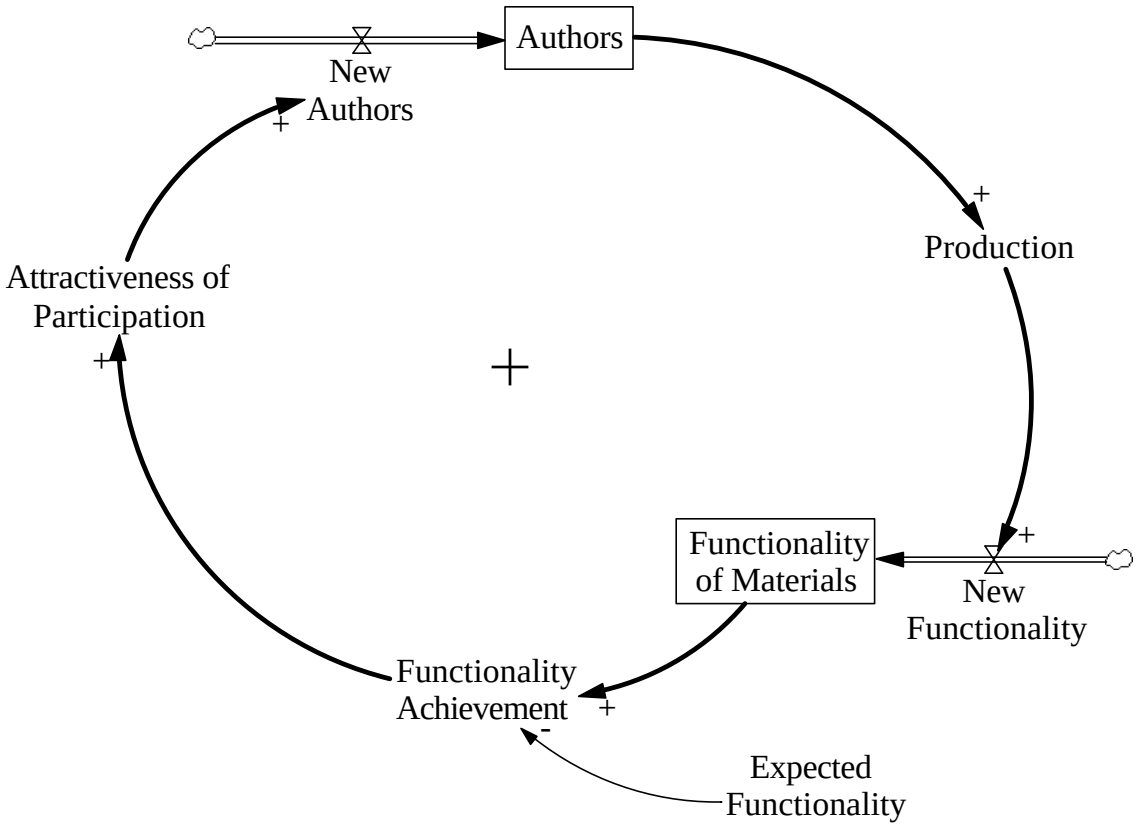


Figure 6.1. Reinforcing Loop 3 ("More Functionality Attracts More Authors") as Shown to the Interviewees

---

the diagrams were edited out for this chapter. For the complete explanations with references to the partial diagrams see Part II in the interview protocol in Appendix A, Section 4.

After this explanation, the interviewees were asked whether they thought such a positive loop reinforces the growth of the number of authors, and the level of functionality of the materials in the case of their community. Table 6.1 summarizes the key comments made by the interviewees about this loop. In Table 6.1 and in the tables from there on the interviewees are listed based on randomly assigned numbers, independent of interview order, name, or other factors, in order not to disclose the identities of the interviewees.

Table 6.1. Key Comments from Interviewees about Reinforcing Loop 3 ("More Functionality Attracts More Authors")

Respondent[5]	**Key Comments**[6]
Interviewee 1	Yes.
Interviewee 2	Yes. It reaches a plateau though.
Interviewee 3	"I am not sure about the applicability of this reinforcing loop [to this community.]" No.
Interviewee 4	Make sense in general. However, the authors do not come from a cloud; they come from users. So potential users become users, and some of those become authors. {This discussion took place before the Users stock was introduced to the interviewee.} There is also attrition, people that leave.
Interviewee 5	"Probably." Not the most important one though.
Interviewee 6	"Not quite the same as my mental model."
Interviewee 7	Yes. {Discussed about users becoming authors.}
Interviewee 8	{Did not comment on this loop.}
Interviewee 9	There has to be some kind of quality control mechanism for this loop to work.
Interviewee 10	{Questioned the link from functionality achievement to attractiveness of participation.} Functionality achievement attracts users, and some users become authors. {This discussion took place before the Users stock was introduced to the interviewee.}

[5] Interviewees 1,3,5 and 6 were affiliated with the same organization. Interviewees 2 and 9, and Interviewees 7, 8 and 10 were also affiliated with two other organizations within the overall community, respectively. Interviewee 4 worked mostly independently. No other details are given about the relationships between the interviewees in order not to reveal their identities.

[6] Notation for comments: Direct quotations are given in quotation marks. Ellipsis dots denote words edited out due to redundancy. Words and phrases in straight brackets were added to the direct quotations for clarification. Curly brackets denote explanations about the comments.

Although only two interviewees [3, 6] challenged this loop, the other interviewees' support for the loop was not very strong. Two other interviewees [4, 10] initially challenged the causal link from the attractiveness of participation to the new authors. Through discussion, the root cause of the challenge was found to be the idea that new authors do not come from outside of the community (represented with a cloud), but from the existing users. In fact, that was the strongest challenge about this loop. Three interviewees [4, 7, 10] explicitly argued along the lines of this idea. The idea of new users coming from existing users was discussed further within the context of the next loop.

6.2.2. Reinforcing Loop 2 ("More Functionality Attracts More New Users, and That Attracts More New Developers")

The interviewees examined the sketch displayed in Figure 6.2 about Reinforcing Loop 2, accompanied with the following explanation:

> "…a higher level of functionality achievement attracts more users. … [and] a higher number of users increases the attractiveness of participation for the authors, thus attracting more new authors."

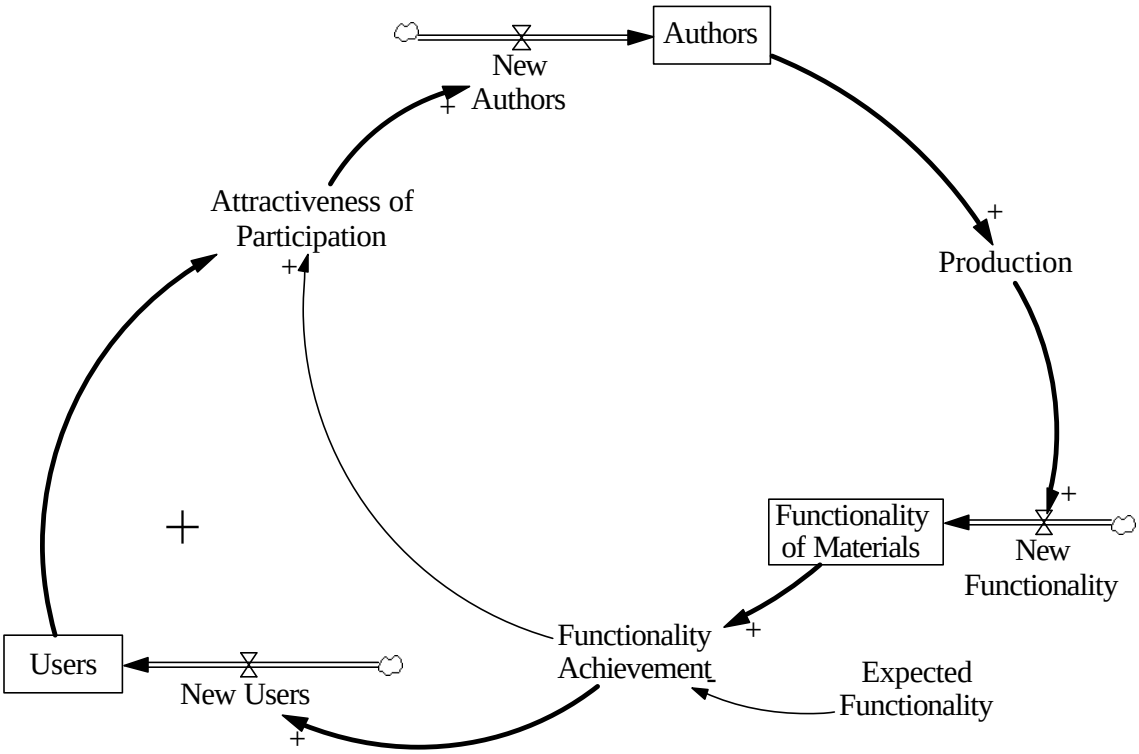


Figure 6.2. Reinforcing Loop 2 ("More Functionality Attracts More New Users, and That Attracts More New Developers") as Shown to the Interviewees

Again the interviewees were asked whether they observed such a positive loop reinforcing the growth of their community. The key comments made by the interviewees about this loop are summarized in Table 6.2.

Table 6.2. Key Comments from Interviewees about Reinforcing Loop 2 ("More Functionality Brings More Users, and More Authors")

Respondent	Key Comments
Interviewee 1	There is no feedback from the users. The link from users to attractiveness does not hold. If there were feedback this loop would work.
Interviewee 2	Yes. However, it reaches a plateau. These two [together with the previous loop] are the most important reinforcing loops, the leverage points.
Interviewee 3	In theory yes; however, there is no feedback available to authors about users. (This causal link does not exist. Theoretically, it would if there were feedback mechanisms from users to authors.)
Interviewee 4	Yes. Don't forget the attrition. There is an outflow from users.
Interviewee 5	Theoretically, but such a feedback does not exist. More users means more users becoming authors. This shoul dbe represented as a stock-flow structure. Authors don't come from a cloud, but only from existing users. That is a stronger loop. {At this point, argued against the previous loop.}
Interviewee 6	The link from functionality achievement to users works. The link from the number of users to attractiveness is questionable.
Interviewee 7	Authors come from users, not from a cloud. {Forcefully argued.}
Interviewee 8	{Did not comment on this loop.}
Interviewee 9	"[The link from functionality achievement to users] would work pretty well, provided that the materials are of high quality." {About the link from users to attractiveness for authors:} "Yes, that would work."
Interviewee 10	Yes.

The main challenge against this loop, argued by four interviewees [1, 3, 5, 6], was that the causal link from the number of users to the attractiveness of participation does not exist. Apparently, the Creative Learning Exchange (CLE) website, which disseminates the instructional material, did not track the number of visitors and downloads in a manner that is visible to the authors. The argument here was that authors could not gather any information regarding the number and characteristics of the users and thus the number of users could not have any effect on the attractiveness of

participation. However, most interviewees [2, 3, 4, 5, 6, 7, 9, 10], including three of those that question the link from the number of users to the attractiveness of participation, suggested the existence of a reinforcing loop that involved functionality, number of users, and number of authors. One plausible explanation here was the argument that new authors came from the stock of existing users. In fact, a fourth interviewee [5] explicitly stated that argument, in addition to the three interviewees [4, 7, 10] who suggested the structure within the context of the previous loop. Those three interviewees [4, 7, 10] repeated their opinion again, and more forcefully within the context of this loop. Apparently, the interviewees observed a "material" type of causal link between the number of users and the number of authors, rather than an "information" link.

6.2.3. Balancing Loop 1 ("Fewer Opportunities for Contribution Bring Fewer Authors")

Figure 6.3 displays Balancing Loop 1, as shown to the interviewees. The following explanation accompanied the sketch:

> "Here as the materials approach the expected level of functionality, opportunities for contribution decrease. Due to decreased opportunity, a smaller number of new authors are attracted to participate."

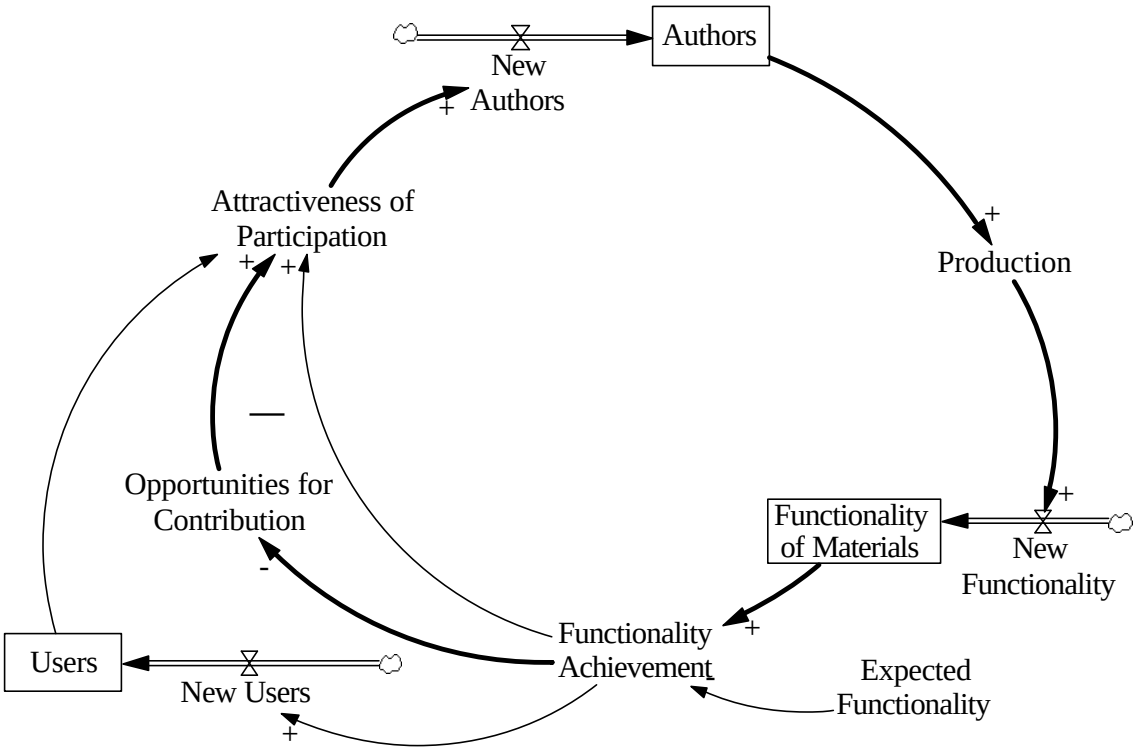


Figure 6.3. Balancing Loop 1 ("Fewer Opportunities for Contribution Bring Fewer Authors") as Shown to the Interviewees

The interviewees were asked whether they observed such a negative loop limiting the growth of their community, or whether they thought such a loop may become dominant in the future. The key comments from the interviewees are summarized in Table 6.3.

Table 6.3. Key Comments from Interviewees about Balancing Loop 1 ("Fewer Opportunities for Contribution Bring Fewer Authors")

Respondent	Key Comments
Interviewee 1	"I don't see that." There are infinite applications. Each lesson can be presented in a new, and different way.
Interviewee 2	Not at this point. "I would hope so." "That [would mean] we have been successful." "50 years down the road, I can see that happening, not sooner."
Interviewee 3	In theory yes. However the community is so far from that.
Interviewee 4	Curriculums change all the time; so the limit on functionality is not constant, but rather a moving target. It changes, just like the achieved functionality level.
Interviewee 5	"I don't think we are anywhere close to that." This will not happen in the foreseeable future.
Interviewee 6	Probably an accurate loop. However, functionality achievement right now is low enough that this loop is not dominant at the time being, and for some time it will not be even remotely dominant. "There are lots of opportunities out there for people to be doing things. [We have not] come close to saturating the domain yet."
Interviewee 7	"We are so far from it, it doesn't have much effect now…. As the gap closes it will have a greater…effect."
Interviewee 8	Not at this time. This may happen in the future. "There are only so many lessons you can write."
Interviewee 9	"I think it could… Most people get the same beginning ideas often, and if there are materials already out there they won't know that they can contribute until they reach a higher level of functionality. [This] makes sense to me."
Interviewee 10	"We are [about]…three decades away from seeing that happen." "That may end up being true, but… it will be so long from now, you can barely even think about it."

Almost all the interviewees, with the exception of one [1], said that Balancing Loop 1 represented a theoretically plausible structure. However, all of them concurred that their community was too far from such a saturation point, where a low level of opportunities for contribution would decrease the attractiveness of participation for the authors. Consequently, they suggested that the loop had no effect on the growth of the community, at the time being. One interviewee [1] suggested that no such limit on

functionality exist, even on a theoretical level, since there are infinite ways to express the same curriculum components. Another interviewee [4] suggested that the limit on functionality would be a "moving target," which changed through time, and thus it would be hard to catch it even over a long period of time. This suggestion reflected the "increasing limit on functionality" assumption, which was used in the OSSD model, but omitted from the interview sketches for simplification purposes.

6.2.4. Balancing Loop 4 ("More Errors Bring Fewer Authors")

Next the interviewees were asked whether they observed a loop similar to Balancing Loop 4 in their community. The following explanation accompanied the sketch displayed in Figure 6.4:

> "… as authors produce content and add functionality to the materials, they also generate errors or weaknesses in the materials… the number of errors decrease the perceived quality of the materials. … A decreased perception of quality decreases the attractiveness of participation for the authors, thus forming another negative loop."

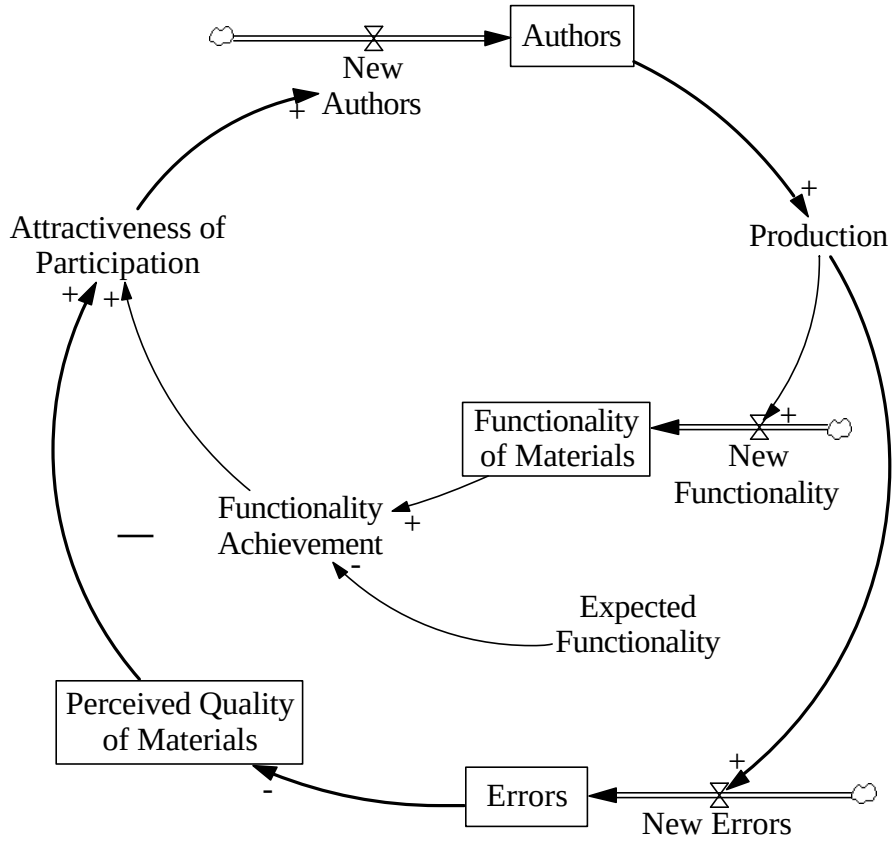


Figure 6.4. Balancing Loop 4 (“More Errors Bring Fewer Authors”) as Shown to the Interviewees

The interviewees were asked whether they observed this balancing loop (which works through errors and weaknesses in the materials) limits the growth of their community. A summary of key comments by interviewees is given in Table 6.4.

Table 6.4. Key Comments from Interviewees about Balancing Loop 4 ("More Errors Bring Fewer Authors")

Respondent	Key Comments
Interviewee 1	"I don't know if it limits the number of existing authors, [and] how that would work." This would not affect existing authors, because they are the ones who generate the errors in the first place. However, it might affect the potential authors' willingness to join. {Also mentioned that it might affect users.}
Interviewee 2	{Chuckled} "Have you been taping all the conversation over the last three years?" Yes. {Strongly supports.}
Interviewee 3	"I don't see that loop operating [in this community.]"
Interviewee 4	{About the link from production to errors:} "I think that happened. I can see that happening. And it did happen. At least the perception was that it was happening. That was why there has been more standardization and quality control at the CLE and Waters Foundation level." That was true in the past, but not any more. The quality is very good now.
Interviewee 5	A decreased perceived quality might increase the number of people who want to contribute, because they want to make it better. So, one can argue both ways. This again works through users.
Interviewee 6	It has the potential. If the ratio between errors and functionality stays the same the perceived quality would not change. The effect of quality would work more through users rather than authors.
Interviewee 7	For certain individuals, yes. As a general dynamic, no. If there is no structure to correct errors, this may have an effect.
Interviewee 8	This is not a very strong loop. However, if the perceived quality stays low for a long time, there may be a problem there.
Interviewee 9	"There you have it." "Absolutely."
Interviewee 10	This loop does not hold for this community, since there are mechanisms to improve quality. Without such mechanisms this loop might hold.

Although three interviewees [2, 4, 9] supported this loop (two of whom [2, 9] rather strongly), the others were skeptical about its existence, or at least its relative power within the overall system. Three interviewees [1, 5, 6] from the skeptical group suggested that this loop again works through users rather than existing authors. Their argument was that perceived quality would affect the number of users, and since new authors should

come from the stock of users, that would eventually have an affect on the number of authors; however, not directly through an attractiveness factor as portrayed in the sketch. An interesting argument made by an interviewee [5] was that a decreasing quality level might motivate more people to become authors, in order to help increase the quality level. The arguments about this loop support the alternative structure, which emerged from the discussions about Reinforcing Loop 3 and Reinforcing Loop 2, where new authors come from the stock of existing users instead of from outside of the model.

6.2.5. Balancing Loop 5 ("More Errors Bring Fewer Users, and Fewer Authors")

The last loop shown to the interviewees was Balancing Loop 5, as shown in Figure 6.5. The following explanation accompanied the sketch:

> "…a decreased Perceived Quality of Materials has a decreasing effect on the number of new users, thus forming another negative feedback loop."

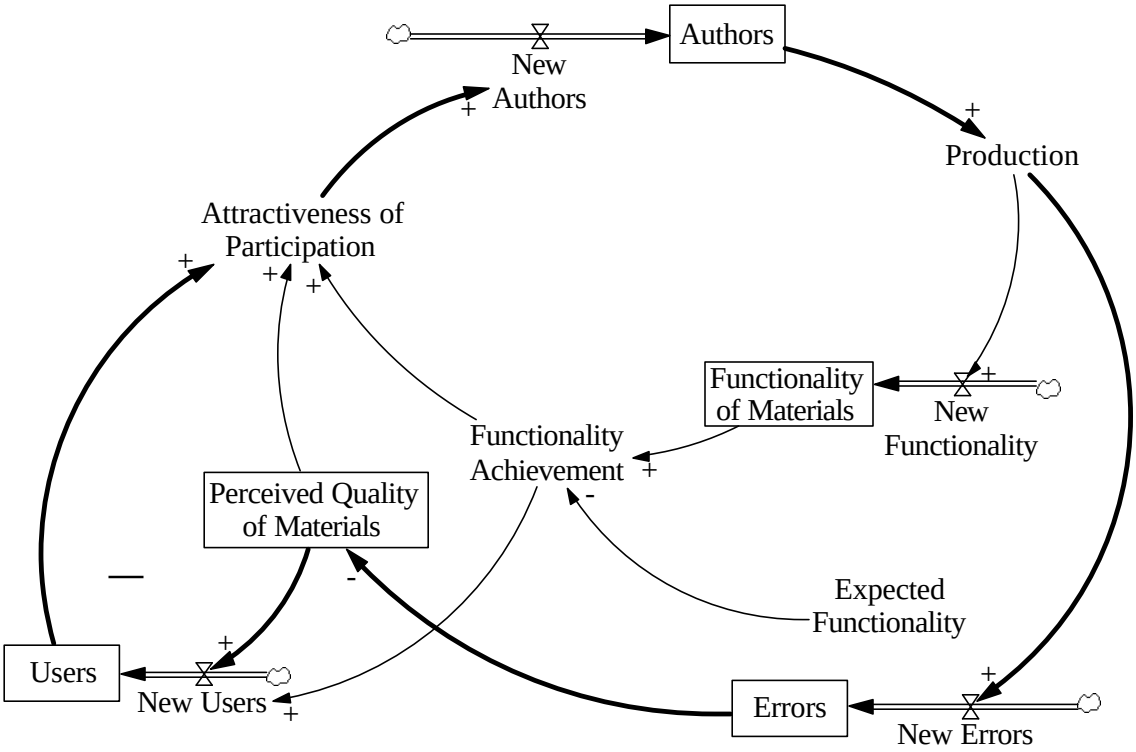


Figure 6.5. Balancing Loop 5 (“More Errors Bring Fewer Users, and Fewer Authors”) as Shown to the Interviewees

Once again the interviewees were asked whether they observed such a balancing loop limiting the growth of their community. Table 6.5 summarizes the key comments by the interviewees.

Table 6.5. Key Comments from Interviewees about Balancing Loop 5 ("More Errors Bring Fewer Users, and Fewer Authors")

Respondent	Key Comments
Interviewee 1	Yes. "That actually, happened… Teachers would say to me 'I don't bother to go to the CLE [website.] There is so much junk there. I don't have time to waste through all that stuff while I'm trying to look for good stuff.'"
Interviewee 2	"Yes, yes, yes. Absolutely." Substantial, however anecdotal, evidence suggests that low quality material turned users off.
Interviewee 3	This may happen in the future. It doesn't happen at this moment. {Later on.} "OK. Now I see the argument. The combination of the perceived quality and functionality do in fact define the new users. I'll buy this."
Interviewee 4	The bigger issue here is not the quality itself, but whether the materials are accessible to the users. The current users are not skilled enough to assess and use the existing quality. This is a minor issue. Most users don't even see the errors.
Interviewee 5	Yes. {Discussed about the previous one.} This one is plausible, not the previous one.
Interviewee 6	"[This] works better for me than a direct linkage to attractiveness." "I like [this] better than [the previous one.]"
Interviewee 7	This is similar to the previous loop. Again, without a structure to correct errors, this may have an effect
Interviewee 8	This is a strong loop.
Interviewee 9	"I have not had direct experience of this, but I think this would be true theoretically."
Interviewee 10	Yes.

More interviewees supported this loop than the previous one. Eight interviewees [1, 2, 3, 5, 6, 8, 9, 10] suggested that this loop was plausible, at least on a theoretical level. Some [1, 2] suggested that they had personal observations that this loop exists. One link that was questioned by some interviewees [5, 6] again was the link from the number of users to the attractiveness of participation. The fact that not many interviewees argued against that link this time might be attributed to the fact that they had already made their points within the context of the discussion about Reinforcing Loop 2 and that they were

mostly focused on the link from perceived quality to number of users while discussing this loop. These arguments again point to the alternative structure where new authors come form the stock of existing users.

6.3. Analysis of the Policy Options

6.3.1. Tension between Building Functionality and Maintaining Quality as the Underlying Policy Problem

After the questions about the main loops, the interviewees were asked to comment on the applicability of the main policy problem of the open source software development (OSSD) model to their community. The main policy problem for the OSSD model has emerged as the tension between building functionality fast enough to attract a critical mass of users and contributors, while trying to maintain an acceptable level of quality in order to retain existing users and contributors.

The system dynamics K through 12 community is a multi-faceted entity, which works to propagate system dynamics concepts to K through 12 education. The community works on many fronts, including, but not limited to developing and disseminating instructional material for introducing system dynamics concepts to K through 12 educators and students. Obviously, the community has many different issues and problems related to different facets of their existence and functions. However, since this research studies the community from an instructional material development perspective, the focus is on the policy problems related to that specific facet of the community. Consequently, the interviewees were asked whether they observed the tension between building functionality and maintaining quality as their community's

underlying policy problem with respect to the functions of developing and disseminating instructional materials.

Table 6.6 summarizes the key comments from the interviewees related to this question. Almost all of the interviewees [1, 2, 3, 5, 6, 7, 8, 9, 10] responded that they observed the tension between building functionality and maintaining quality as the underlying policy problem leading to the symptomatic problems related to the development and dissemination of instructional materials within the community. Some interviewees [1, 2, 5, 7] argued strongly in support of this tension being the main problem, which eventually led the leaders of the community to take serious measures in order to improve the quality of the materials without hurting the growth of the materials collections in terms of quantity and functionality.

Table 6.6. Key Comments from Interviewees about the Tension between Building Functionality and Maintaining Quality as the Underlying Policy Problem in the Community

Respondent	**Key Comments**
Interviewee 1	Yes. At first, when CLE tried to get as much material as possible on the website, the quality went down. Then the low quality material is taken off, and this time quantity suffered. Now the quantity does not increase as fast, probably because of the quality control process.
Interviewee 2	"Absolutely."
Interviewee 3	"Yes. I think that's valid."
Interviewee 4	This is not the problem right now. It may become a problem when the community becomes more mainstream.
Interviewee 5	"Yes, of course. We have just tried to address it."
Interviewee 6	"Yeah, I think so."
Interviewee 7	"Yes. This is exactly the way we look at it, too."
Interviewee 8	Yes
Interviewee 9	Yes. Coaching/training could help improve this.
Interviewee 10	Yes.

An interesting theme narrated by several interviewees was how the quality and quantity of the materials collection hosted by the CLE website have changed over time. As mentioned in Chapter 3, the CLE website is the main repository of instructional materials for introducing system dynamics concepts to K through 12 education. According to several interviewees, when the CLE started to gather instructional materials from contributors and disseminate them through their website, they avoided putting a quality control mechanism in place. One of the reasons for that was the concern that the limited number of contributors might be discouraged by such a quality control mechanism. That led to a considerably low level of quality for the general collection, especially in terms of the accuracy of system dynamics concepts, although there were occasional pieces of really high quality. The low quality level was not a big concern back then, since the users of the collection were mostly newcomers to the field of system dynamics, and they were not yet knowledgeable enough to find the small system-dynamics-related errors in the materials. During those initial stages the focus of attention was to build as much quantity and functionality as possible in order to reach a critical mass, which would be useful for many people and thus could attract a high number of users.

However, as the collection grew over time, two important dynamics came into play. On one hand, the users became far more knowledgeable about system dynamics, and started to find and complain about the system-dynamics-related errors. This shifted the focus of attention from quantity and functionality to quality, since the main problem became retaining exiting users as well as attracting new users. Another dynamic that helped shift the focus was the fact that the materials collection had reached a considerable

mass. At that stage, the CLE felt more confident about putting a quality control mechanism in place, even if it meant sacrificing some functionality in order to improve quality.

The first step taken to improve quality was to carry out an audit of existing materials. Three experienced system dynamicists reviewed all of the existing materials and grouped them into three categories according to their quality. One group was those of high quality, which stayed on the collection "as is." The second group consisted of materials that needed slight improvements and updates, which were easily revised and put back in the collection. The third group consisted of materials that needed a considerable amount of rework. These materials were sent back to their authors for revision. Some of these were so low quality that even the authors did not seek to revise them. A mechanism for continued quality control was also put in place. New materials were not directly added to the collection any more, but went through a similar quality assurance process. However, the CLE is still sensitive about not discouraging contributors. They try to keep the "quality threshold" at a level that strikes a balance between improving quality and maintaining an acceptable stream of new materials into the collection. This narrative reflects how the tension between building functionality and maintaining quality can become an important policy problem and shape the policies of the leadership of an open online collaboration community.

6.3.2. Policy Option 1: Filtering New Material

Following the discussion about the main policy problem, the first policy option discussed with the interviewees was filtering new materials produced by inexperienced authors, as shown in Figure 6.6. The counterpart of this policy option in the context of the

OSSD model is the barriers to contribution policy option. The following explanation was presented with the sketch:

> "The first policy option is filtering materials that are produced by inexperienced authors. This option is based on the premises that inexperienced authors generate more errors per production, and by filtering the materials that are produced by inexperienced authors, it may be possible to decrease the number of new errors or weaknesses in materials. … materials produced by inexperienced authors are not added directly to the overall materials produced, but instead diverted to a backlog to be filtered. … a certain portion of this backlog would be accepted and added to the overall production, while the rest is rejected. … filtering would be done by experienced developers, with a certain filtering rate per time unit, and an average rejection ratio would determine the amount of materials that are accepted or rejected. The rejection ratio would depend on the level of scrutiny experienced developers apply during filtering, and thus decrease the number of new errors that go into the materials collection. … a higher rejection ratio, which means a higher scrutiny level, would reduce the number of new errors. … a possible adverse effect of this policy would be decreasing motivation for production on the part of the inexperienced authors. It is possible that as the rejection rate increases, motivation for producing materials would decrease. … another adverse effect of this policy [might be that materials] produced by experienced authors would decrease, since they would dedicate a portion of their time to filtering."

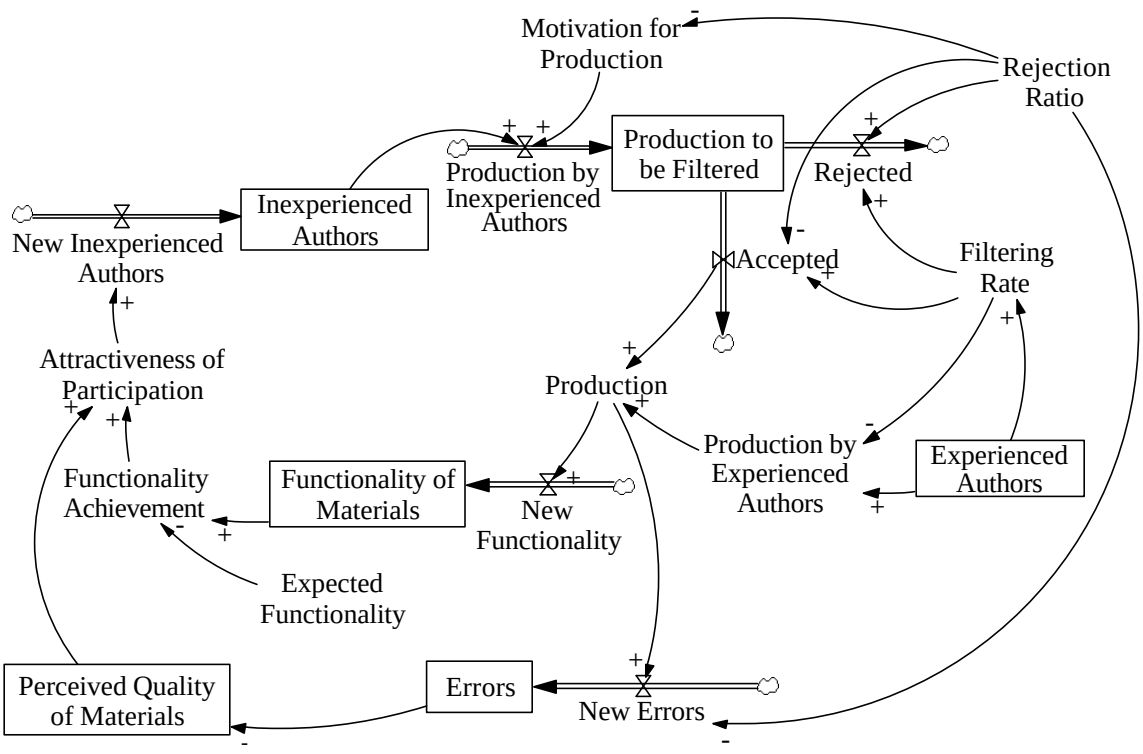


Figure 6.6. Policy Option 1: Filtering New Material as Shown to the Interviewees

The interviewees were asked whether they observed a similar policy implemented in their community, and if so what the consequences were. Table 6.7 summarizes the key comment by the interviewees.

Table 6.7. Key Comments from Interviewees about Policy Option 1: Filtering New Material

Respondent	Key Comments
Interviewee 1	{Discussed about diverting the production from inexperienced authors to a backlog to be filtered.} ”Yes. That’s what we do now.” {About the two adverse effects:}“We observed all of that.”
Interviewee 2	CLE applies this. For a period, submissions were down. This may be due to the decrease in motivation or other factors. CLE has a three-tier policy, including rework. With just an accept-or-reject policy average quality would increase but quantity would suffer.
Interviewee 3	Yes. A decrease in motivation could happen, but rejection rate is not that high now. Decreased production by experienced authors argument does not hold, because the materials to be filtered and reviewed and edited are negligible in number at CLE.
Interviewee 4	“Yes; this is what [CLE] is doing.” Motivation decrease would in fact be a problem. When the community is not mainstream rejecting is more risky. Later on when the community becomes more credible and mainstream that might become easier to do. At this stage re-writes would work better. Inexperienced authors need coaching, not rejecting. Decrease in experienced authors’ own production happens. But it could be made positive, if filtering [and reviewing and editing] is done together with coaching.
Interviewee 5	“Filtering is what we are doing. But we are filtering from all authors. [Not just from inexperienced authors.]” We have a limited number of editors, so they cannot spend a lot of time on other things. We tried to avoid the motivation decrease. We have not observed the amount of reaction we expected.
Interviewee 6	Not enough filtering is being done to hurt production by experienced authors. “It may become a problem [in the future], I haven’t seen it yet.” The first adverse effect would happen rather in the form of inexperienced authors leaving the community, rather than their contribution level decreasing.
Interviewee 7	“[Our policy] is exactly this…[However,] we don’t differentiate between experienced and inexperienced authors. [All work is filtered.]” {Refers to a filtering and editing type of policy.}
Interviewee 8	Yes. Combined with reviewing and editing. Motivation decrease is not observed, but makes sense theoretically. Decrease in experienced authors’ production is observed.
Interviewee 9	“This makes a lot of sense to me.” CLE does this. There is no direct observation, but a very high rejection ratio might in fact hinder the production by inexperienced authors. CLE was concerned about that. “[CLE is] trying to find some middle ground.” However CLE also has reviewing and editing.

Table 6.7. Key Comments from Interviewees about Policy Option 1: Filtering New Material (continued)

Interviewee 10	"Absolutely." This is CLE's approach. Waters Foundation combines this with reviewing and editing. There is not enough sample to comment on motivation decrease. Decrease in experienced authors' production is actively avoided. "[We suggest experienced authors to] focus on what [they] are doing, rather than fixing things."

All the interviewees suggested that they had observed some form of a filtering policy being implemented in their community. However, in terms of the implementation they emphasized two important differences between the suggested policy and the real life applications within the community. First, filtering was not implemented as a pure policy, where materials were either accepted as-is or rejected flatly. Whenever some kind of a filtering policy was implemented, it was coupled with a revision extension, so that the materials that are not accepted as-is can be "reworked" by the authors, the reviewers or both.

The second difference suggested was that the community filtered all materials that were submitted, both by experienced or inexperienced authors. This difference was not brought up by as many interviewees as the first difference. However, those [5, 7] who brought it up emphasized it forcefully.

One interviewee [1] suggested that the negative effect of filtering on motivation for production was actually observed whenever a very high quality threshold was used. Most other interviewees [2, 3, 4, 5, 6, 8, 9, 10] suggested that the negative effect was not observed. They attributed that to the fact that the quality threshold, and consequently the rejection rate were not high enough to trigger such an effect. However, even those

interviewees suggested that motivation for participation would decrease if the rejection rate were high enough. In fact, they suggested that the understanding that motivation for production would decrease under a high rejection rate was the reason why the quality threshold and the rejection rate had been kept low. One interviewee [6] suggested that this adverse effect would manifest itself as a portion of inexperienced author leaving the community after being rejected, rather than a decrease in their motivation for production.

The other adverse effect of this policy, namely the decrease in experienced authors' own production due to spending time on filtering, did not receive as much support on a theoretical level from the interviewees as the first adverse effect. However, at least four interviewees [1, 4, 8, 10] suggested that they had observed either a decrease in experienced authors' production or a deliberate effort on the community's part to avoid such a decrease. One interviewee [3] argued that filtering had not hindered the production by the experienced authors who participate in filtering, and that it did not have the potential to do so.

6.3.3. Policy Option 2: Reviewing and Editing Existing Material

Figure 6.7 displays the second policy option, reviewing and editing exiting material, as shown to the interviewees. This policy option is the counterpart of the higher debugging emphasis policy option in the context of the OSSD model. The following explanation accompanied the sketch:

> "The second policy option is reviewing and editing content in order to fix existing errors. … Here again, experienced authors and inexperienced authors build functionality by producing materials and while doing that they generate errors and weaknesses in materials. … experienced authors would spend time on

reviewing and editing content and thus fix a portion of existing errors. … reviewing and editing would decrease production by experienced authors. This decrease would probably be greater than that would happen under the filtering option, since reviewing and editing existing content would take more time than filtering new production."

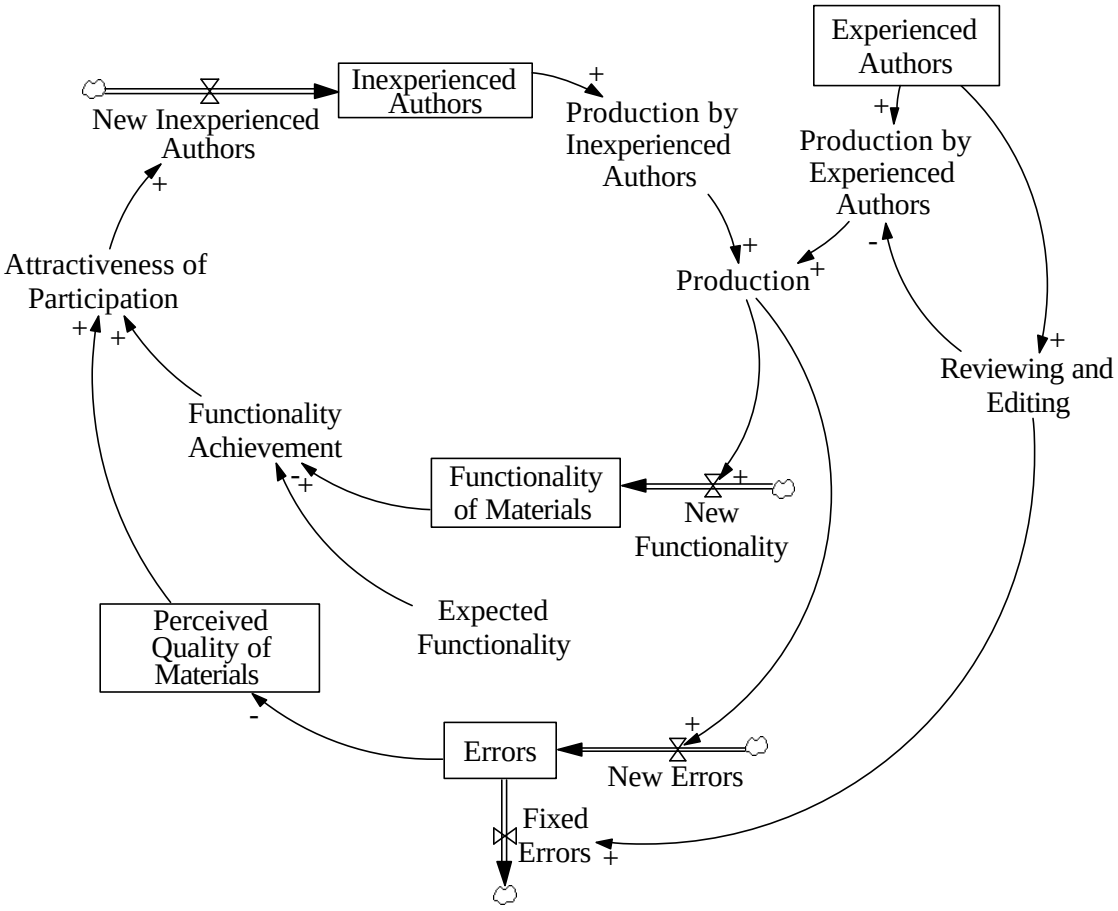


Figure 6.7. Policy Option 2: Reviewing and Editing Existing Material as Shown to the Interviewees

Once again, the interviewees were asked whether they observed a similar policy implemented in their community, and if so what the consequences were. Table 6.8 summarizes the key comment by the interviewees about the second policy option.

Table 6.8. Key Comments from Interviewees about Policy Option 2: Reviewing and Editing Existing Material

Respondent	Key Comments
Interviewee 1	Yes. A combination of filtering and reviewing and editing is implemented.
Interviewee 2	This policy has been implemented intermittently. Now it exist in CLE.
Interviewee 3	Production decrease happened for a while during the general retrospective review at CLE. CC-STADUS must have had that during a period in the past. "It is periodic, …rather than continual." CLE's review process is a combination of filtering, and reviewing and editing
Interviewee 4	Filtering is done together with reviewing and editing. CLE waited for material to gather for a while and then carried out a big filtering/reviewing and editing intervention. Nowadays materials go through filtering, and reviewing and editing as they arrive.
Interviewee 5	Reviewing and editing is combined with filtering. CLE filters, but doesn't spend time on reviewing and editing if the author does not do rework on a piece that is found to be of low quality. If the author does rework, CLE does reviewing and editing, as well."
Interviewee 6	CLE implements a combination of filtering and reviewing and editing. There are four categories for incoming materials: 1) Good enough to publish as is. 2) Requires very little editing. 3) Requires substantial editing. 4) Has no value at all; diplomatically rejected. Filtering does not take too much time, reviewing and editing does.
Interviewee 7	This is not used outside of Waters Foundation. Decrease in experienced authors' production does not hold, since the reviewing and editing load is not too big. If the expectations from the Waters Foundation sites in terms of quantity were higher, that would probably hold.
Interviewee 8	Yes. Combined with filtering. Decrease in experienced authors' production is observed.
Interviewee 9	This is used extensively. "I agree [that reviewing and editing takes more from experienced authors' time.] I don't think you're going to find very many experienced authors willing to do this scenario…. [They] love to write, …and if you take too much of their time reviewing other materials, there has to be a really good compensation for that." Have observed that effect.
Interviewee 10	Combined with filtering. Not enough sample size to comment on the production decrease effect. There is not a lot of material to be reviewed and edited.

All the interviewees suggested that they had observed the policy of reviewing and editing existing material being implemented in their community. Many interviewees reiterated the fact that their community implements the “filtering” and “reviewing and editing” policies in combination. The interviewees suggested two reasons for this combined implementation. First, the community tries to bring out the best in each author and each work, so they encourage revisions based on rounds of reviews and edits in order to help improve the quality of the submitted materials. Second, the community does not want to flat out reject materials, in an effort not to discourage authors. Several interviewees [3, 4, 6, 8] attributed the unwillingness to reject materials to the culture within the overall community of K through 12 educators. They suggested that, as a cultural value, criticism within the general K through 12 community tends to be more indirect, encouraging, and constructive in nature, compared to criticism in an academic setting, which is essentially direct and at times confrontational. To loosely paraphrase an interviewee [4], the skins of K through 12 educators are not as thick as academicians. When filtering is coupled with reviewing and editing, it becomes possible to reject materials indirectly. As one interviewee [5] suggested, a work that is not found to be of merit can be sent back to the author for revision numerous times. In the end it would either become good enough to be added to the collection or the author would give up trying to improve it further without feeling directly rejected.

Some interviewees suggested another difference between the reviewing and editing policy as presented during the interviews and its actual implementation within the community. The policy as presented during the interview assumes a continual reviewing and editing intervention. As the materials are received, they are added to the backlog of

materials to be reviewed, and the experienced authors review the backlog with a certain, continual rate. According to four interviewees [2, 3, 4, 5], the real life implementation of the policy had been periodic, or intermittent rather than continual. One of the four interviewees [5] suggested that the process had become more continual recently, and that they were trying to keep it that way.

Several interviewees supported the hypothesis that this policy would have an adverse effect on the production level of experienced authors who participate in reviewing and editing. Four interviewees [3, 6, 8, 9] suggested that they had observed a decrease in experienced authors production whenever they participated in reviewing and editing. Two other interviewees [7, 10] suggested that they had not observed the adverse effect, and they attributed it to the fact that there were not too many materials to be reviewed. They concluded that if the submissions would increase, maintaining an acceptable amount of reviewing and editing would take from experienced authors' own production time.

6.3.4. Policy Option 3: Selecting New Inexperienced Authors

Selecting new inexperienced authors was the third policy option discussed with the interviewees. The counter part of this policy option in the context of the OSSD model is the barriers to entry policy option. Figure 6.8 displays this policy as shown to the interviewees. The sketch was presented with the following explanation:

> "… the third policy option is selecting new inexperienced authors according to their talents. … Here, new inexperienced authors are not directly accepted into the existing inexperienced authors pool. Rather, they apply and wait to be selected. … selecting is carried out by experienced authors, and of course

some applicants are refused, based on an average refusal ratio. Refusal ratio would be based on the scrutiny level of the selection process. A higher level of scrutiny would mean a higher refusal rate, and that in turn would mean a higher average inexperienced author talent level… One possible adverse effect here would be a decrease in the number of inexperienced authors applying. … as refusal ratio increases, number of inexperienced authors applying would decrease. … another adverse effect of this policy [might be that materials] produced by experienced authors would decrease, since they would dedicate a portion of their time to selecting."

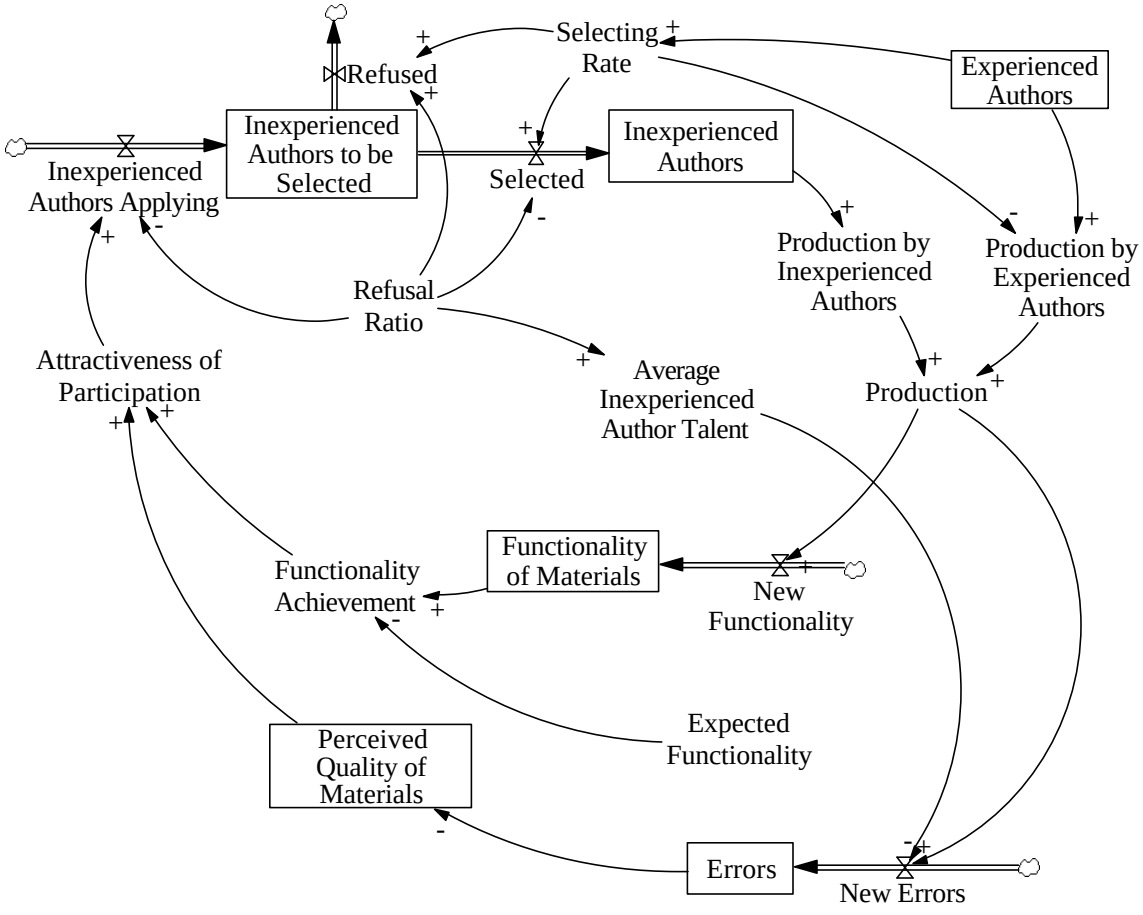


Figure 6.8. Policy Option 3: Selecting New Inexperienced Authors as Shown to the Interviewees

Table 6.9 summarizes the key comments by the interviewees about this policy.

Table 6.9. Key Comments from Interviewees about Policy Option 3: Selecting New Inexperienced Authors

Respondent	Key Comments
Interviewee 1	Authors are not authors until they send something in; and when someone sends something in, they are never refused. Their work would be reviewed and comments would be sent in any case. What may happen is someone sends something in that does not meet the guidelines, and a comment is sent back suggesting following the guidelines. If that person does not work on the submission any further, the person does not become an author; but CLE never refuses people. At least in the current stage, refusing people would not help, because there is not enough participation anyway.
Interviewee 2	No. If this policy is implemented, a reduction in the number of people interested could be observed in short run, and that would have a negative impact. There might be a positive impact in the long run.
Interviewee 3	Why are not there a lot of inexperienced authors joining the community? Is it because experienced authors are not supportive enough? Maybe, maybe not? Along with that, time, willingness, and predisposition for being an author might all be reasons. "In fact, rather than a cloud … we may have a small stock of [potential authors], ... a finite stock."
Interviewee 4	"No, I haven't seen that… I am personally for it. But, I don't know how it would work."
Interviewee 5	No. Because motivation of inexperienced authors to join would in fact decrease. I don't think this should be implemented. CLE has the policy that anybody can submit to its website.
Interviewee 6	CLE is involved not in selecting, but in training novices.
Interviewee 7	Done in a very refined, diplomatic, and covert way. People do not apply formally, however, Waters Foundation site managers do pick people to encourage and support. Motivation decrease has not been an issue so far.
Interviewee 8	No. There is some covert selection, though. Most of the time not even the mentor that does the selecting is aware of this. An overt selection policy would not be a "helpful" policy.
Interviewee 9	"Yes, I have seen this." The best people among one year's workshop participants become next year's instructors. However people don't know about this beforehand, and thus it does not affect the motivation. Production decrease on part of experienced authors is negligible.
Interviewee 10	No. "We are not rejecting people."

Most interviewees [1, 2, 3, 4, 5, 6] suggested that they had not observed a policy of selecting new authors being implemented in their community. Three interviewees [7, 8, 9] argued that although an open selection process had never been implemented, there were covert and subtle processes for selecting new authors. In most cases these processes would be so subtle that even those doing the selecting would not be aware that what they do is selecting. One example given was the process of picking people to support and encourage.

Interviewees expressed varied opinions about the overall usefulness of this policy option. Some [4, 7, 9] suggested that they are in favor of it, since it is an efficient way to determine and encourage the best people to become authors. However, a larger portion of interviewees [1, 3, 5, 6, 8] suggested that they would be against such a policy since it would discourage and alienate most potential authors. One common concern expressed by most of the interviewees, regardless of their being for or against selecting as a policy option, was its potential for decreasing new contributions, either only in the short run, or both in the short and the long run. Thus, most interviewees [1, 2, 3, 5, 6, 8] supported the hypothesized potential adverse effect of an overt selecting policy on the number of inexperienced authors willing to join the community. The three interviewees [7, 8, 9] who argued that covert and subtle forms of selecting were implemented suggested that these implementations had not had a negative effect on the willingness of new authors who want to join the community. However, one of them [8] suggested that an open selecting policy could have such an adverse effect.

The hypothesis that the selecting policy would have an adverse effect on experienced authors' own production was not supported. The interviewees who commented on that potential effect suggested that it would not be substantial.

6.3.5. Policy Option 4: Coaching Existing Inexperienced Authors

The last policy option discussed with the interviewees was coaching existing inexperienced authors, which is the counterpart of the higher coaching emphasis policy potion in the context of the OSSD model. Figure 6.9 displays this policy as shown to the interviewees. The following explanation accompanied the sketch:

> "…Here experienced authors coach inexperienced authors, and … coaching increases average inexperienced author talent gradually over time (with a delay). Accordingly, average inexperienced author talent is defined as a "smooth" in this context. Both experienced and inexperienced authors would dedicate a portion of their time to coaching under this policy. So, … coaching would decrease materials produced by experienced and inexperienced authors, thus affecting the functionality growth negatively in the short run.

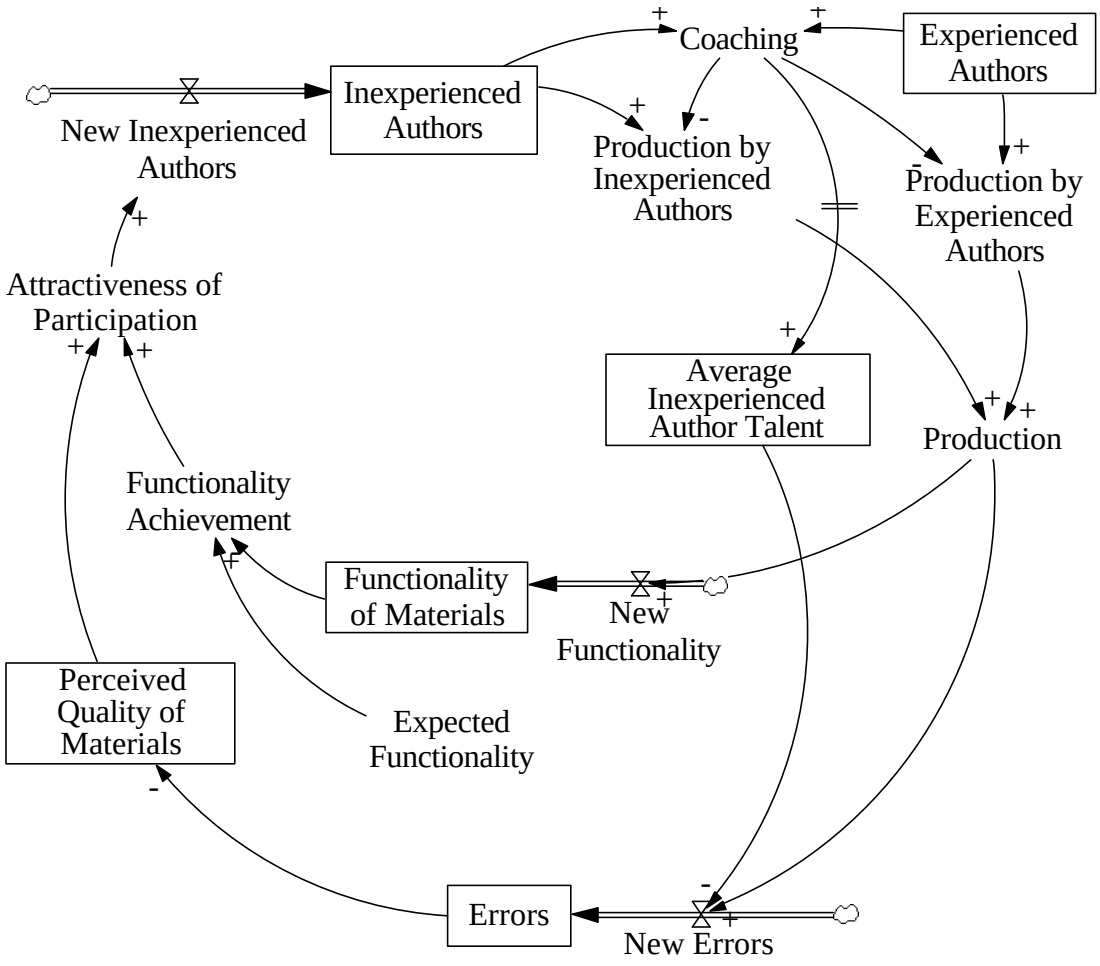


Figure 6.9. Policy Option 4: Coaching Existing Inexperienced Authors as Shown to the Interviewees

The key comments by the interviewees about this policy option are summarized in Table 6.10.

Table 6.10. Key Comments from Interviewees about Policy Option 4: Coaching Existing Inexperienced Authors

Respondent	Key Comments
Interviewee 1	The expectation is that coaching will increase the general talent level, and the number of errors will decrease. Coaching indeed takes a lot of time, even more than reviewing and editing.
Interviewee 2	This is not done systematically. It might have a positive effect in the long run.
Interviewee 3	Yes. However, not everybody does that. It is mostly done in a localized manner. Carlisle site is an example where coaching exists.
Interviewee 4	"There is quite a lot of that. I have probably done [this] more than anything else." Coaching is very time-consuming. It's the slowest, but the safest policy.
Interviewee 5	It is hard to separate coaching from reviewing and editing. Coaching is the only way to increase the talent level across the board.
Interviewee 6	Coaching is a very common strategy. Coaching overlaps with production, It may limit production, but these two activities are not completely divorced.
Interviewee 7	"You are probably getting closer to what happens within [the Waters Foundation] network; and it does slow down the production. But we think that that is probably desirable. The quality will make up for the quantity. You are much better off over the long term."
Interviewee 8	Yes. Coaching increases the talent level of both inexperienced and experienced authors.
Interviewee 9	Time is an issue in making coaching work. However if it works, coaching is very effective.
Interviewee 10	Yes. Coaching raises awareness level. It absolutely helps. "You can consider… the editing process to be some coaching." "It keeps me from going back and revising my own [work]." "It's something of a trade off. I'm taking time away from getting my thing done, but I'm putting more material our there through that other person I'm coaching."

All interviewees suggested that one form or another of coaching was done within their community. They suggested that coaching was mostly done in an unorganized and localized manner. In that sense, coaching was not implemented as a systematic policy, but nevertheless encouraged within the community. Some interviewees [4, 5, 6, 10] suggested that coaching was done in the form of experienced authors reviewing inexperienced authors' work and giving ideas to improve them. Thus, coaching was coupled with reviewing and editing in some cases.

Most interviewees [1, 4, 5, 7, 8, 9, 10] suggested that coaching is indeed an effective policy for improving the average author talent and consequently the overall quality of the materials collection in the long run. Several interviewees [4, 5, 7] argued that coaching is the only policy that would sustain the quality level of the collection in the long run, since it gradually shifts the burden of maintaining quality within the community from a few experts to a larger number of experienced contributors.

The potential adverse effect of coaching on the experienced authors' own production was supported by most of the interviewees. Several interviewees [1, 4, 7, 9] emphasized that coaching was indeed a time-consuming process and that it took a lot from the experienced authors own production time. One interviewee [7] suggested that the loss of production is compensated by the increasing level of overall quality. Another interviewee [10] suggested that the production loss on the experienced authors' part would be compensated by the increase in the production by the inexperienced authors who were coached.

6.3.6. Comparing the Policy Options

At the end of the discussions about the policy options, the interviewees were asked to compare the policy options based on their merits and potential adverse effects, and suggest which policy options would bring the highest improvement with the least amount of loss when applied to their community. The key comments by the interviewees are given in Table 6.11.

Table 6.11. Key Comments from Interviewees Comparing the Four Policy Options

Respondent	Key Comments
Interviewee 1	Coaching takes a lot of time, but pays off. Filtering, reviewing and editing, and coaching all have their merits.
Interviewee 2	Coaching would be beneficial. Reviewing and editing and filtering combined would also be useful. Coaching, combined with reviewing and editing, has critical positive implications for the future. Coaching can take the form of collaboration or mentoring depending on the situation.
Interviewee 3	Selecting seems to be the most promising policy. Not everybody would become an author. So pick the right ones and encourage those. That can even be done overtly. It would require time, money and structure.
Interviewee 4	"I like [the selecting policy], but it would be very dangerous." Other than that, coaching is the most constructive, most positive. It develops more collaboration.
Interviewee 5	CLE is trying to do the reviewing and editing with coaching, "A little bit of filtering done in a very polite way, combined with reviewing and editing, which becomes coaching." If you do reviewing and editing in a coaching fashion it becomes most effective.
Interviewee 6	Reviewing and editing has coaching intrinsically. Coaching is the most efficient strategy. Plain reviewing and editing is an "after-the-fact" approach, and thus it is not so efficient. It involves some time and energy lost along the way. Selecting implies some kind of pre-judgment before anything has been done. "Coaching is more of a continuous quality improvement model."
Interviewee 7	A combination of local filtering ("narrowing-down"), and network-wide reviewing and editing works best.

Table 6.11. Key Comments from Interviewees Comparing the Four Policy Options (continued)

Interviewee 8	Coaching is the best policy. Especially, in the initial stages when gathering a critical mass of authors.
Interviewee 9	Filtering is beneficial. Reviewing and editing is not feasible. Selecting actually works in a small group. However, it may or may not be practical on a larger scale. Coaching has great potential, but the time issue is going to be critical. Release time for authors would help make coaching work.
Interviewee 10	Coaching.

Half of the interviewees [1, 4, 6, 8, 10] suggested that coaching was the most constructive and effective policy option. Two other interviewees [2, 5] suggested that coaching coupled with reviewing and editing would have considerable potential in improving product quality and developer talent while maintaining product functionality growth. Some interviewees [1, 2, 4] reiterated their arguments that coaching worked better in the long run. All of these suggestions were in parallel with the findings of the OSSD model policy runs about the higher coaching emphasis option. Higher coaching emphasis coupled with higher debugging emphasis provided an optimal mix of product quality and developer talent improvement while causing very small amounts of product functionality and community growth losses within the context of the OSSD model. (See Sections 5.5.5 through 5.5.7 and Section 5.5.9.)

One interviewee [3] argued that selecting would be the best policy option. Another interviewee [4] suggested that selecting would be favorable, but there was the danger of losing the interest of potential authors, which might decrease the number of authors, and consequently slow down the product functionality and community growth to the point where the community would fail to sustain itself. The policy runs with the

counterpart of the selecting policy in the OSSD model, higher barriers to entry, provided substantial product quality improvement without causing a dramatic loss in product functionality growth. However, one argument about the findings of those runs was that a very high barrier to entry coupled with developer participation or developer productivity levels lower than those in the base case could cause a large enough product functionality loss that the community would fail to sustain itself. Thus, the selecting policy was found to be a beneficial, but also a dangerous policy option within the context of the OSSD model. Accordingly, it can be argued that that interviewee's mental model about this policy option was in parallel with the findings of the policy runs.

One interviewee [9] suggested that the only feasible policy option was filtering. This interviewee argued that reviewing and editing and coaching were not feasible since they consume too much of experienced authors' time, and selecting would only work in a small group. These arguments were in disagreement with the findings of the policy runs on the OSSD model. While barriers to contribution, the counter part of the filtering policy in the OSSD model, did improve product quality, the improvement was just a little better than the other policy options, and it came in expense of product functionality and community growth.

One interviewee [7] suggested that the combination of the filtering and the reviewing and editing options would be the best policy to improve quality while maintaining product functionality growth. Another interviewee [2] suggested this combination as a feasible and beneficial policy, although not the best. A policy run that combined the counterparts of these two policy options, namely barriers to contribution and higher debugging emphasis, had not been performed on the OSSD model in the

original policy run set. However, such a run was performed in order to analyze the implications of the arguments of these interviewees within the context of the OSSD model. During the policy analysis of the OSSD model, the combination of filtering with reviewing and editing was found to be less favorable than a pure reviewing and editing policy in overall. On the other hand, the combination policy performed better than a pure filtering policy.

The analysis of the responses to the policy comparison question revealed that many of the interviewees' observations and mental model were in parallel with the findings of the policy analyses performed on the OSSD model. This finding provided more support for the argument that open source software development communities and instructional development communities such as the system dynamics K through 12 community can be studied as the instances of the same kind of online communities, which this study has defined as open online collaboration communities.

6.4. Implications for the General Dynamic Feedback Framework

The analysis of the interviews provided several important implications that guided the development of the dynamic framework. These implications are summarized in Table 6.11.

Table 6.11. Key Implications Indicated by the Interviews for the General Dynamic Feedback Framework

Loop	Summary Findings and Implications
Reinforcing Loop 3 ("More Functionality Attracts More Authors")	Keep the loop. However, change the detail structure: Authors come from Users, not from outside the community (represented by a cloud in the initial model.)
Reinforcing Loop 2 ("More Functionality Attracts More New Users, and That Attracts More New Developers")	Combine with Reinforcing Loop 3. The causal link from the number of users to the number of authors is a "material" link, instead of an "information" link.
Balancing Loop 1 ("Fewer Opportunities for Contribution Bring Fewer Authors")	Mark as suspect. May not hold for some communities, which focus on divergent products. May come into play very late in the life cycle of a community, thus having little effect on the success or failure of the community as a whole.
Balancing Loop 4 ("More Errors Bring Fewer Authors")	Keep the loop. However it works through Users rather than through Authors, agreeing with the changes to Reinforcing Loop 3.
Balancing Loop 5 ("More Errors Bring Fewer Users, and Fewer Authors")	Combine with Balancing Loop 4. Once again, the causal link from Users to Authors is not an "information" link, but a "material" link.

Using these implications as guidelines, the basic feedback structure of the OSSD model was transformed into a simplified, general dynamic feedback framework. In that sense, the final framework is a representation of the fundamental dynamic feedback structure of open online collaboration communities based on the initial OSSD model and the mental models of the interviewees about system dynamics K through 12 community. The details of the dynamic feedback framework are discussed in Chapter 7.

CHAPTER 7 -- A GENERAL DYNAMIC FEEDBACK FRAMEWORK FOR OPEN ONLINE COLLABORATION COMMUNITIES

7.1. The Framework

A general dynamic feedback framework was built based on the initial open source software development (OSSD) model and the findings of the ten interviews with the members of the system dynamics K through 12 instructional material development community. The framework is a simplified theoretical representation of the fundamental dynamic feedback structure underlying open online collaboration communities. The framework is represented as a causal loop/stock-and-flow diagram.

The dynamic feedback framework contains four main feedback loops. These loops were among the main reinforcing and limiting structures in the OSSD model. Furthermore, the interviewees identified these loops as structures that determine the performance of their community. These four loops are introduced below.

The members of the community are grouped into three: Users, Inexperienced Authors, and Experienced Authors. When users decide to make contributions to the collection they become inexperienced authors. The OSSD model did not involve a flow between the user pool and the author (developer) pool. In other words, there was no material link from the user pool to the author (developer) pool. The user pool influenced the author (developer) pool thorough an information link: as more users adopted the product, more developers were attracted to the community. However, most of the interviewees strongly argued in favor of a material link between the user and author pools. They had a twofold rationale for that: 1) Potential authors did not have adequate means of knowing the size and the structure of the user pool, and 2) New authors did not

come from outside of the community, they came from among the existing users. Accordingly, in the framework, new authors come from the user pool. Inexperienced authors become experienced authors as they mature in authoring. There are members that leave at every stage of the maturing process. (See Figure 7.1.)

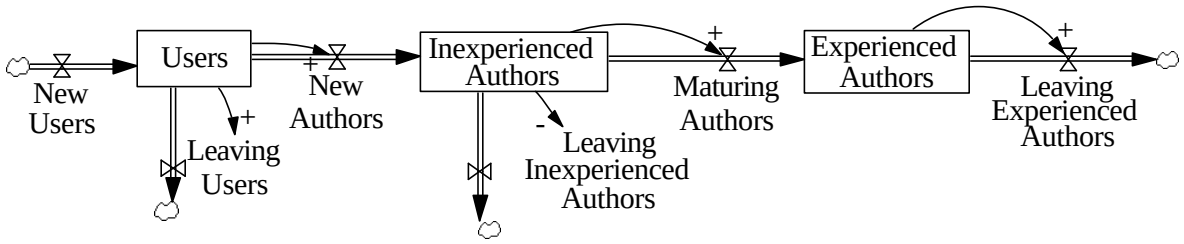


Figure 7.1. The General Dynamic Feedback Framework.

Inexperienced and experienced authors contribute to the production and build the product or the materials collection. The size and the functionality of the collection determine the number of new users. A larger and more functional (more useful) collection brings more new users, thus forming the main reinforcing loop of the framework. (See Figure 7.2.)

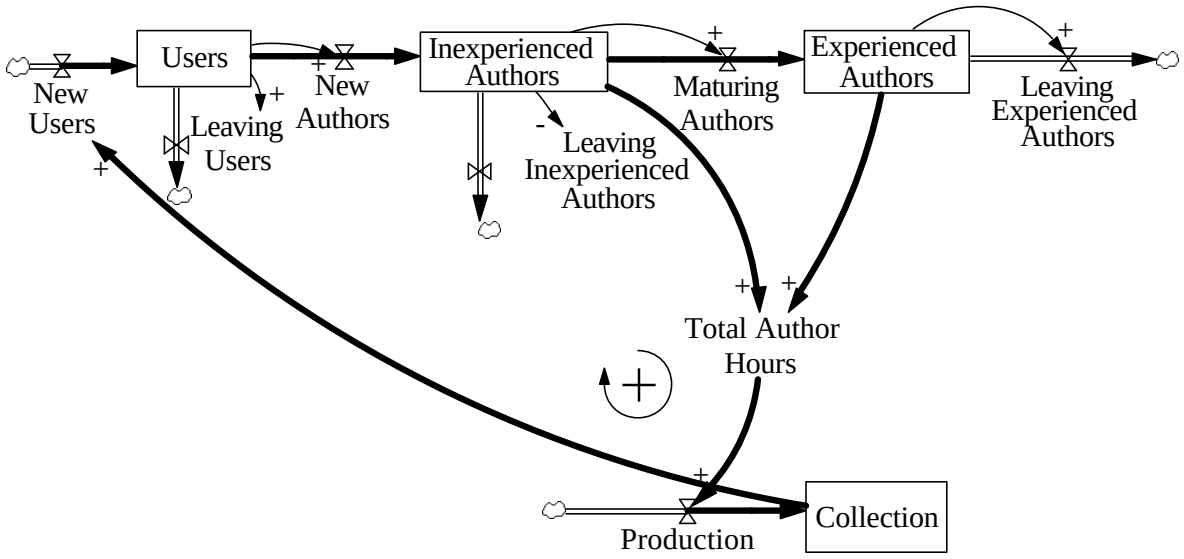


Figure 7.2. The General Dynamic Feedback Framework.

As authors produce content, they add quality problems to the collection. (See Figure 7.3.) Experienced authors review the collection, discarding materials that are of very low quality, and choose some other materials for rework. Inexperienced and experienced authors revise and improve the materials chosen for rework. (See Figure 7.4.)

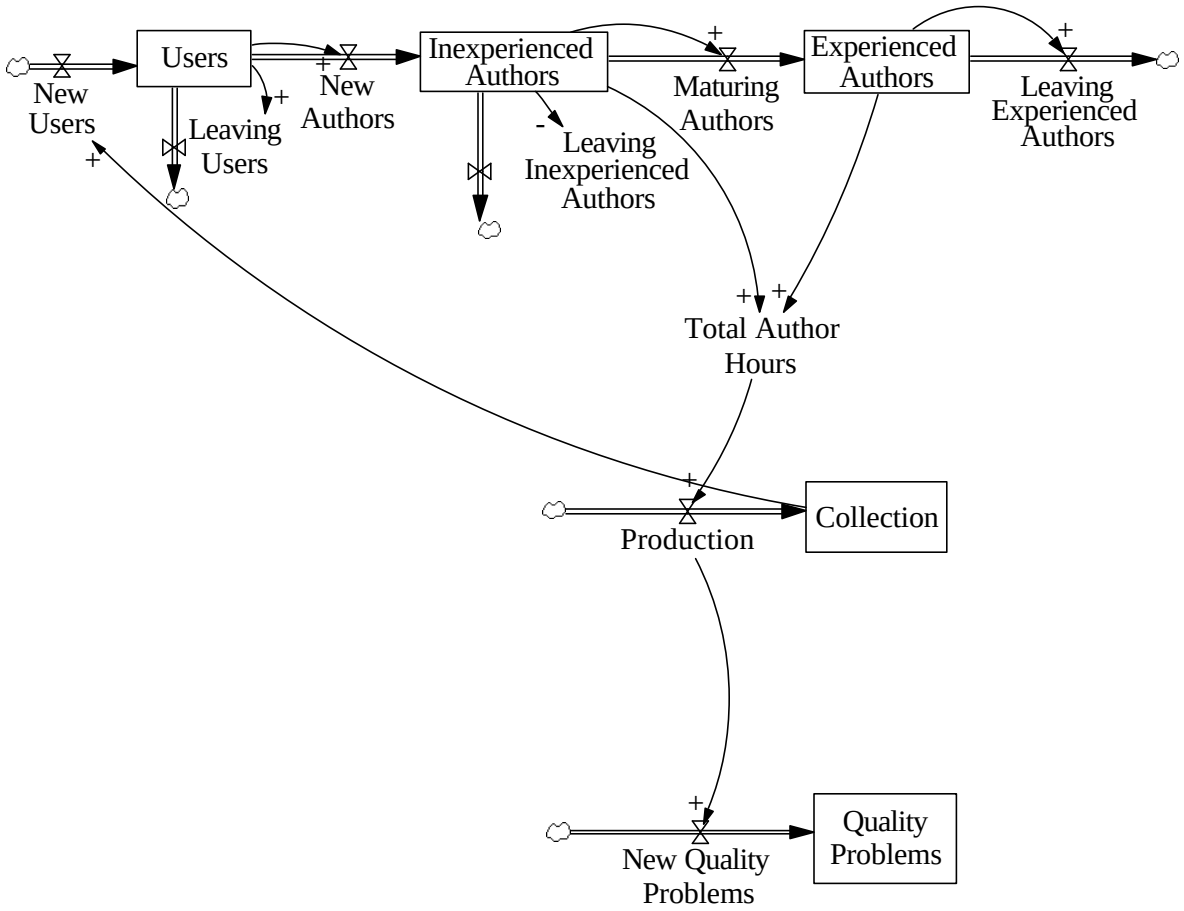


Figure 7.3. The General Dynamic Feedback Framework.

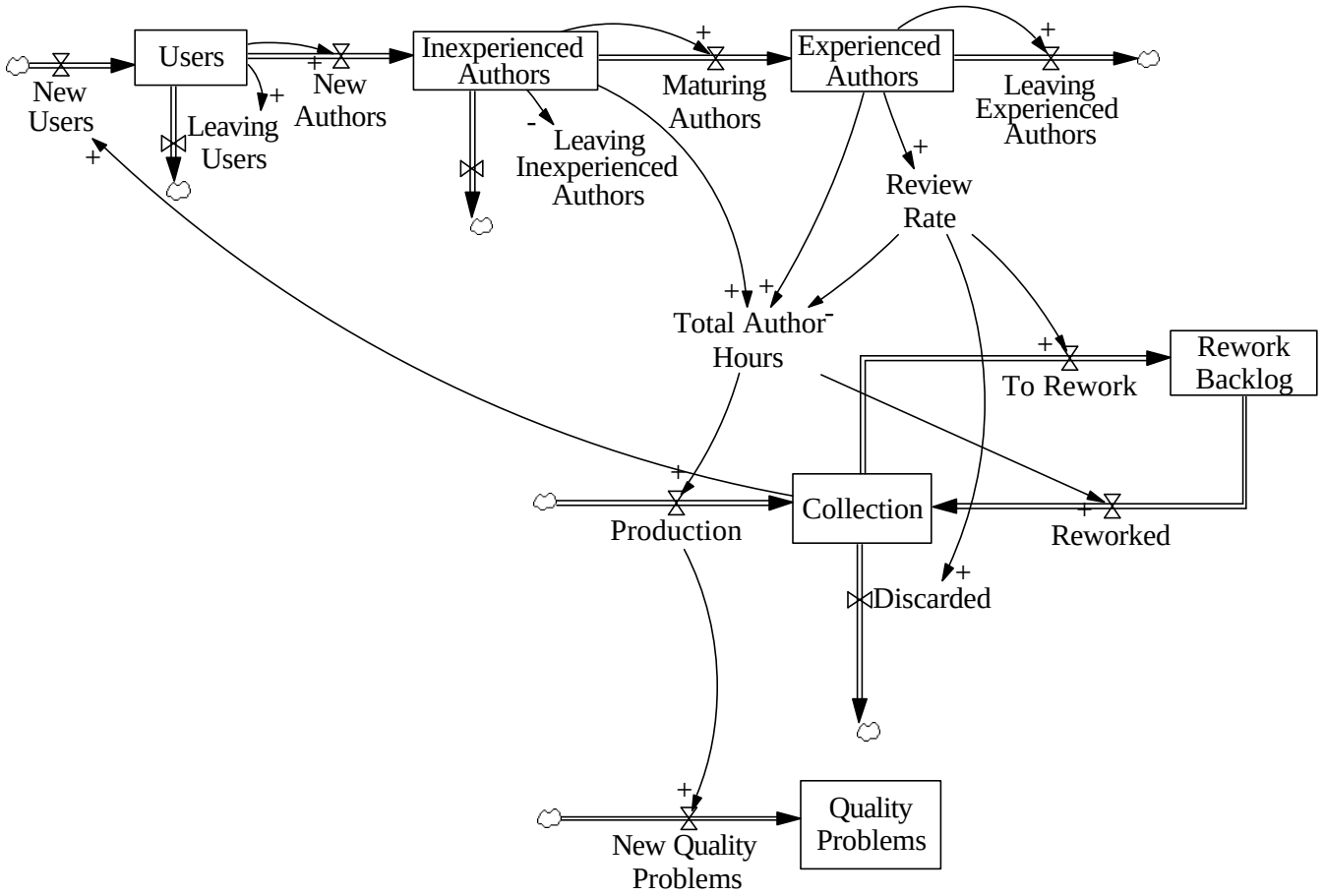


Figure 7.4. The General Dynamic Feedback Framework.

Discarding and reworking materials eliminate certain portions of the quality problems. The amounts of discarded and reworked materials are determined by the quality threshold, which is used by the experienced authors as the benchmark for evaluating the collection. This threshold also affects the ratio of users who become authors. A higher quality threshold means more discarded and reworked material, thus yielding a higher quality level. However, it also means a lower number of new authors. (See Figure 7.5.) The rationale is that a higher probability of their work being discarded or sent for rework will decrease users' motivation to make contributions and become authors. This follows from the findings of the policy runs based on the higher barriers to contribution option, which suggested that higher rejection ratios yield lower number of new authors (developers). (See Section 5.5.2 for the discussion about the higher barriers to contribution policy runs.) Also, the discussions with the interviewees supported the hypothesis that a considerably high rejection ratio would decrease the motivation to participate in production. (See Section 6.3.2 for the discussion about interviewees' comments on the effects of high rejection ratios.)

Quality threshold also determines the barriers to entry. As the quality threshold increases the community becomes more selective in accepting new authors, and the number of users accepted into the inexperienced author pool decreases. (See Figure 7.5.)

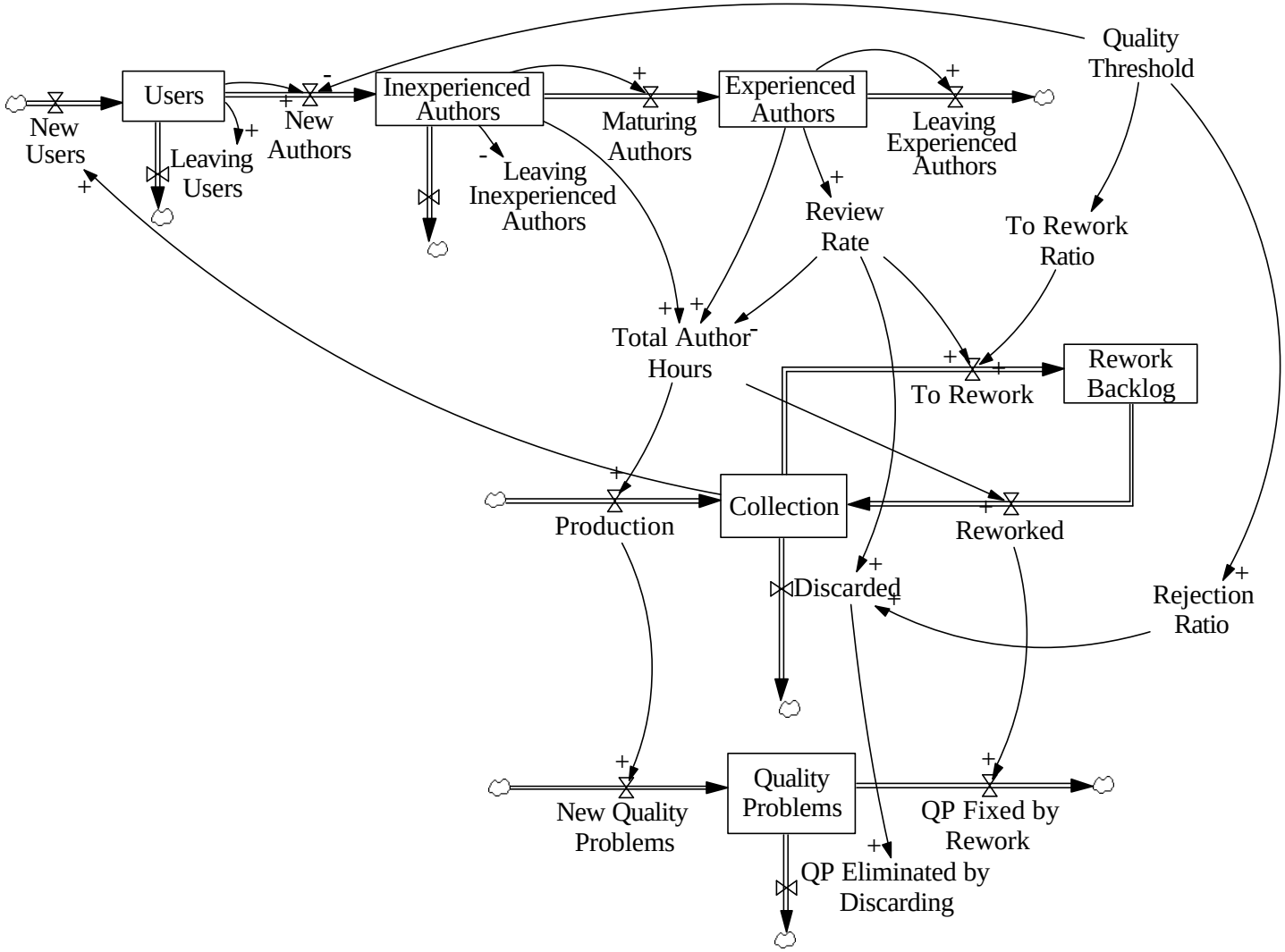


Figure 7.5. The General Dynamic Feedback Framework.

The density of quality problems, which is defined as the number of quality problems per unit of the collection, determines the rates with which new users join the community, and exiting users leave. A higher density of quality problems would yield a lower number of new users, and a higher number of leaving users. These links form the second main loop in the framework, which is a balancing (limiting) one. (See Figure 7.6.)

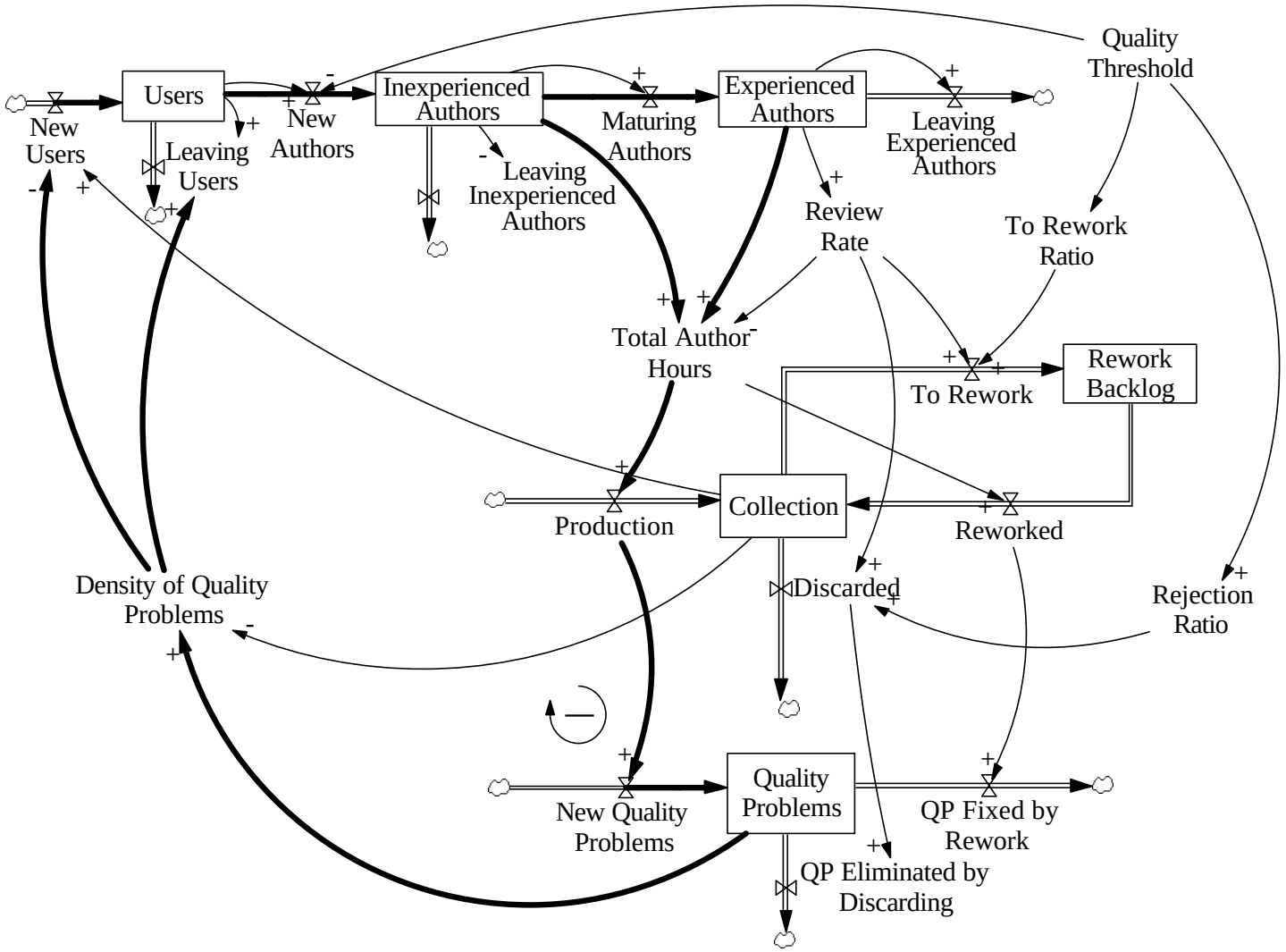


Figure 7.6. The General Dynamic Feedback Framework.

The third main loop of the framework, which also is a balancing one, is formed by the opportunities for contribution. As the collection gets larger and more functional, opportunities for contribution decrease. Decreasing opportunities for contribution decrease the number of new authors, since potential authors may be discouraged by the lack of vast opportunities for making contributions. (See Figure 7.7.) This was one of the main limiting loops in the OSSD model. However, the discussions with the interviewees about the existence and effects of such a limiting loop suggested that this loop might come into effect considerably late in the process for some open online collaboration communities. In fact, it may not come into effect for some communities that focus on divergent tasks such as instructional materials collections, rather than convergent tasks such as software products. Although many interviewees suggested that this loop was plausible theoretically, they emphasized that they have seen no indication that this loop exists or can be effective in their community. Thus the link from the collection to the opportunities for contribution and the link from the opportunities to new authors are marked as “questionable” in the final framework, and shown in dashed lines. (See Figure 7.9.)

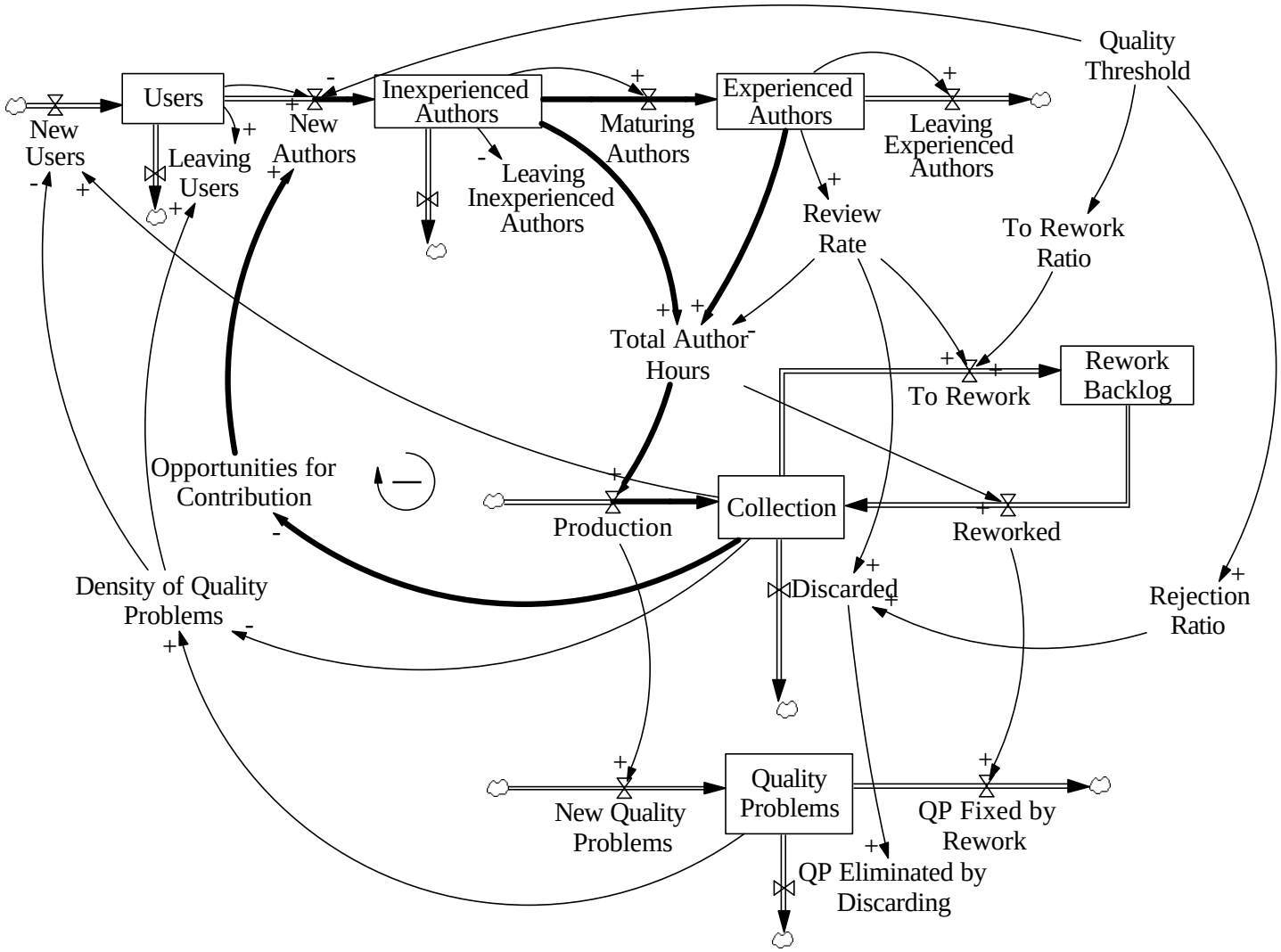


Figure 7.7. The General Dynamic Feedback Framework.

Average developer talent and coaching form the fourth loop, which is a reinforcing one. Coaching increases average developer talent, and as average developer talent increases quality problems decrease, causing a lower density of quality problems. A lower density of quality problems brings more new users, and slows the leaving of the existing users, thus increasing the number of users more quickly. More users mean a higher number of new authors, which increases the number of authors more quickly, and thus provides more author hours available. More author hours close this reinforcing loop by giving way to more coaching. (See Figure 7.8.) This loop shows its reinforcing effect in the long run, since average talent takes a lot of time to build. This was an important point that the interviewees argued when discussing the effects of coaching. Most of the interviewees suggested that coaching is the most effective policy option in the long run. (See Section 6.3.5 and Section 6.3.6 for discussions with interviewees about the coaching policy and the comparison of the policy options.) Figure 7.9 shows the final framework in its entirety with all the loops and variables involved.

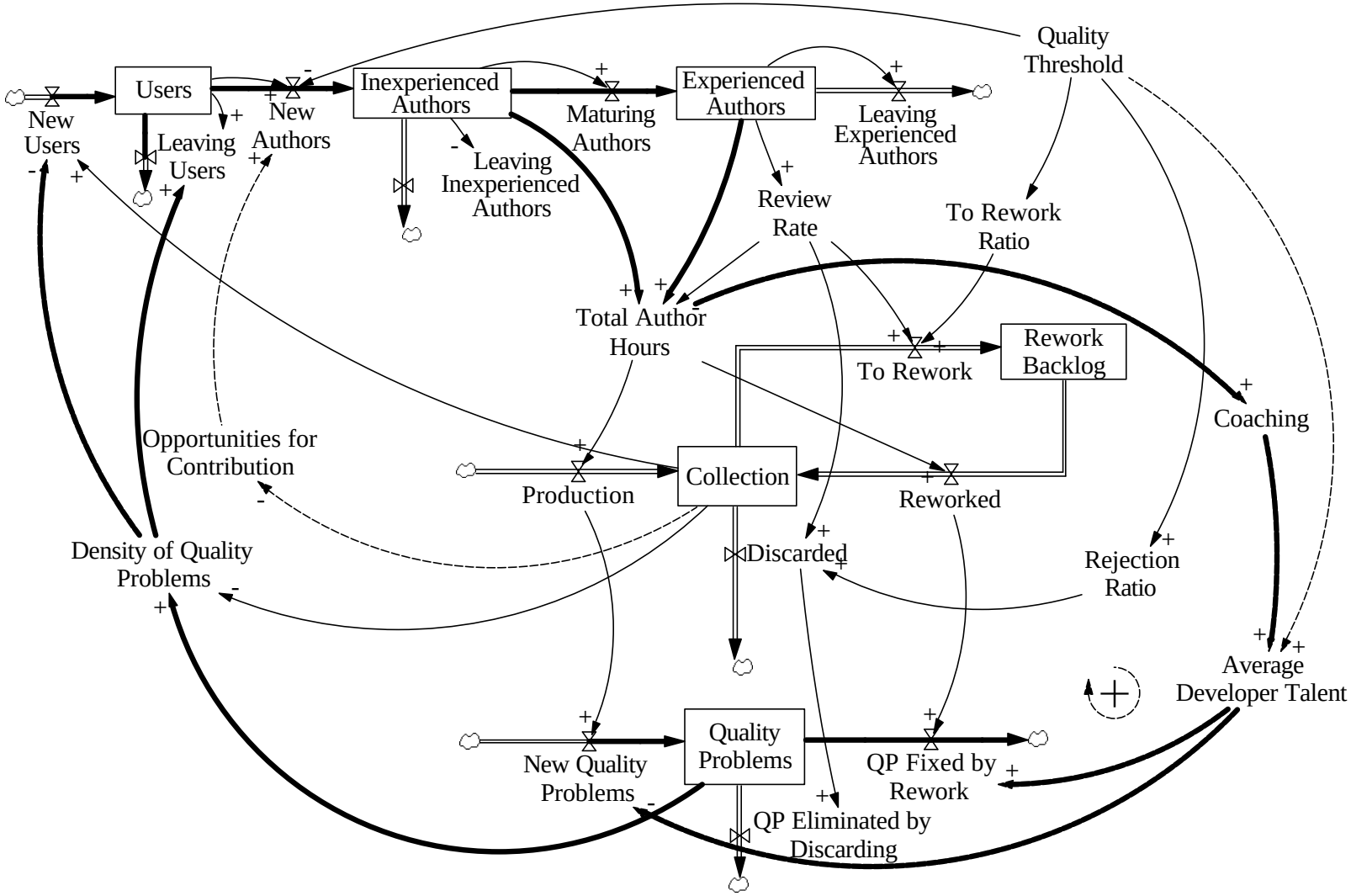


Figure 7.8. The General Dynamic Feedback Framework.

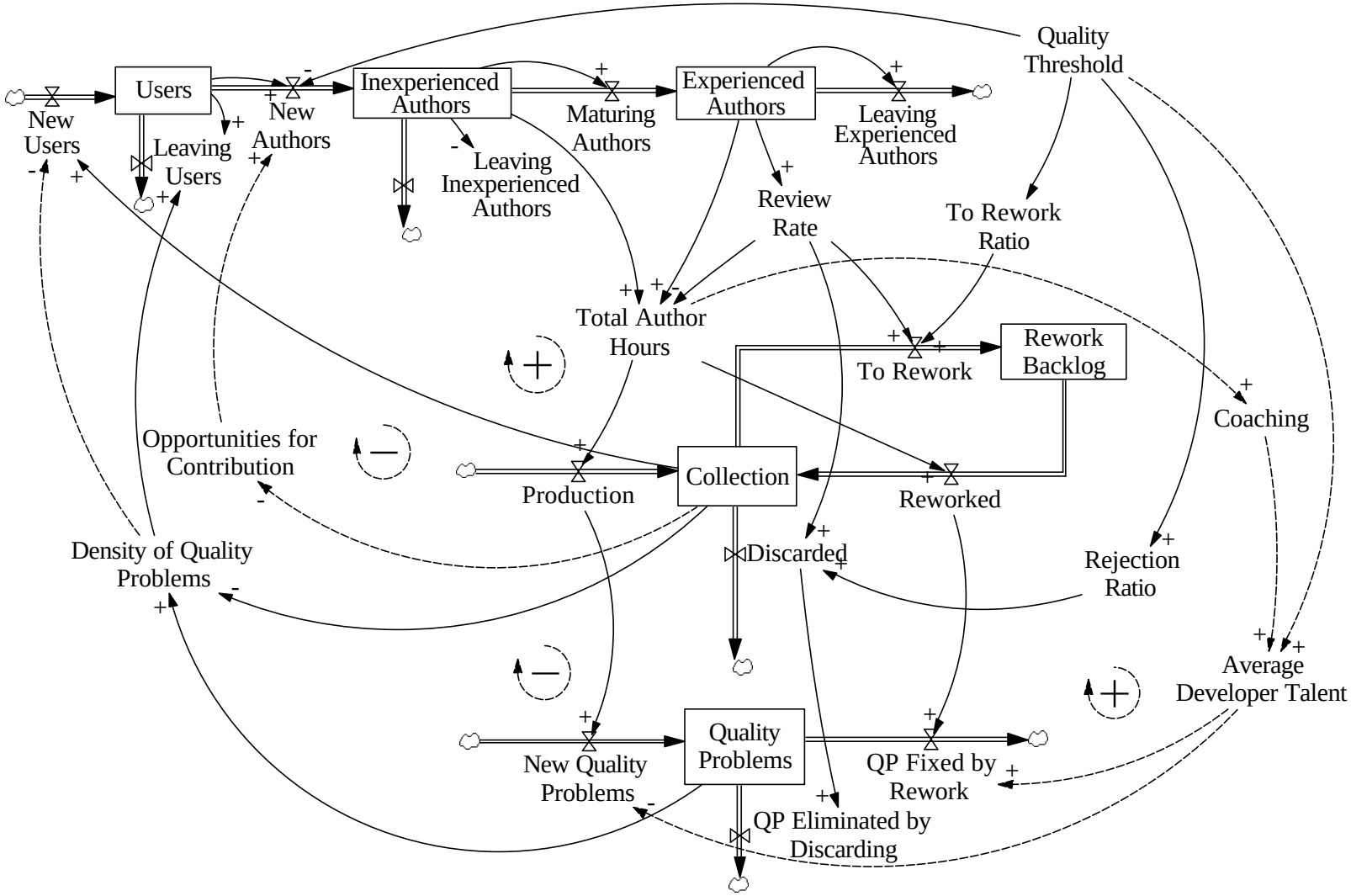


Figure 7.9. The General Dynamic Feedback Framework.

The framework is a concise representation of the dynamic feedback structure that underlies open online collaboration communities. It has the potential of explaining the phenomena that determine the growth or decline of an open online collaboration community. The feedback framework can be used as a basis for developing a generalized dynamic feedback simulation model of an open online collaboration community. The causal relationships between the variables of the framework or the feedback loops can be used as hypotheses for empirical research studies. The framework can be further refined based on the findings of such research studies.

7.2. Strengths and Limitations of the Study

7.2.1. Strengths of the Study

This study used a multi-method research approach, which combined system dynamics and qualitative analysis of structured interview data. A multi-method approach provides a way to study complex social phenomena by integrating different methodologies. That way the researcher has the opportunity to combine the strengths and compensate for the limitations of each individual methodology (Brewer and Hunter 1989 pp.17). The system dynamics modeling phase provided a means to develop and articulate a preliminary hypothesis in the form of a system dynamics model. The interview and qualitative analysis phases provided a way of testing the hypothetical model against the observations and mental models of the members of a specific open online collaboration community.

System dynamics is a quantitative modeling approach, which is particularly fit for modeling complex, large scale, non-linear, partially qualitative social systems. System dynamics provides a means to conceptualize and model systems of mutual causal

relationships and feedback structures among high numbers of variables, which is harder to achieve with other quantitative modeling methods.

The hypothetical system dynamics model was based on a literature review of three relevant literature streams. The literature review was particularly focused on the parallels that exist between these literature bodies, in an effort to ground the model on multiple theoretical foundations.

Qualitative analysis of structured interviews is an efficient method for building theories and testing hypotheses based on rich, qualitative data, which provides a means to look at complex social phenomena within a deeper context than that provided by most quantitative approaches. The depth provided by a qualitative approach is hard, if not impossible, to achieve using solely quantitative methods.

The research design provided a means to test the hypothetical system dynamics model of open online collaboration communities against the personal observations and mental models of the members of a representative community. The analysis of the interviews provided rich and deep conceptual basis for testing the model and articulating the dynamic framework. The final theoretical dynamic framework provided a basis for developing models that can be applied to a wider range of cases.

Another important strength of this study was the contributions it made on several fronts. The study contributed to various literatures, provided critical insights for managing OOCCs, and constituted a basis for numerous potential future research studies. For a full discussion about the contributions made by the study see Section 7.3.

7.2.2. Limitations of the Study

Qualitative research methods that are used to analyze rich qualitative data through interviews and fieldwork have limitations with respect to reliability and representativeness (Babbie 1998 pp.304). Findings of a qualitative research study would inevitably bear subjective judgments on the researcher's part. The researcher's duty is to consciously minimize this subjective "noise." If more than one data collector or analyzer is involved in the study, the differences in their subjective dispositions would bring about a problem with consistency of data collection and analysis.

Qualitative research studies based on interviews generally involve smaller sample sizes compared to those of questionnaire-based surveys, and experiments. Also the sampling schemes are not always random (not probability-based). These factors bring about a concern about the representativeness of qualitative research study findings. When drawing conclusions from the findings of a qualitative research study, the researchers should always keep in mind that external validities of qualitative research findings are very limited.

Consequently, the research design used for this study had limitations with respect to external validity. The findings of the case analysis have limited representative value. The applicability of the initial system dynamics model was tested against only one open online collaboration community. Thus, the final dynamic feedback framework, which is based on the hypothetical system dynamics model and the findings of the interviews, requires further empirical testing before it can be generalized to other cases.

System dynamics models are limited and simplified representations of the real world. Classic system dynamics texts acknowledge this fact and argue that models cannot

be "verified," or "validated" in the exact sense (Richardson and Pugh 1981, Sterman 2000). Thus, system dynamics methodology provides tests for "confidence building" in models, rather than "validating" models. As discussed previously in this document, system dynamics methodology is mostly used for modeling partially qualitative systems. All variables in a model should be quantified in order to be able to simulate the model, including the "soft," qualitative variables, and that may mean that some (or in many cases most) variables will be parameterized based on limited or no "hard" data. However, this generally does not hurt the level of confidence toward robust models, since the overall behavior of a robust model does not change significantly based on parameter changes. On the other hand, system dynamics models are not good point estimators, partly due to the parameterization issue, and partly due to the fact that they involve complex feedback structures, and a significantly higher number of variables compared to predictive models such as time series models or econometric models.

System dynamics models, just like any conceptual model, contain biases that their modelers bring in about the systems or problems being modeled. The modelers can try to identify and stay aware of their biases, but ultimately, any model would contain a number of biases. The effort of a modeler to keep his or her model bias-free is beneficial not because it can actually achieve that goal, but because it can reduce the number and extent of biases in the model. The OSSD model is not an exception to the rule. While the model is an integration of existing literatures on the theoretical approaches to online communities, open source software development, and applications of system dynamics to software project management, it undoubtedly hosts the biases and presumption of its modeler about these concepts. However, the OSSD model is as good as any other similar

model can be, since those similar models would be the products of modelers who have their own biases and presumptions. The important point here is the acknowledgement of this limitation about the model.

The research design lacks a component to test the initial system dynamics model against empirical data from actual open source software development communities. This limits the confidence with respect to the internal validity of the model. A possible solution is testing the model through interviews with members of an actual open source software development community, as discussed in the future research opportunities section below.

When the initial model was tested for its applicability to the specific instructional material development community through the interviews, the ideal case would be asking the interviewees about the applicability of all the loops, variable definitions and causal relationships in the model to their community. However, the limited time and attention span of the interviewees during the interviews did not allow for an analysis of that magnitude. Consequently, a substantial portion of the initial model could not be tested. The loops and causal relationships that were selected for presentation to the interviewees were those that were identified as the main drivers of model behavior during the model building and model analysis phases. This approach holds a potential bias factor on the part of the modeler/researcher. Some of the feedback loops, variables and causal relationships that were omitted from the interviews based on the researcher's judgment about their importance could be identified by the interviewees as important structures that affect the behavior of their community. However, the interviewees did not have the chance to see and comment on these loops, variables, and relationships. This limitation is

not particular to this study. Any model-based social research study based on testing a large model against individuals' observations and mental models should involve some degree of simplification of the model before it can be tested. The OSSD model has over 270 variables, several hundred causal relationships, and over 500 major and minor feedback loops. A model this big cannot be tested *in its entirety* against individuals' observations and mental models. That would require not only an enormous amount of time, but also a tremendous cognitive effort and concentration on the part of the subjects in order to understand all the structural details of the model. Not many subjects would be willing and able to do that. Furthermore, trying to mentally digest a very large model structure may overwhelm the cognitive ability of the subject. In such a case, the subject may choose to accept or reject the model in its entirety without further comparison to his or her observations and mental models. The subject may also choose to neglect certain portions of the model in order to be able to digest at least a part of it, and thus compare only the part that he or she chooses to focus on against his or her mental models. Any of these strategies by the subject would defy the purpose of testing the model in its entirety. Furthermore, it would not always be possible to detect whether the subject is using such a strategy. Therefore, we can argue that any large model would need some degree of simplification before it can be tested against individuals' mental models.

Testing simplified models pose cognitive problems too. Even small and simple models presented to subjects have the potential of distorting subjects' own mental models and inducing biases in subjects' selection of personal observations to support or refute the structures presented to them. In fact, we can argue that it is impossible to keep the mental models of the subjects intact while presenting them external models. To further

complicate the problem, it is not practically possible to assess the degree to which the mental model of a subject is affected by the introduction of the external model.

Another problem that is inherent to testing models with subjects is the effect of psychological processes between the researcher and the subjects, and internal to the subjects. The personalities of the subject and the researcher, the relationship between the researcher and the subject, methodological factors such as the way the questions are designed, the order in which the questions were asked, and external factors such as interview media (phone, face-to-face, etc.) may all have substantial effects on the responses of the subject. The subject may approach the questions in a conformist manner and support even those structures that do not fit to his or her mental model or personal observations. On the other hand, the subject may tend to reject more than what does not fit to his or her mental models in an effort to avoid looking (or feeling) too conformist. Once again, it is not possible to catch all of the instances when such a process comes in effect, and measure to what extend it affects the subject's arguments about the model being tested.

These problems limit the refuting power of interviews. Refutation is a crucial feature of hypothesis testing. A research design or instrument that does not allow refuting is not fit for hypothesis testing. It is an important task on the part of the researcher to improve the refuting power of the design he or she is using. All these arguments may cast doubt about the validity of model-based social research. However, until we find better methods and instruments to assess individuals' mental models and personal observations, model-based social research should be used as a viable approach with known and acknowledged limitations.

In the case of this study, the interviewees supported most of the loops and policy options presented to them during the interviews. A naïve explanation for this would be that the OSSD model captured the reality so accurately that there was not a lot in it to be refuted. However, a better approach would be to question the refuting power of the interviews and the data analysis phase. The refuting power of the interviews and the data analysis in the case of this study might have been limited by the factors inherent to model-based social research as discussed above. The particular design of the study might have limited the refuting power, as well. On the other hand, the interviews identified two important differences between the generalized OSSD model and the interviewees' mental models. First, the interviewees rejected the casual link from the number of users to the rate of new authors. The interviewees did not reject the overall loop and suggested an alternate structure where the new authors come from the existing users rather than from out of the community. Also, the interviewees argued that the balancing loop that is driven by decreasing opportunities for contribution is not observed in their community, although most of them suggested that it might be a plausible loop in theory. These examples show that the research design used in this study had a certain refuting power. In fact, arguably the most valuable findings of this study were these two challenges to the model. We can argue that if a research study does not refute anything, nothing new has been learned from it, since whatever the study supported was already known. In that sense, we would not learn anything from the interviews if all the structure presented to the interviewees went unchallenged.

Another limitations of the study is that the final dynamic feedback framework does not provide a quantitative means to articulate and test the emergent theory until it is

further developed into a system dynamics model. Policy options cannot be tested without simulation, and the final theoretical framework cannot be simulated until it is developed into a system dynamics model. This hampers the usefulness of the final theoretical framework for policy analysis purposes. Despite all its limitations, this study made several contributions on different fronts. The following section discusses these contributions.

7.3. Contributions of the Study

This study made notable theoretical and practical contributions to several fields. These contributions can be grouped under three headings: 1) contributions to several streams of literature 2) critical insights for managing OOCCs, and 3) topics for future research studies. These groups of contributions are discussed below with references to relevant sections of the dissertation.

7.3.1. Contribution to Related Literatures

Literature on Online Communities

This study made an important contribution to the online communities literature by defining open online collaboration communities as a special type of online communities. As discussed in the research purpose section, most of the existing studies approached online communities without distinguishing between different types. On the other hand, there are some studies that focused on special types of online communities, such as studies on open source software development communities. However, these studies kept their focused too tight, and did not attempt to encompass as a wide a group of communities as open online collaboration communities. This study defined open online

collaboration communities as online communities that are formed by loosely connected groups of people, who use the Internet as a medium for carrying out collaborative projects for producing and improving a wide range of information products. (See Section 2.2 for a discussion about the definition and characteristics of open online collaboration communities.) This definition can be used as a starting point for studying phenomena related to OOCCs, such as motivation factors for people to participate in these communities, ways to manage motivations and expectations in order to accelerate and sustain community growth, and strategies to improve the products developed by the communities. Other questions related to OOCCs such as strategies to improve the talent levels of inexperienced participants, ways to determine and enforce quality standards within an OOCC, and methods to improve dissemination of products developed by the community can also be studied using the framework as a starting point.

This study also analyzed and integrated several theoretical approaches to the study of online communities. Gift economies, public goods, social informatics and social networks perspectives were analyzed, and the implications of the first three perspectives for open online collaboration communities were identified. An important one of those implications was that communities that provide more utility with their products would attract more participants. Another important implication was that as the user pool of a product developed by an OOCC becomes larger, the community would become more attractive to contributors. The argument that an easier and simpler contribution process would make an OOCC more attractive to contributors was another important implication that emerged from the review and integration of the theoretical approaches. These

implications were integrated within the context of the open source software development (OSSD) model.

Parallels between the online communities and open source software development literatures with respect to the theoretical approaches were also identified. Both literature streams used all of the four theoretical perspectives, which were discussed in the literature review section, to explain phenomena related to communities they studied. Furthermore, the two literature streams derived similar implications from those theoretical perspectives about the communities they studied. (For a detailed discussion about those implications see Section 2.3.)

The findings of the literature review provided a comprehensive theoretical basis for future studies that may approach phenomena related to open online collaboration communities such as different ways of organizing these communities and how the way of organizing affects the performance of communities, how leadership is structured and how power is distributed within such communities, and how collaboration among participants is realized. Those potential studies can employ different methodological perspectives such as survey based statistical analysis to test hypotheses, ethnographic or grounded-theory based qualitative approaches to identify deeper concepts and causal relationships that are hard to discover without analyzing rich qualitative data, and simulation based methods to test scenarios related to phenomena that pertain to OOCCs.

Furthermore, this is the first research study which applied system dynamics modeling and simulation method to studying online communities. As discussed in the methodology chapter, system dynamics is a quantitative modeling approach, which is particularly fit for modeling complex, large scale, non-linear, partially qualitative social

systems such as open online collaboration communities. System dynamics provided an adequate means to conceptualize and model the system of mutual causal relationships and feedback structures that exist in open source software development communities. That is a task that would be considerably harder to achieve with most other quantitative modeling methods. (See Section 3.2 for a discussion about the system dynamics methods.)

Literature on Open Source Software Development

The study made contributions to open source software literature, as well. First of all, it placed open source software development communities within the concept of open online collaboration communities. While the validity of this classification is open to discussion, it nevertheless provides a framework for integrating studies on open source software development with studies on other types of open online collaboration communities, such as courseware development communities, collaborative authoring communities or collaborative music making communities.

The system dynamics model provided valuable insights about the structure and potential behaviors of a hypothetical open source software development community. An important insight was that any policy aimed at improving the quality of an open source software product would slow product functionality growth. In fact, extremely high levels of such policies may lead to severe impediments of functionality growth, which may stagnate community growth and even cause the community to fail to sustain itself and disintegrate. The importance of the patience factor in making an open source software community succeed or fail was another valuable insight. The findings of the analyses on the patience factor implied that managing expectations about product functionality

growth within the community is a crucial task that the leaders of the community should undertake in order to make the community successful. (See Section 5.4.9 and Section 5.5.9 for a full discussion about the implications of the potential behaviors of the OSSD model under different external conditions and policy settings.) The model also served as a hypothetical representation of open online collaboration communities, which was tested through the interviews with the members of the system dynamics K through 12 community. (See Section 6.4 and Section 7.1 for discussions about the implications of the interviews for the OSSD model, and the final framework based on the model and the findings of the interviews.)

The OSSD model is the first system dynamics model of open source software development, which integrated concepts such as code production, debugging, coaching and developer motivation. The model provided critical insights about the relationships between performance measures such as product functionality, product quality, community growth, and determinants of success such as participation, productivity and developer talent. Sensitivity and policy tests performed on the model provided important implications about the potential effects of policy interventions on performance measures. One such implication was that variables such as average developer talent and average developer productivity have critical values below which an open source software development community fails to sustain itself in terms of product functionality and community growth. Critically low values for such variables keep product functionality growth so slow that achieved functionality growth cannot reach the level of expected functionality. The fundamental cause behind such a failure was found to be the patience factor -- as patience of users and developers runs out their expectations about product

functionality grows. Runs showed that, under an infinite patience assumption, failures caused by low achieved functionality would not happen. Policy options such as barriers to entry and barriers to contribution also had critical levels above which the community could not sustain product functionality and community growth. Diminishing patience assumption was the fundamental reason behind this as well. Under infinite patience even extremely high levels of such policies did not cause a failure driven by slow functionality growth. Higher barriers to entry, a combination of higher debugging and coaching emphases, and and overall combination of these two policies were found to be the most beneficial policy option in overall. (For a full discussion on the comparison of various policy options see Section 5.4.9.)

Literature on Applications of System Dynamics

System dynamics is another stream of literature to which this study contributed. As emphasized above, the OSSD model is the first comprehensive system dynamics model of open source software development. Although many software project management models have existed in the system dynamics literature, the OSSD model is the first that focused specifically on an open source software development project. (See Section 2.4 and Section 2.5 for a review of applications of system dynamics to software project management and instructional material development.) The OSSD model opened a new topic area for applying system dynamics modeling, since it was conceptualized also as a representation of open online collaboration communities. In that sense, the OSSD model serves as a starting point for future system dynamics studies applied to open online collaboration communities in particular, and online communities in general.

This study also provided an example of combining qualitative and quantitative research methods by using a multi-method research approach, which combined system dynamics and qualitative analysis of structured interview data. That way the researcher had the opportunity to combine the strengths and compensate for the limitations of each individual methodology. The system dynamics modeling phase provided a means to develop and articulate a preliminary hypothesis in the form of a system dynamics model. The interview and qualitative analysis phases provided a way of testing the hypothetical model against the observations and mental models of the members of a specific open online collaboration community.

7.3.2. Implications for Practice

Defining OOCCs and the Underlying Policy Problem in OOCCs

The study provided some critical implications for practice, as well. One important contribution of the study in this regard was defining open online collaboration communities as a special type of online communities, which involve a common underlying policy problem. The analysis of the policy runs performed on the OSSD model revealed that a fundamental trade-off exists between building functionality and improving quality of products developed in open online collaboration communities. (See Section 5.5.9 for a discussion about the underlying policy problem.) The interviews with the members of the system dynamics K through 12 community supported the argument that the tension between building functionality and improving quality is a common policy problem in open online collaboration communities. Almost all the interviewees, with the exception of one, suggested that they see that tension as the underlying policy problem in their community. (See Section 6.3.1 for the interviewees' comments about the underlying

policy problem.) These findings imply that the leaders of open online collaboration communities should be prepared to make decisions about how to best manage this tension, and determine what emphases they will put on building functionality and improving quality within their communities. This implication should be particularly relevant for leaders of open source software development communities and instructional material development communities since these communities were defined as open online collaboration communities in this study.

Defining the Structure that Causes the Underlying Policy Problem

The study also identified the structure that caused the underlying policy problem by building the OSSD model, testing it with the interviews and finally developing the dynamic feedback framework based on the OSSD model and the implications of the interviews. The dynamic feedback framework can be used as a tool for understanding and communicating the structure that causes the success or failure of open online collaboration communities, as well as the fundamental tension between building functionality and maintaining quality while building an open online collaboration community.

Implications about Potential Solutions for the Policy Problem

Besides defining the fundamental policy problem, and the underlying structure that causes it, the study provided several critical implications about the potential solutions for the policy problem. One important implication was that each potential solution was limited in terms of the improvement it can provide before causing another problem within the overall system. For example, each of the policies tested for improving quality slowed down product functionality and community growth beyond a certain level. Some of these

policies even had critical levels, above which the community would fail to sustain itself due to losses in functionality and community growth. On the other hand, policy decisions aimed at accelerating functionality growth beyond a certain point would impede quality improvement. All these findings point to the implication that the leaders of open online collaboration communities can use a specific policy only to a certain extent while trying to improve quality or accelerate functionality growth. Pushing any given policy option beyond its optimal extent would not only fail to provide any additional performance improvement in the expected direction, but also hamper the overall performance in terms of other measures. (See Section 5.5.9 for a discussion about the policy implications of the model.)

The analysis of the model through policy runs provided implications about the effectiveness and the side effects of some specific policies. The two most favorable options were a pure barriers to entry policy, which involved a high level of refusal ratio for selecting new developers, and a combination of higher debugging and higher coaching emphases, which was based on higher pressures for bug discovery, bug fixing and talent building. These policies yielded substantial improvements in product quality at the expenses of very limited losses in product functionality and community growth. An overall combination of the two best policy options yielded an even higher quality improvement, but caused much bigger losses in product functionality and community growth. Consequently, no single best policy option emerged from the policy runs. However, depending on the structure and the culture of a given open online collaboration community, a pure higher barriers to entry policy, a combination of higher debugging and higher coaching emphases, or an overall combination of higher barriers to entry, and

higher debugging and coaching emphases policies would be effective policy alternatives for improving product quality while maintaining functionality and community growth. (See Section 5.5.9 for a detailed comparison of the policy runs.)

The analysis of the interviewees' responses to a question about comparing the policy options revealed that the interviewees' observations and mental models were mostly in parallel with the findings of the policy runs performed on the OSSD model. Most of the interviewees argued that coaching either as a pure policy or in combination with reviewing and editing would be the most effective policy, particularly in the long run. (The counterpart of the reviewing and editing option was the higher debugging emphasis policy in the context of the OSSD model.) Two interviewees suggested selecting new developers as a viable option; however, one of them emphasized that such a policy would pose the danger of alienating potential authors. These findings, coupled with the findings of the policy runs on the OSSD model, imply that coaching is the most effective policy for improving quality and developer talent while maintaining functionality and community growth, especially in the long run. Selecting, either as a pure policy option, or coupled with coaching, and reviewing and editing can also be a viable and effective policy option for certain open online collaboration communities, provided that their structures and the cultures allow such an approach.

7.3.3. Topics for Future Research Studies

Improving the Open Source Software Development Model

The study also provided a wide range of topics for future research studies. One group of potential topics involves improving the open source software development (OSSD) model. As discussed in Chapter 5, the OSSD model generally performed

satisfactorily under extreme condition and sensitivity runs. However, there were some runs, which indicated that the model could benefit from further refinement. For example, the extreme condition run which involved zero bug generating rate normal did not generate a behavior as extreme as expected. Also, the sensitivity runs with different values of normal time to attract developers, normal time for developers to leave, and normal time to attract users did not generate behavior as varied as expected. A study that focuses on the revision and refinement of the equations involving these variables would be very beneficial.

Testing the OSSD Model Against Empirical Data from Actual Communities

The research design used for this study did not include a component to test the initial system dynamics model against empirical data from an actual open source software development community. It is possible to design an interesting study that would test and improve the model through interviews or questionnaires with the members of actual open source software development communities. Another variation of this study would be testing the policy implications of the OSSD model with the members of an actual open source software development community.

The limited time and attention span during the interviews did not allow testing all the feedback loops, causal relationships and variable definitions of the OSSD model against the observations and the mental models of the interviewees. An extended version of the interviews could put more of the structure of the OSSD model to test. Such a study could be based on a delphi-type iterative approach, which would involve more than one interview with each subject. This could improve the OSSD model dramatically, as well as build a very high level of confidence in the final model.

Developing a General System Dynamics Model of OOCCs

Another potential research topic is developing a general system dynamics model for open online collaboration communities. The final dynamic feedback framework is an adequate starting point for such a model. The general system dynamics model would provide a quantitative means to articulate and test the theory that emerged through the dynamic feedback framework. Such a model could be used as a testing platform for policy analysis in the context of a wider range of open online collaboration communities. An empirical component can be added to such a study by including data collection from several open online collaboration communities, preferably with different characteristics and product types. That would increase the representativeness of the model and build a higher level of confidence in the model.

Testing the Implications of the Final Framework Against Data from OOCCs

Data collection from several open online collaboration communities through interviews and questionnaires could also used in a study that would further test the implications of the initial OSSD model or the final dynamic feedback framework. That would increase the representative value of the dynamic feedback framework substantially. Inclusion of a questionnaire-based component would be particularly beneficial, since such a component would provide data from a larger sample. The interviews used for this study involved 10 subjects and thus did not lend themselves to statistical analysis. A questionnaire-based data collection could provide enough sample size for statistical analysis. A larger number of interviews would also provide a sample size large enough for statistical analysis; however, that would obviously require more time and resources than a questionnaire-based data collection method.

Empirical Research on Hypotheses Derived from the Final Framewrok

Another group of studies could focus on empirically testing the causal relationships and feedback loops from the framework. The framework provides an adequate theoretical basis for developing hypotheses related to open online collaboration communities.

7.4. Conclusion

This study was a first look at open online collaboration communities. It defined open online collaboration communities as a special type of online communities, identified the fundamental policy problem that exists in such communities, identified the underlying structure that caused the policy problem, and analyzed the potential consequences of several policy options for addressing that problem.

The study integrated several theoretical approaches to the study of online communities and open source software development, built a dynamic feedback simulation model of a hypothetical open source software development community, tested the model under a range of external conditions and policy options, tested the applicability of the model and its policy implications to a specific instructional material development community, and integrated the implications of the initial model and the interviews to built a theoretical dynamic feedback framework for studying open online collaboration communities. The study contributed to several streams of literature, provided critical implications for practice, and laid a foundation for a wide range of potential future research studies.

APPENDIX A -- INTERVIEW PROTOCOL AND RELATED DOCUMENTS

A.1. Initial E-mail Request

Dear_____,

I am a PhD student at the University at Albany, working with David Andersen, George Richardson, Deborah Andersen, and Karl Rethemeyer. I am currently studying the efforts to develop teaching materials within the system dynamics K-12 community. I would like to carry out a telephone interview with you in order to gather information pertaining to this issue. The information I gather in the interview will be used as data in my dissertation. Your name or any information that might identify you as an individual will not be used in the dissertation, or elsewhere.

I expect the interview to last between 60 to 90 minutes. I would be happy to call you any day and time within the next two weeks, as long as my schedule permits. If you give your permission, I would like to tape the interview for detailed analysis of your answers. Please let me know, by replying to this e-mail, if you would accept doing such an interview, and if so when you would like to do it. Also, if you decide to do the interview, please fill out the attached consent form and fax or mail it to me. My contact information is given below. If you cannot open or print the form, I would gladly fax or mail you a copy.

Thank you

Sincerely,

Vedat Diker
Rockefeller College
University at Albany
Milne 300
135 Western Avenue
Albany, NY, 12222
Phone: (518) 442 3865
Fax: (518) 442 3398
E-mail: vd7606@albany.edu

A.2. Follow up E-mail Messages

Dear_____,

Thank you for voluntarily accepting to do a telephone interview with me about the efforts to develop teaching materials within the system dynamics K-12 community. As you suggest in your message, I will call you on _ (Date) _ at _ (Time) _ . [*Alternately:* Unfortunately, I am not able to call you on _ (Date) _ at _ (Time) _ . Please let me know if you would be available on _ (Date) _ at _ (Time) _ . If this does not work for you please suggest another date and time.] [*If the phone number is not provided:* Please let me know the phone number I should use to reach you.]

As I mentioned in my previous message I would like to tape the interview. In order to comply with applicable laws, I will ask your permission to tape the interview during the initial stage of our phone conversation. I will also ask your permission to quote the interview anonymously. As I mentioned before, your name or any information that might identify you as an individual will not be used in the dissertation, or elsewhere. Please refer to your copy of the consent form for details about confidentiality and your rights as a participant in this study.

If there is anything you would like to ask about the interview, please let me know. Thanks again for accepting my request.

Sincerely,

Vedat Diker

Dear _____,

I am sorry that you will not be able to do an interview with me. Thank you for considering my request, all the same.

Sincerely,

Vedat Diker.

A.3. Interview Packet Cover Letter

Dear _____,

Thank you once again for accepting to participate in my dissertation research. Please find enclosed the hardcopy of the consent form, along with a stamped and addressed return envelope. You may use the return envelope to mail the signed consent form if you have not mailed or faxed it yet. Also enclosed are the reference mode worksheets, and diagrams that we will use during the interview. Please do not open the smaller envelope, which holds the diagrams, until you are prompted to do so during the interview. Please keep the reference mode worksheets, the diagram envelope, and a pen or a pencil close by during the interview. A second return envelope is enclosed for you to mail the filled-out reference mode worksheets after the interview. Please contact me at vd7606@albany.edu or (518) 235 7048 if you have any questions or concerns.

Sincerely,

Vedat Diker

A.4. Participation in Research Consent Form

STUDY WORKING TITLE: Toward a Theory of Open Online Collaboration

RESEARCHER: Vedat G. Diker (University at Albany, Ph.D. Program in Information Science)

STUDY DESCRIPTION

My dissertation research is aimed toward exploring the mutual relationships between factors that have potential of affecting the success of open online collaborative projects, (e.g., motivation, participation, product quality and product functionality). The ultimate purpose of the research is to articulate a theory of open online collaboration phenomena, in the form of a dynamic feedback framework.

I concluded that the community of researchers and practitioners who are applying system dynamics concepts to K-12 education, and their efforts for developing and sharing instructional material on the Internet would be an excellent case for exploration, for the purposes of my research. Since you are an important figure within that community, I would very much like to carry out a telephone interview with you. During the interview I will ask you several questions about the instructional material development projects within the system dynamics K-12 community. I expect the interview to last about one and a half hours.

Your participation in the study is completely voluntary, so you may stop and discontinue the interview any time you wish without any adverse consequences on your part. You may also choose not to answer any questions you do not wish to for any reason.

In order to keep a better record of the interview, I would like to tape the telephone conversation, though this is not mandatory. The tapes of the telephone conversation will be stored in my home, in a locked box, and will not be made available to anybody except myself and the below listed members of my dissertation committee.

While I may quote from interviews in my dissertation or any other future publication related to this research, I will not identify you or your organization in any quote or opinion. The tapes and the transcript of the interview will not be made available to other researchers for secondary analysis or any other research purposes without your written consent. I will keep all records that identify you private to the extent allowed by law. However, officials from the federal government and/or the University at Albany may inspect the records that identify you for the purpose of protecting your rights as a human participant.

While I cannot promise you any direct benefit from your participation in this study, I hope that this study will provide more information on the dynamics of open online communities. This information may help us develop policies that would increase the success of such projects in the future.

I will report my findings in my dissertation, which I expect to complete in August 2003. I can provide you with an abstract and/or an electronic copy of the dissertation when it is completed, if you would like.

My contact information is below, as is my dissertation committee members' information, if you would like to discuss this research. If you would like to be interviewed, please sign this form, and mail or preferably fax it to me.

CONSENT

If you agree to be interviewed for my research, please sign below.

_____ ____ / ____ / _____

Name Date

Furthermore, if you agree to me taping the telephone conversation, please sign below.

_____ ____ / ____ / _____

Name Date

CONTACT INFORMATION

I am a doctoral student in the Information Science Program at the University at Albany. The above mentioned interview will provide data for my dissertation. My contact information follows.

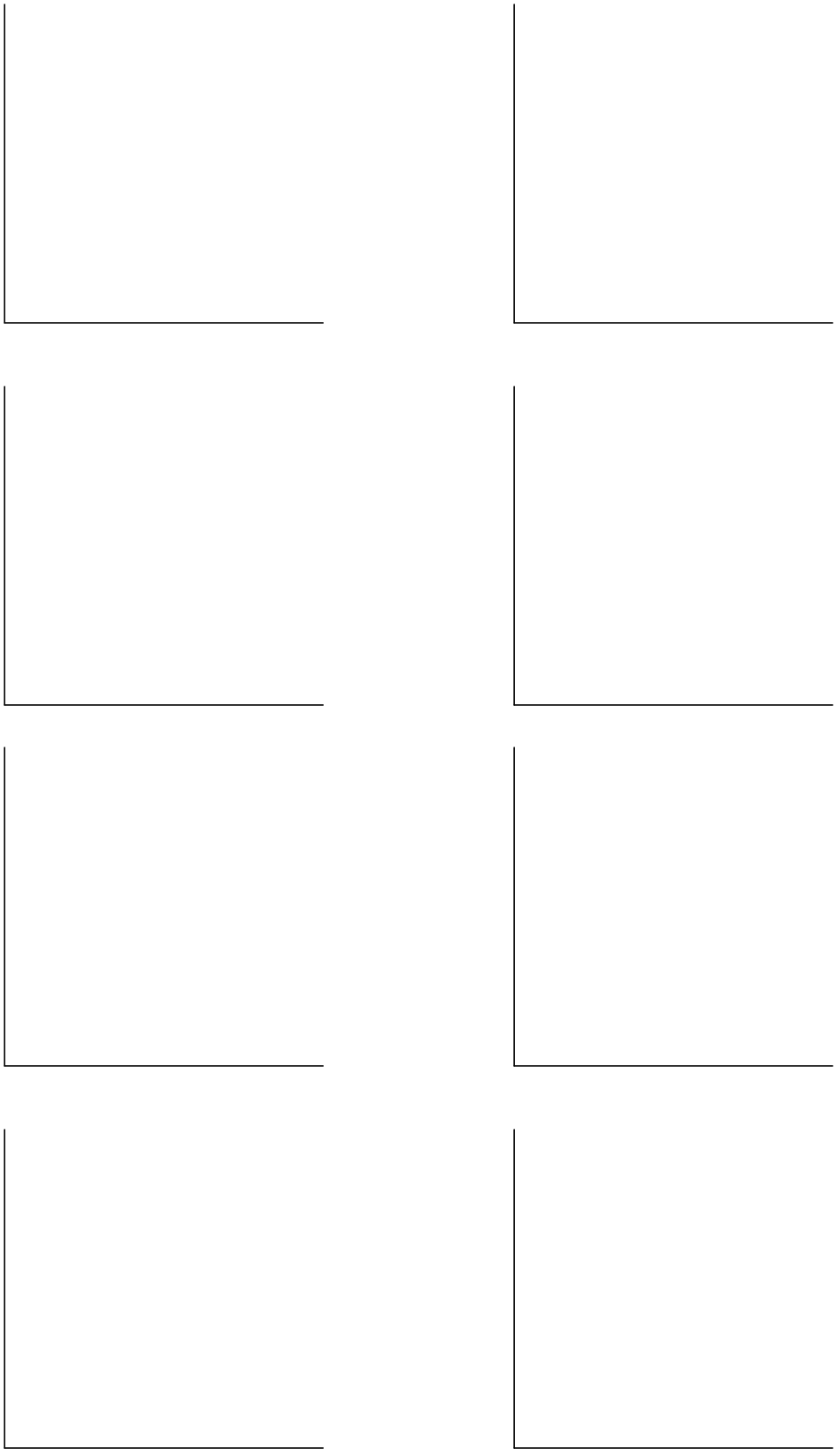
Vedat G. Diker Rockefeller College University at Albany Milne 300 135 Western Avenue Albany, NY, 12222	 Phone: (518) 442 3865 Fax: (518) 442 3398 E-mail: vd7606@albany.edu

Dissertation Committee:

David F. Andersen (co-chair) Rockefeller College University at Albany Milne 315-B 135 Western Avenue Albany, NY, 12222 (518) 442 5280 david.andersen@albany.edu	**George P. Richardson** (co-chair) Rockefeller College University at Albany Milne 318 135 Western Avenue Albany, NY, 12222 (518) 442 3859 gpr@albany.edu
Deborah L. Andersen School of Infornmation. Science and Policy University at Albany Draper 113 135 Western Avenue Albany, NY, 12222 (518) 442 5115 dla@albany.edu	**R. Karl Rethemeyer** Rockefeller College University at Albany Milne 312-A 135 Western Avenue Albany, NY, 12222 (518) 442 5258 kretheme@albany.edu

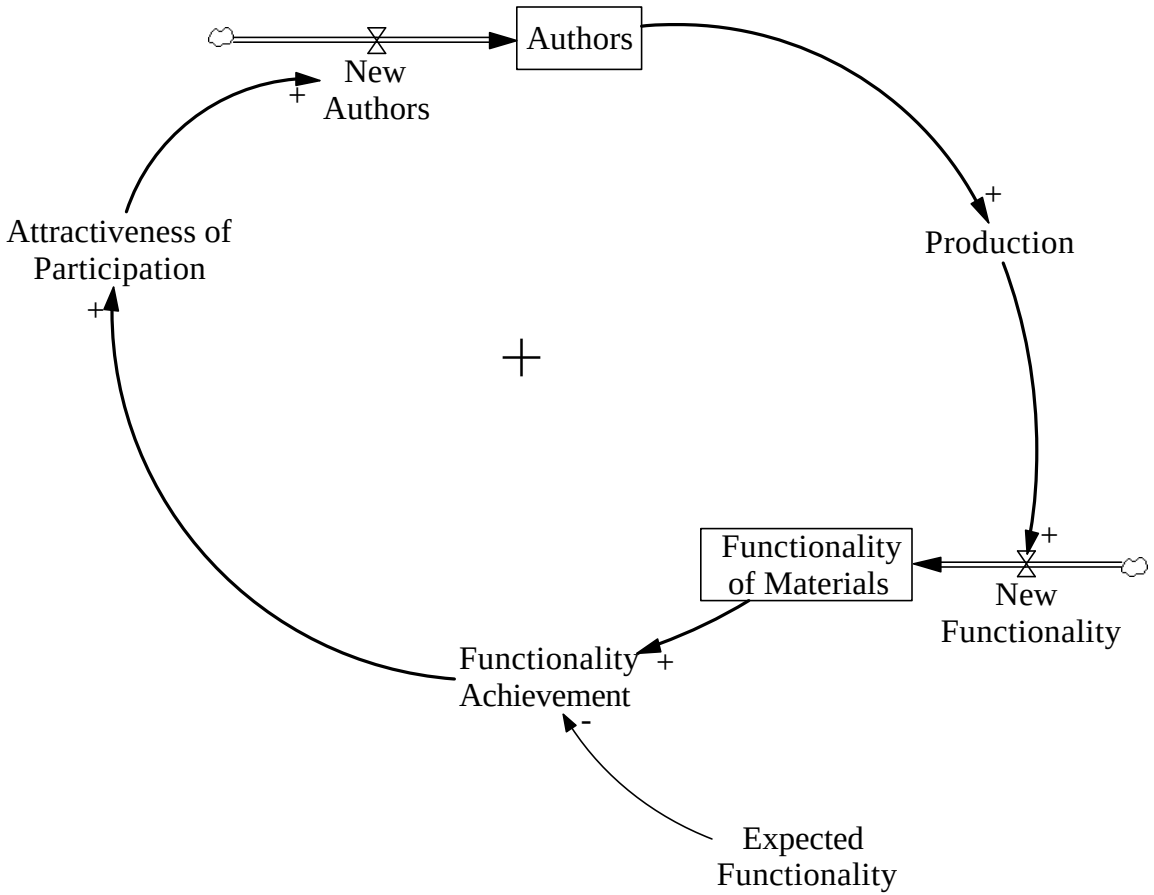
If you have any questions regarding your rights as a participant, contact the Compliance Office, Office for Sponsored Programs, at (518) 437-4569.

A.5. Reference Mode Worksheet

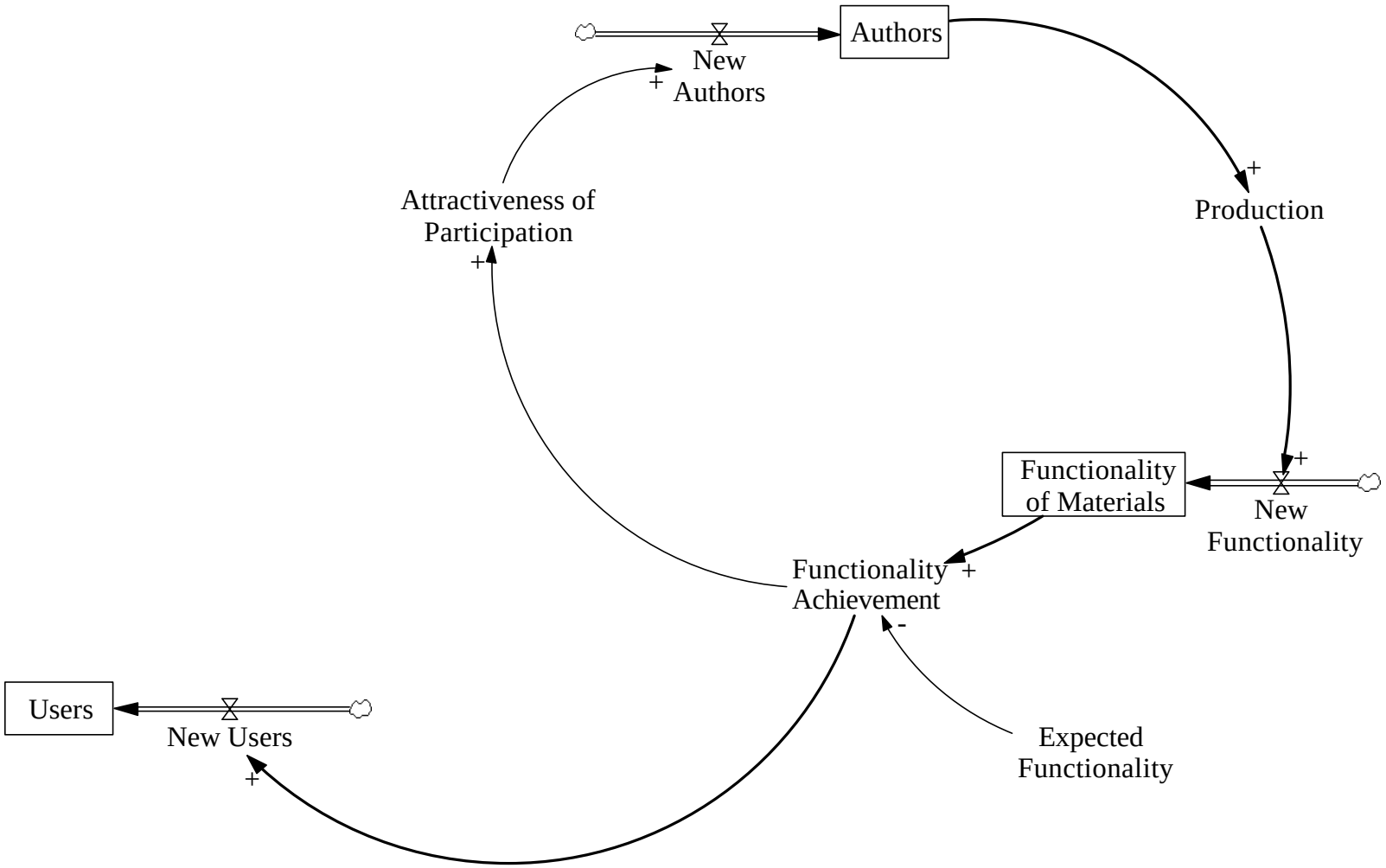


A.6. Model Sketches

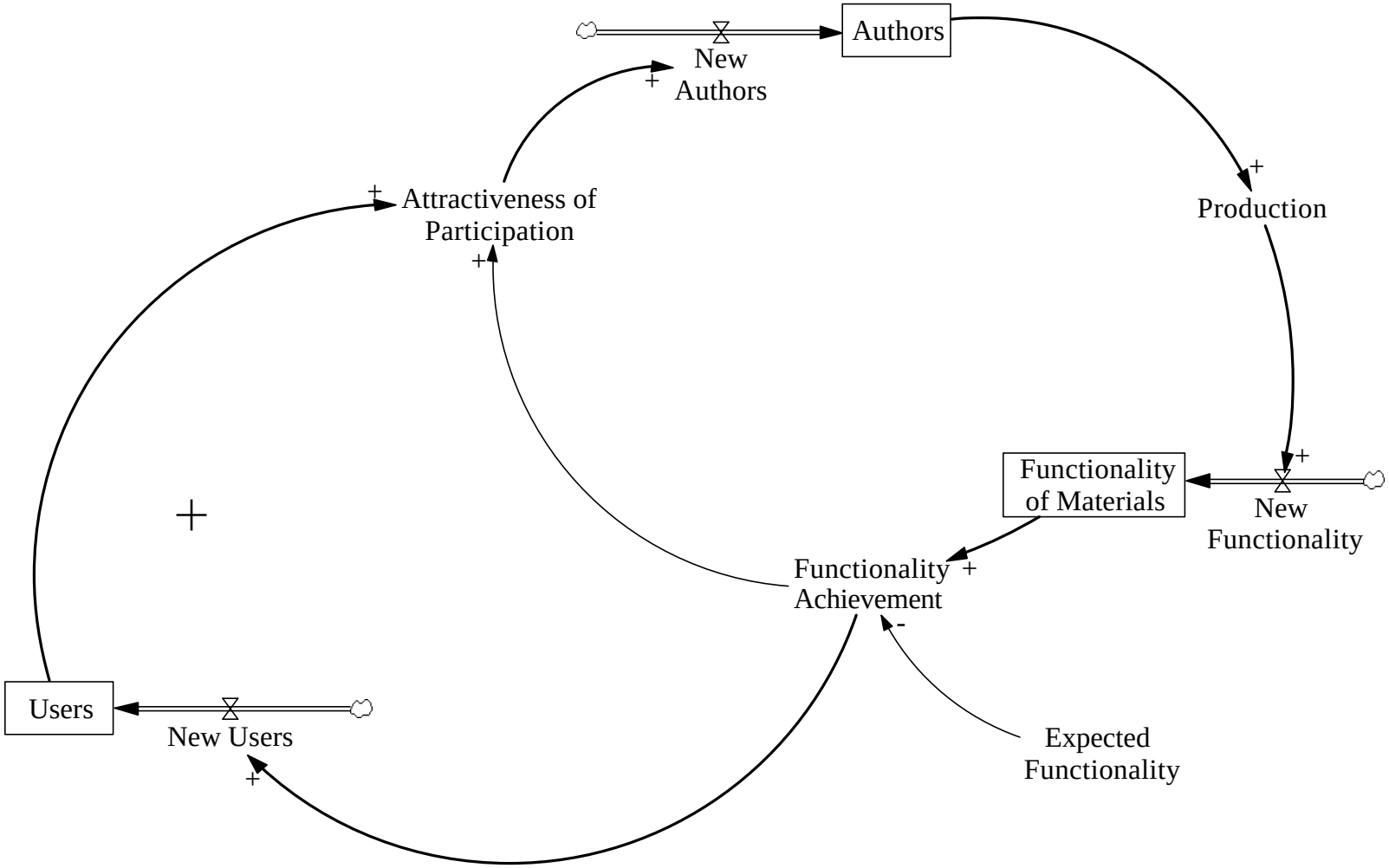
Sketch 0.1.



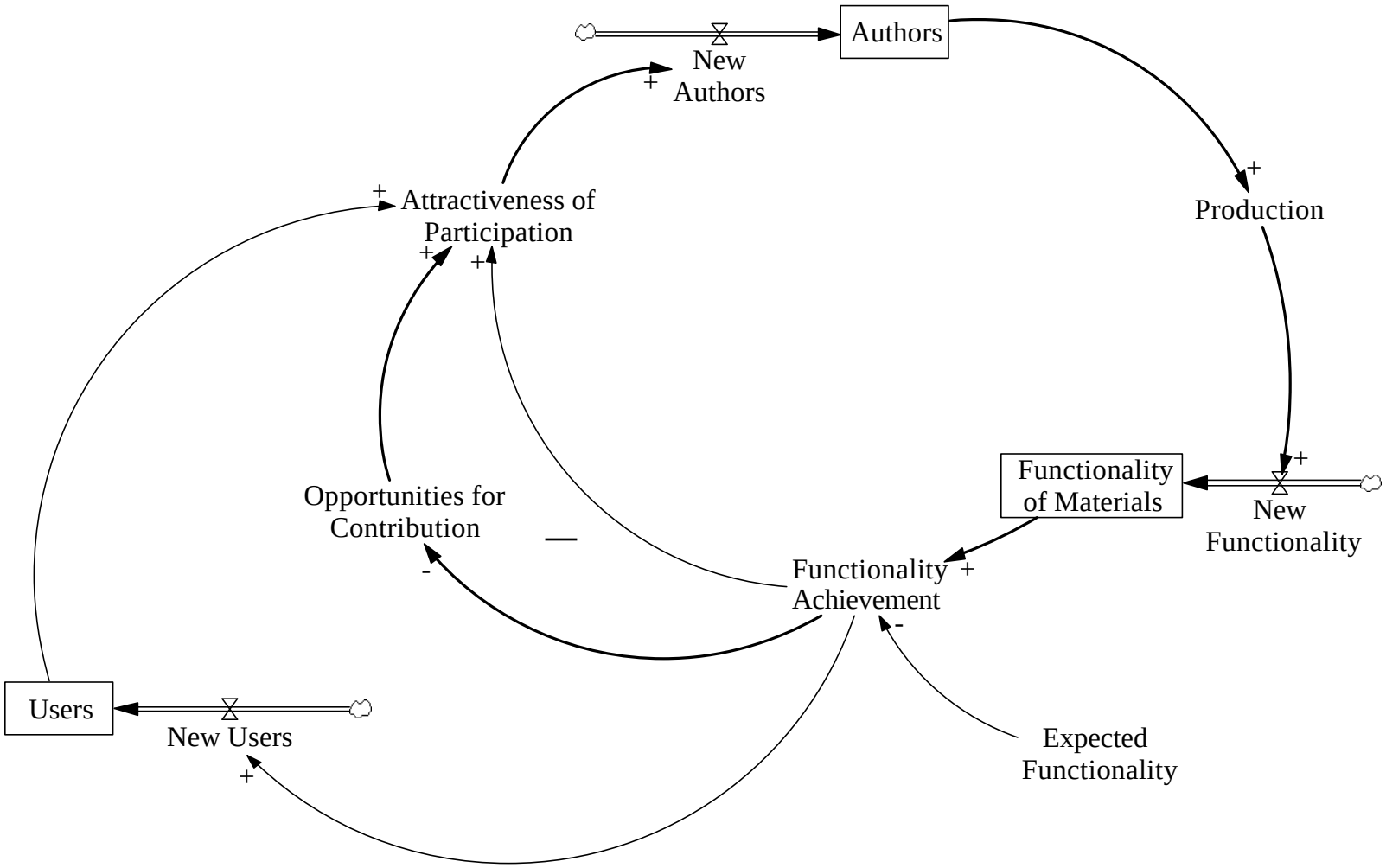
Sketch 0.2.


Authors
New Authors
+
Attractiveness of Participation
+
Production
+
Functionality of Materials
+
New Functionality
Functionality Achievement
+
-
Users
New Users
+
Expected Functionality

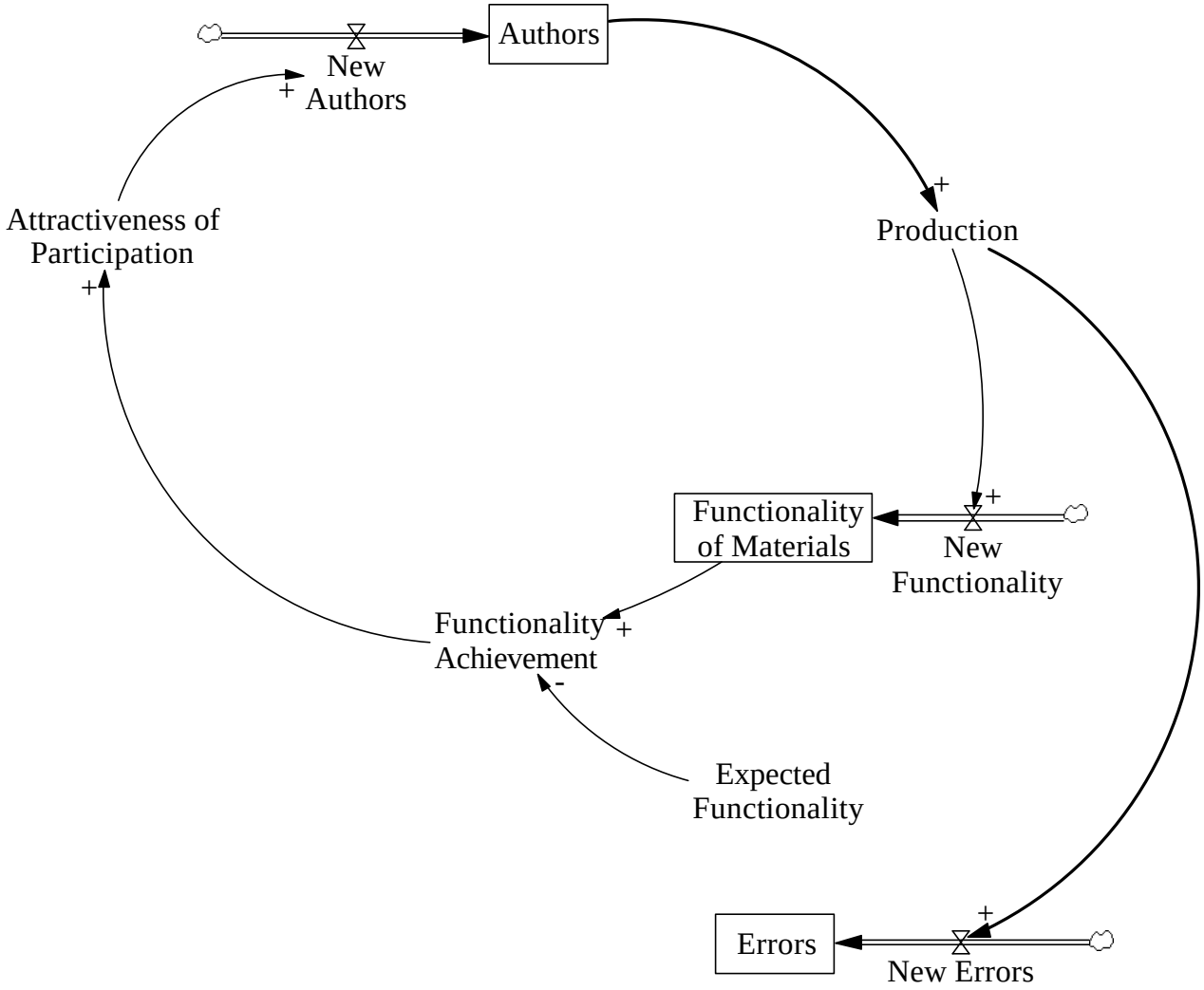
Sketch 0.3.


Authors
New
Authors
Attractiveness of
Participation
Production
Functionality
of Materials
New
Functionality
Functionality
Achievement
Users
New Users
Expected
Functionality

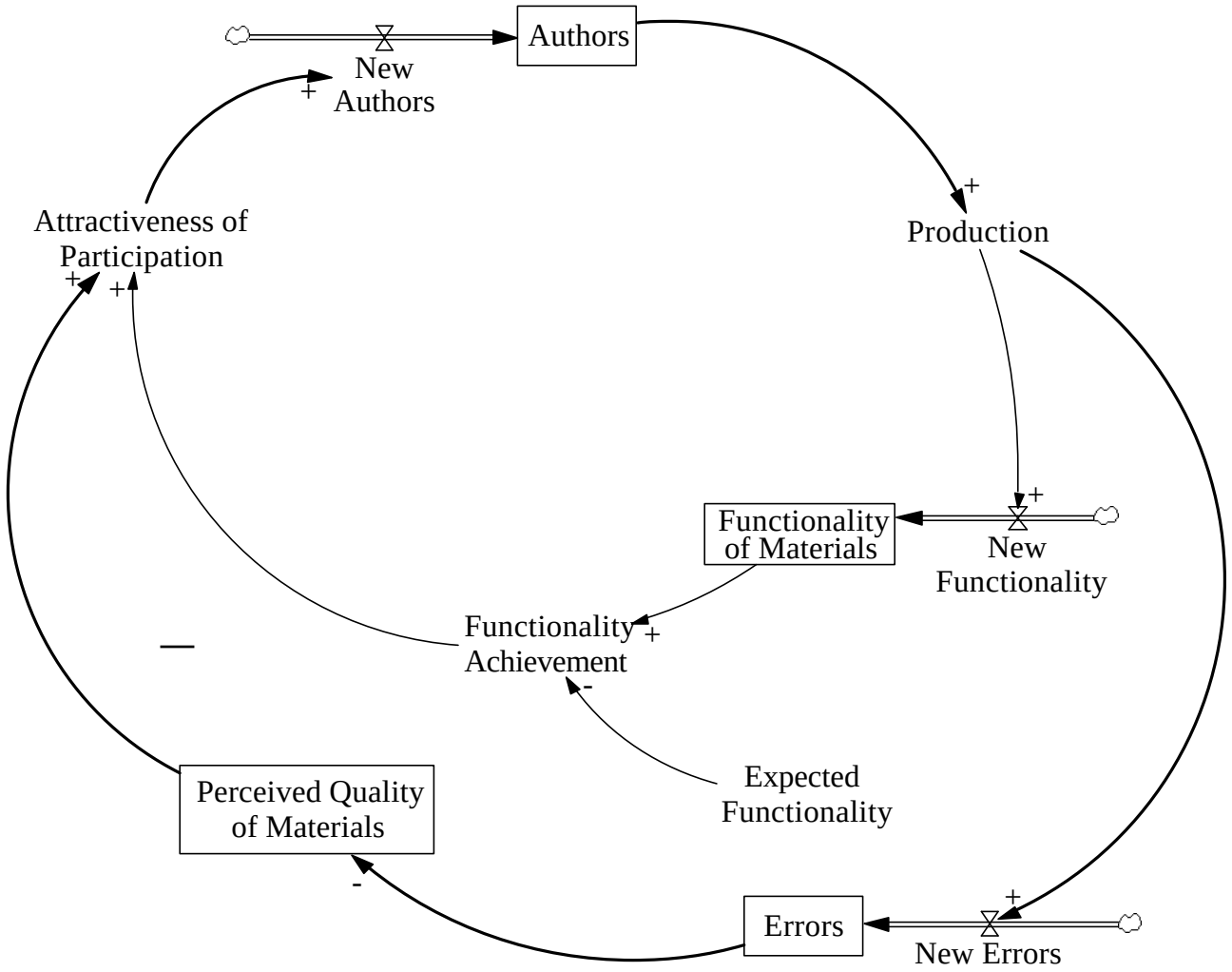
Sketch 0.4.


Authors
New
Authors
Attractiveness of
Participation
Production
Opportunities for
Contribution
Functionality
of Materials
New
Functionality
Functionality
Achievement
Users
New Users
Expected
Functionality

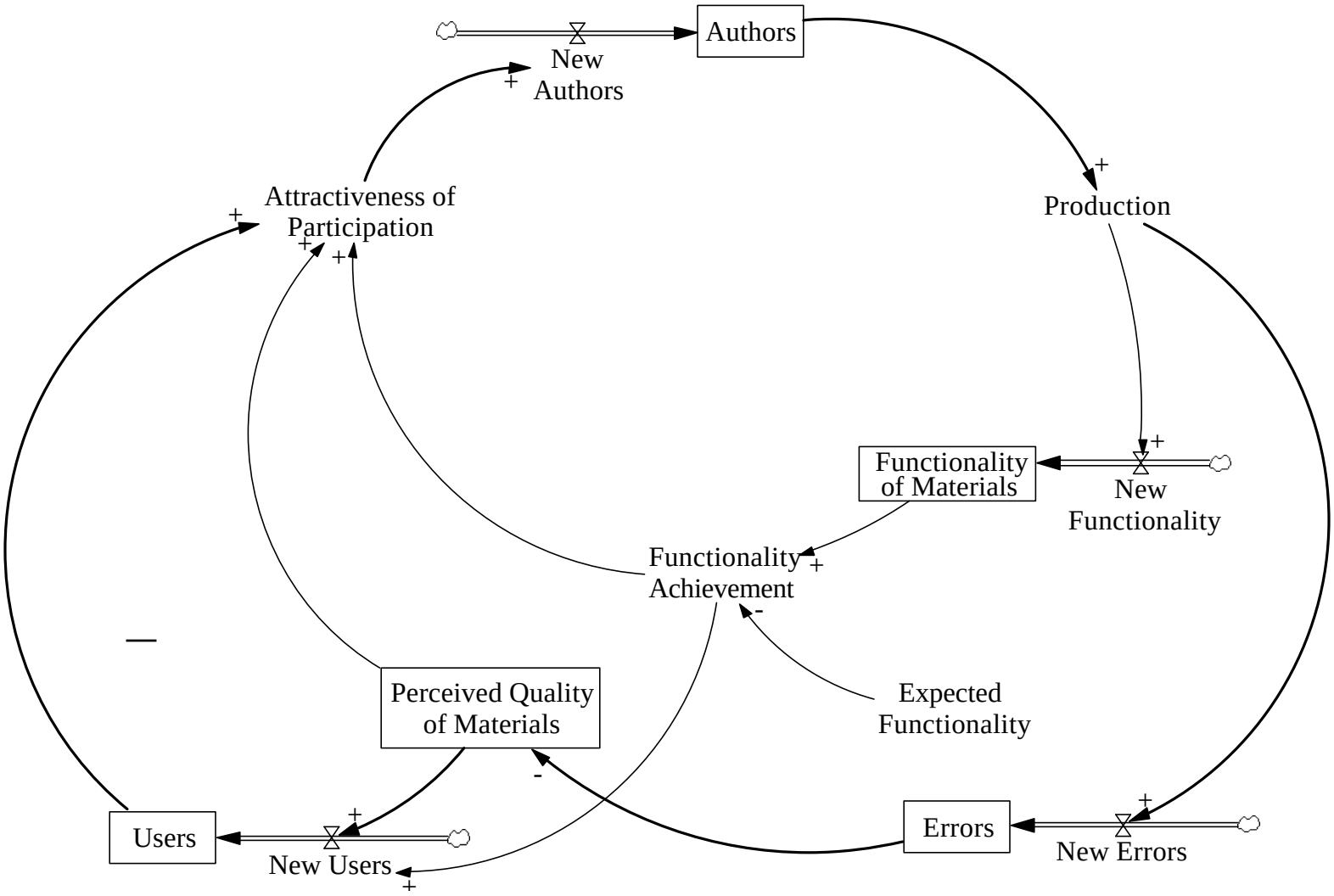
Sketch 0.5.


Authors
New
Authors
+
Attractiveness of
Participation
+
+
Production
Functionality
of Materials
+
New
Functionality
Functionality
Achievement
+
-
Expected
Functionality
+
Errors
New Errors

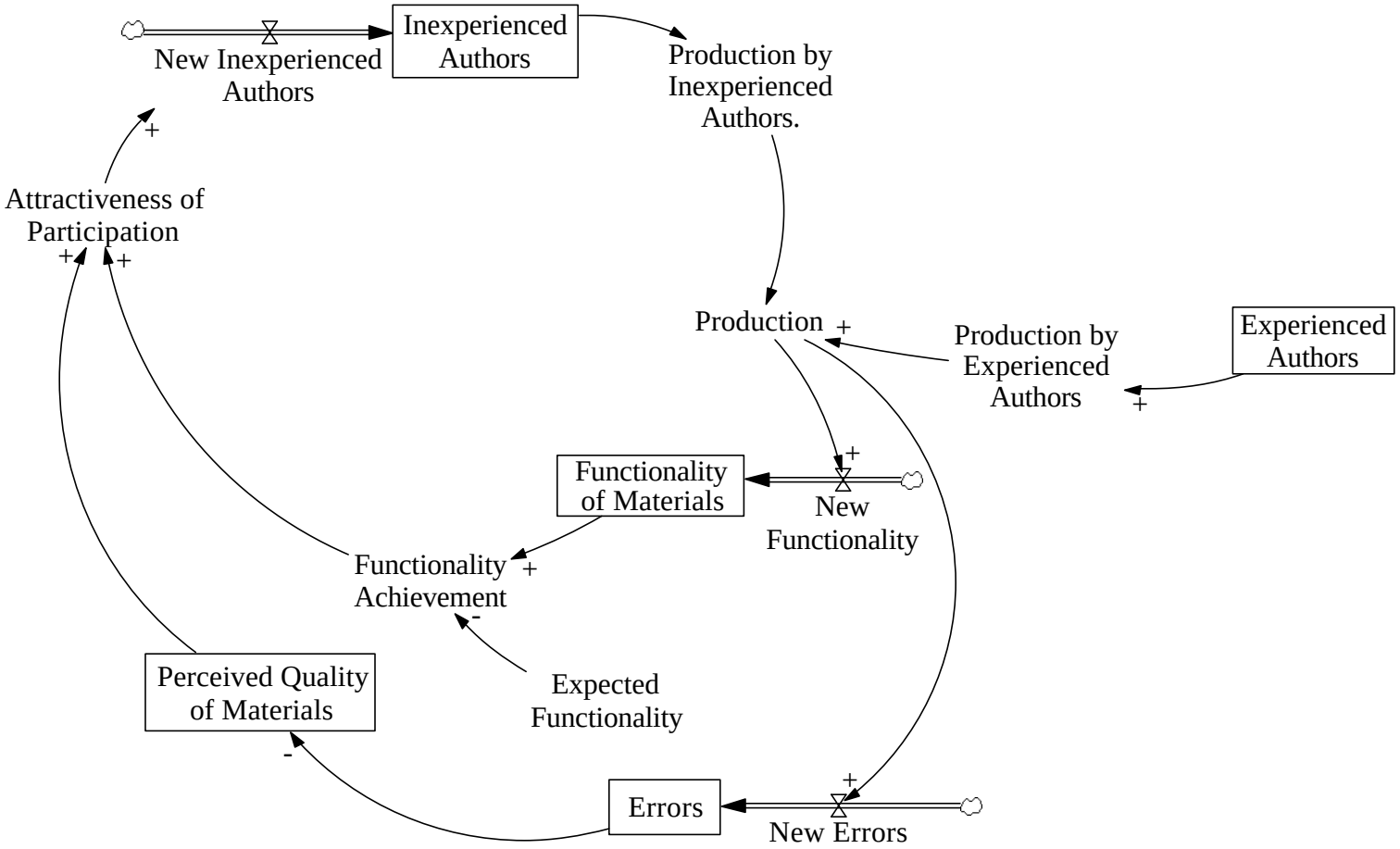
Sketch 0.6.


Authors
New
Authors
+
Attractiveness of
Participation
+
+
+
Production
+
Functionality
of Materials
New
Functionality
Functionality
Achievement
+
-
Expected
Functionality
Perceived Quality
of Materials
-
+
Errors
New Errors

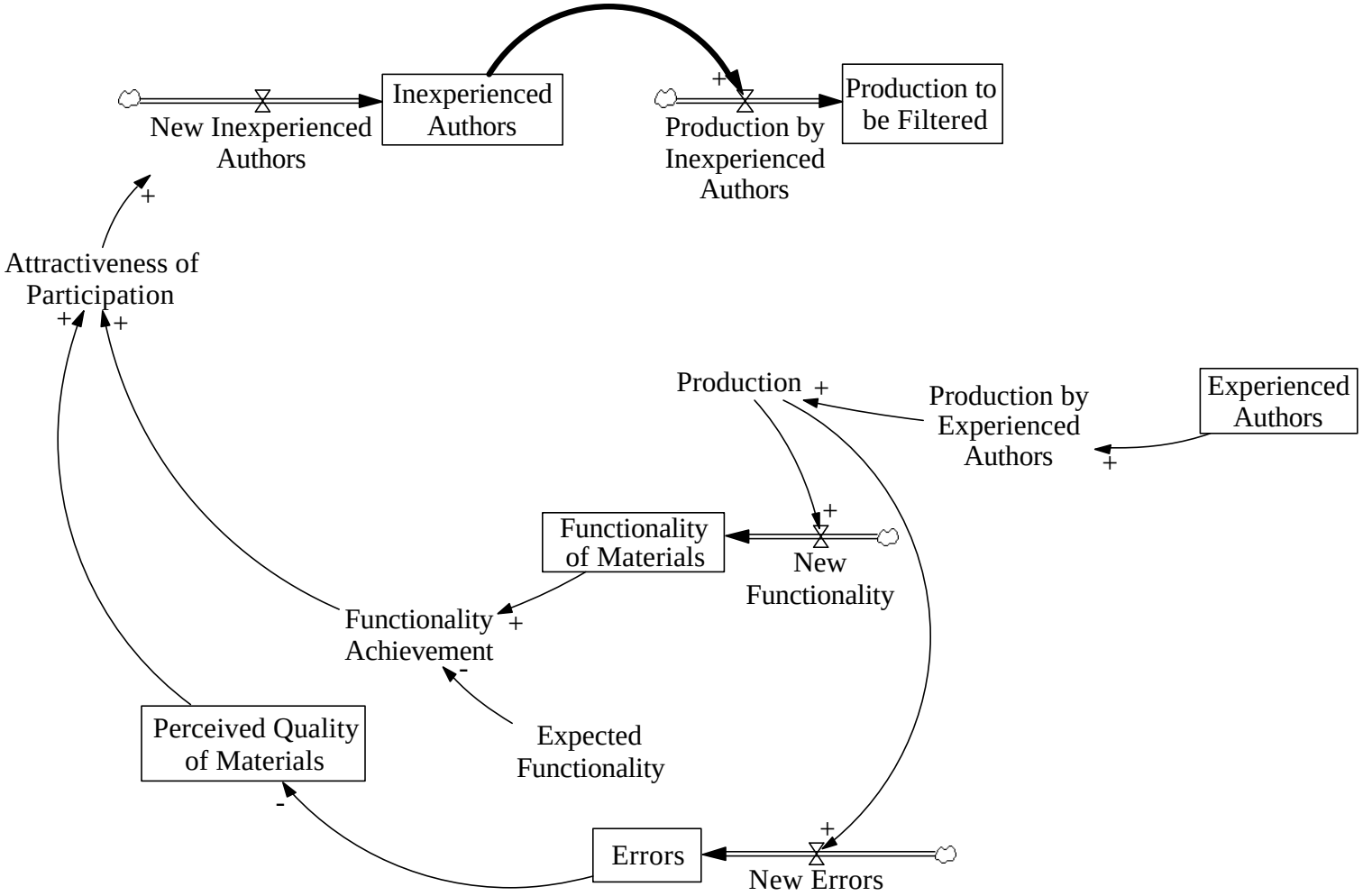
Sketch 0.7.


Authors
New
Authors
Attractiveness of
Participation
Production
Functionality
of Materials
New
Functionality
Functionality
Achievement
Perceived Quality
of Materials
Expected
Functionality
Users
New Users
Errors
New Errors

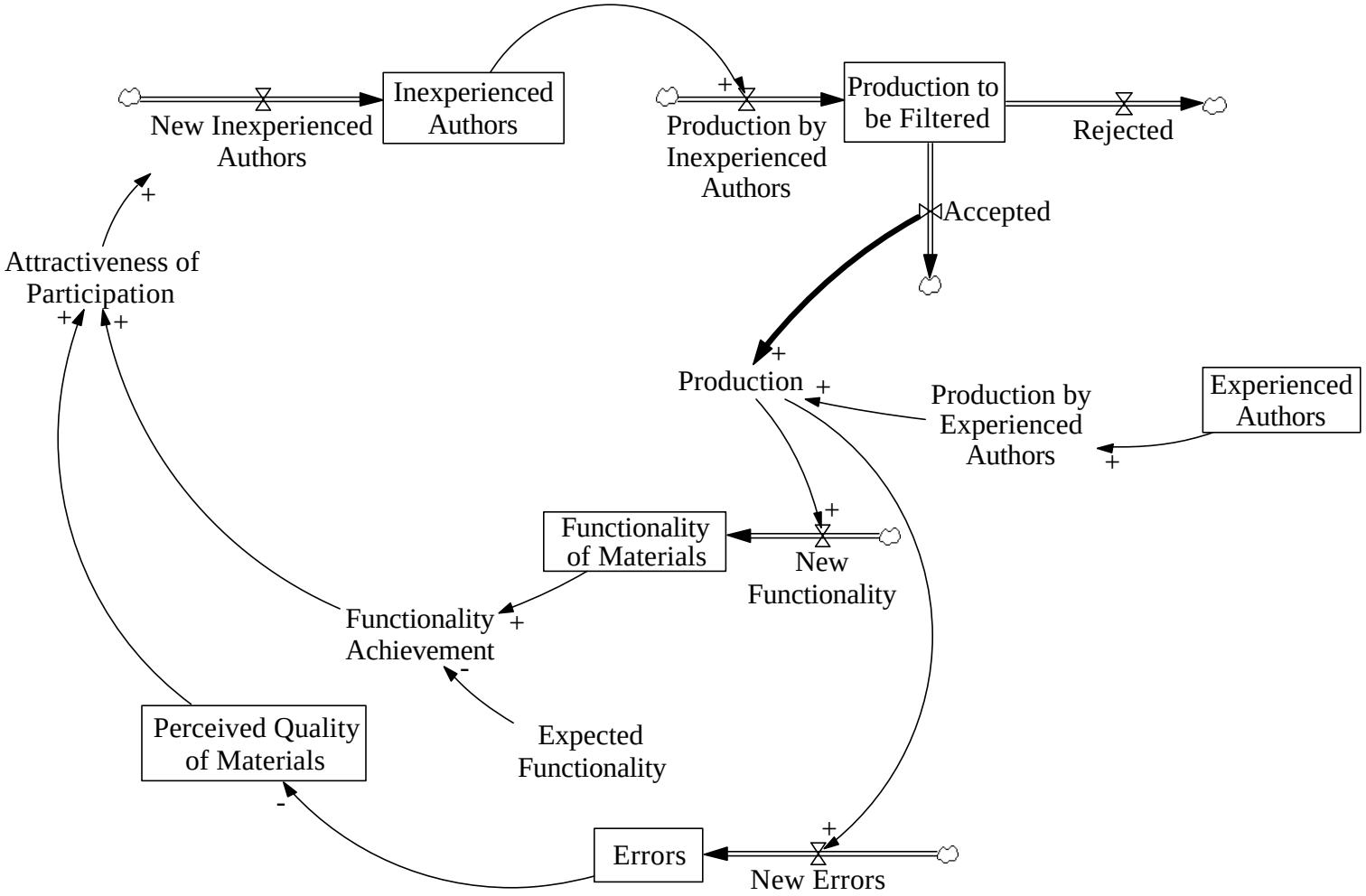
Sketch 1.0.


Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors.
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Perceived Quality of Materials
Expected Functionality
Errors
New Errors

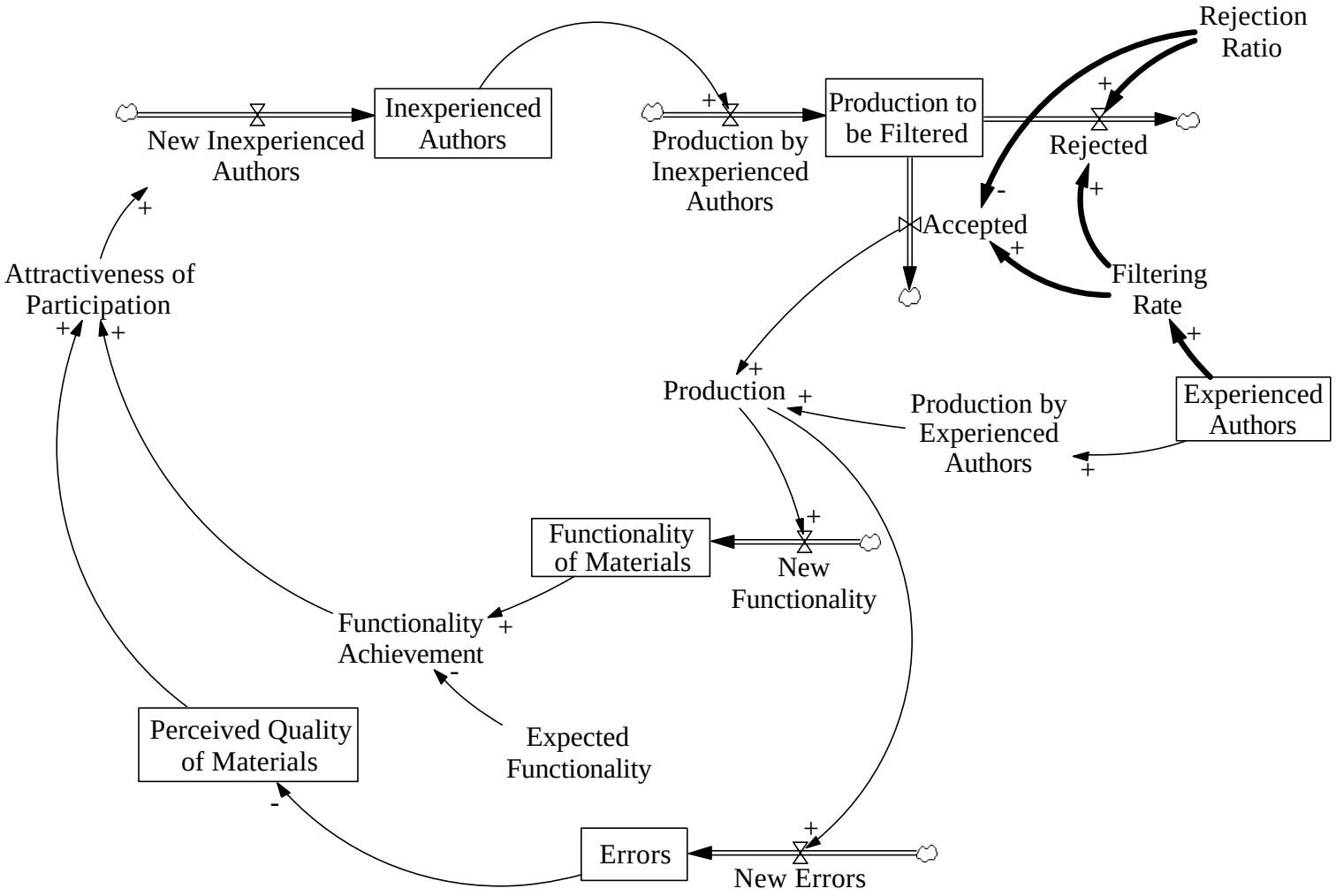
Sketch 1.1.


Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

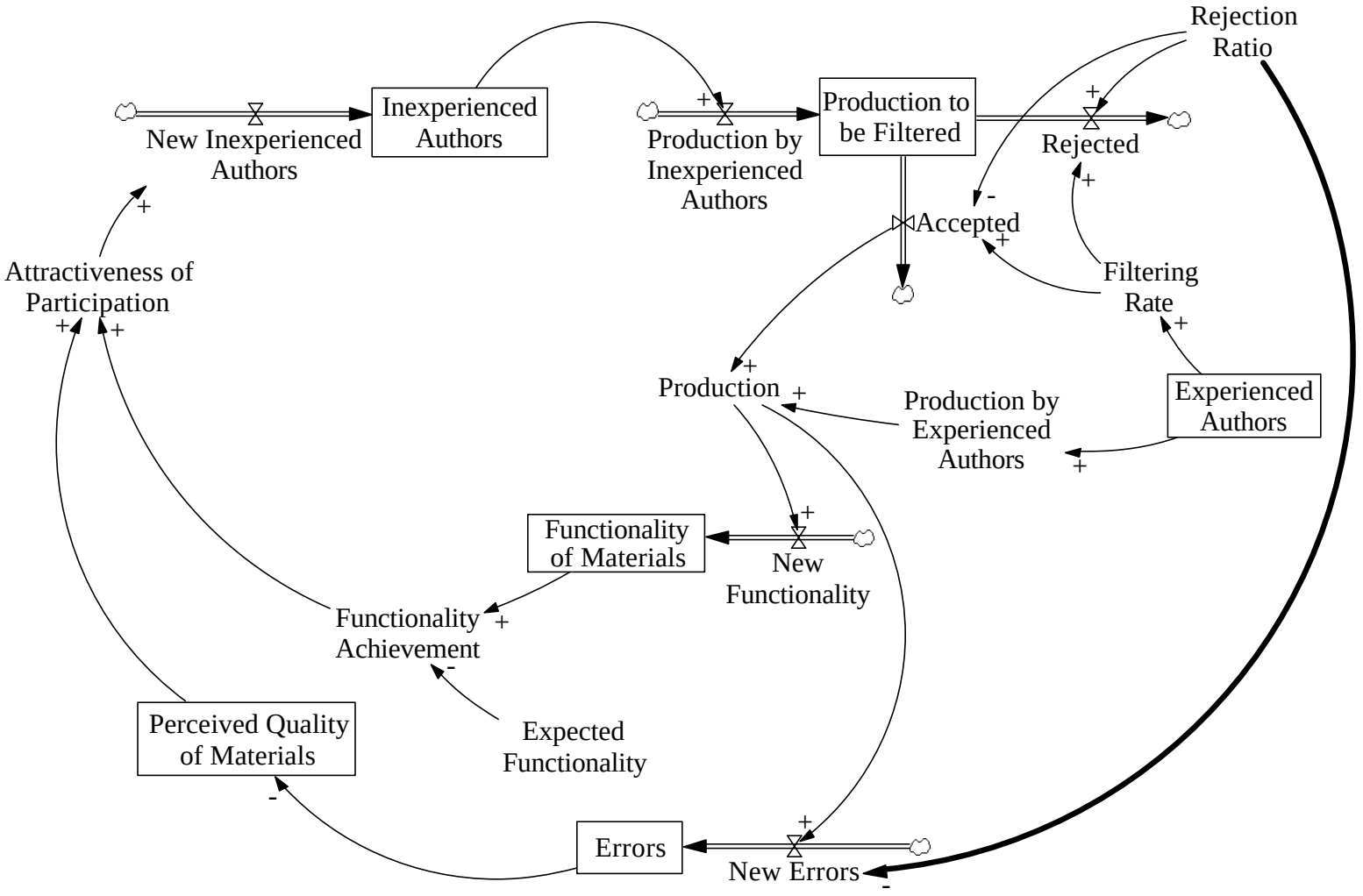
Sketch 1.2.


Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Rejected
Accepted
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Perceived Quality of Materials
Expected Functionality
Errors
New Errors

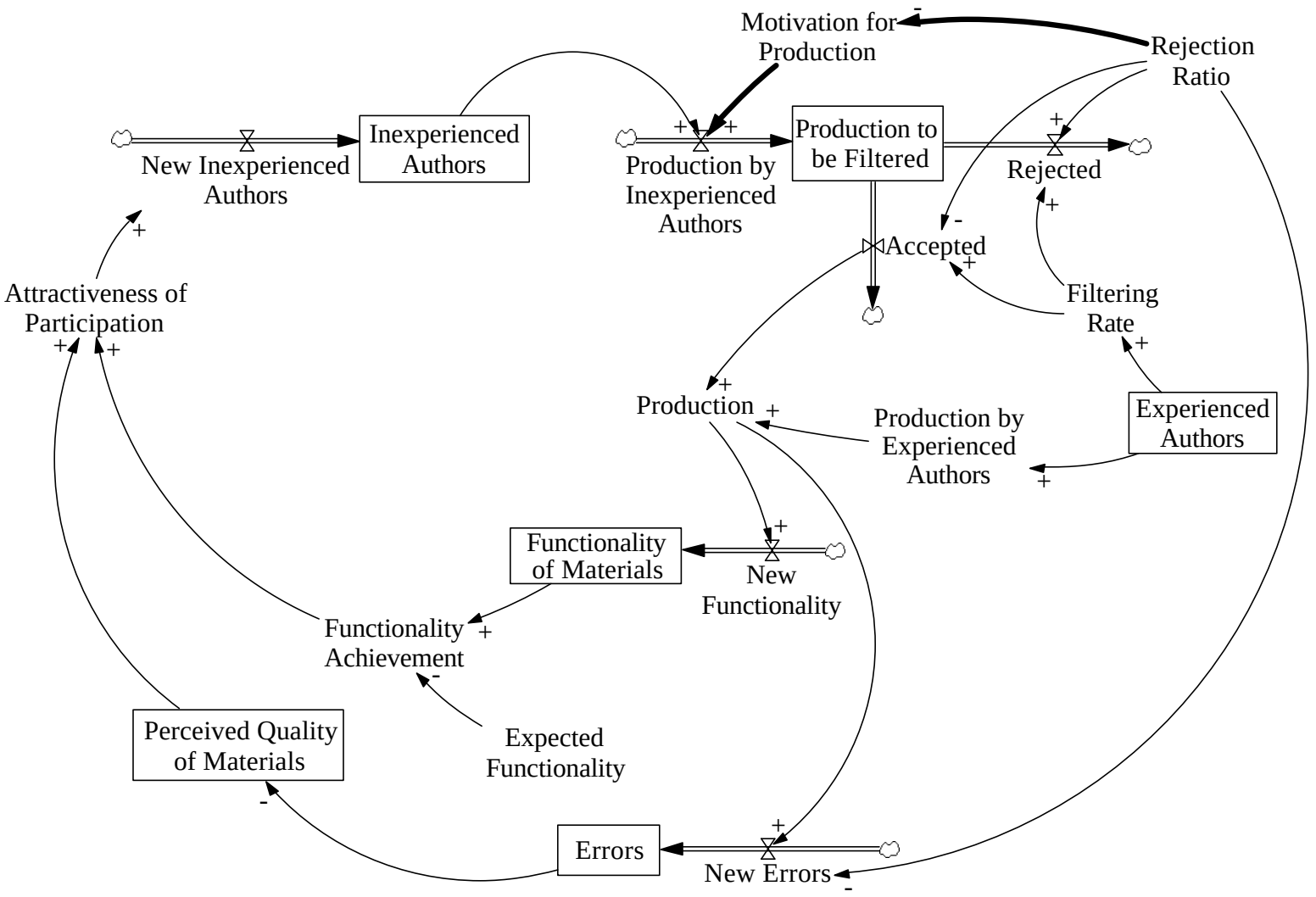
Sketch 1.3.


Rejection Ratio
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Rejected
Accepted
Filtering Rate
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Perceived Quality of Materials
Expected Functionality
Errors
New Errors

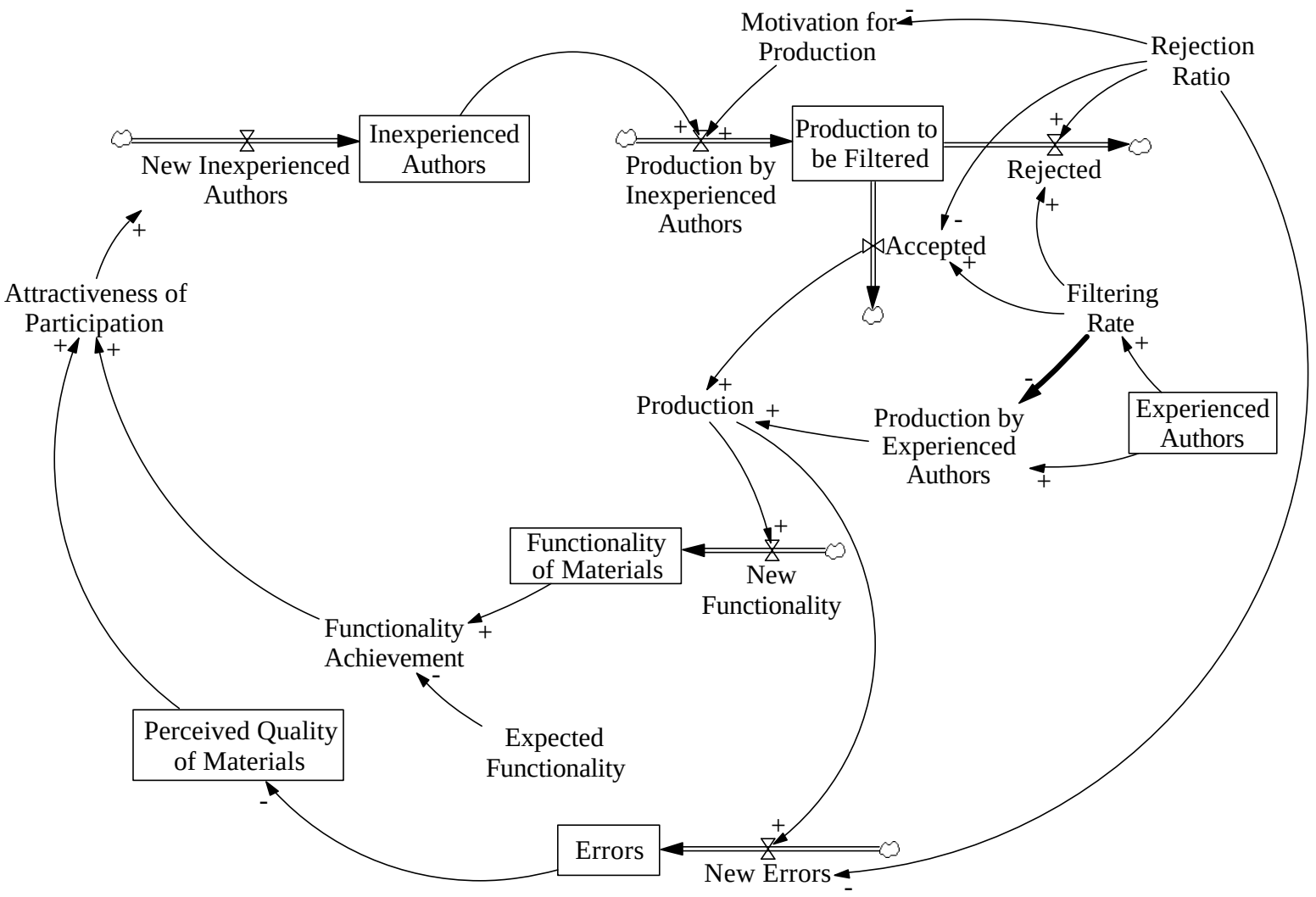
Sketch 1.4.


Rejection Ratio
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Rejected
Accepted
Filtering Rate
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Perceived Quality of Materials
Expected Functionality
Errors
New Errors

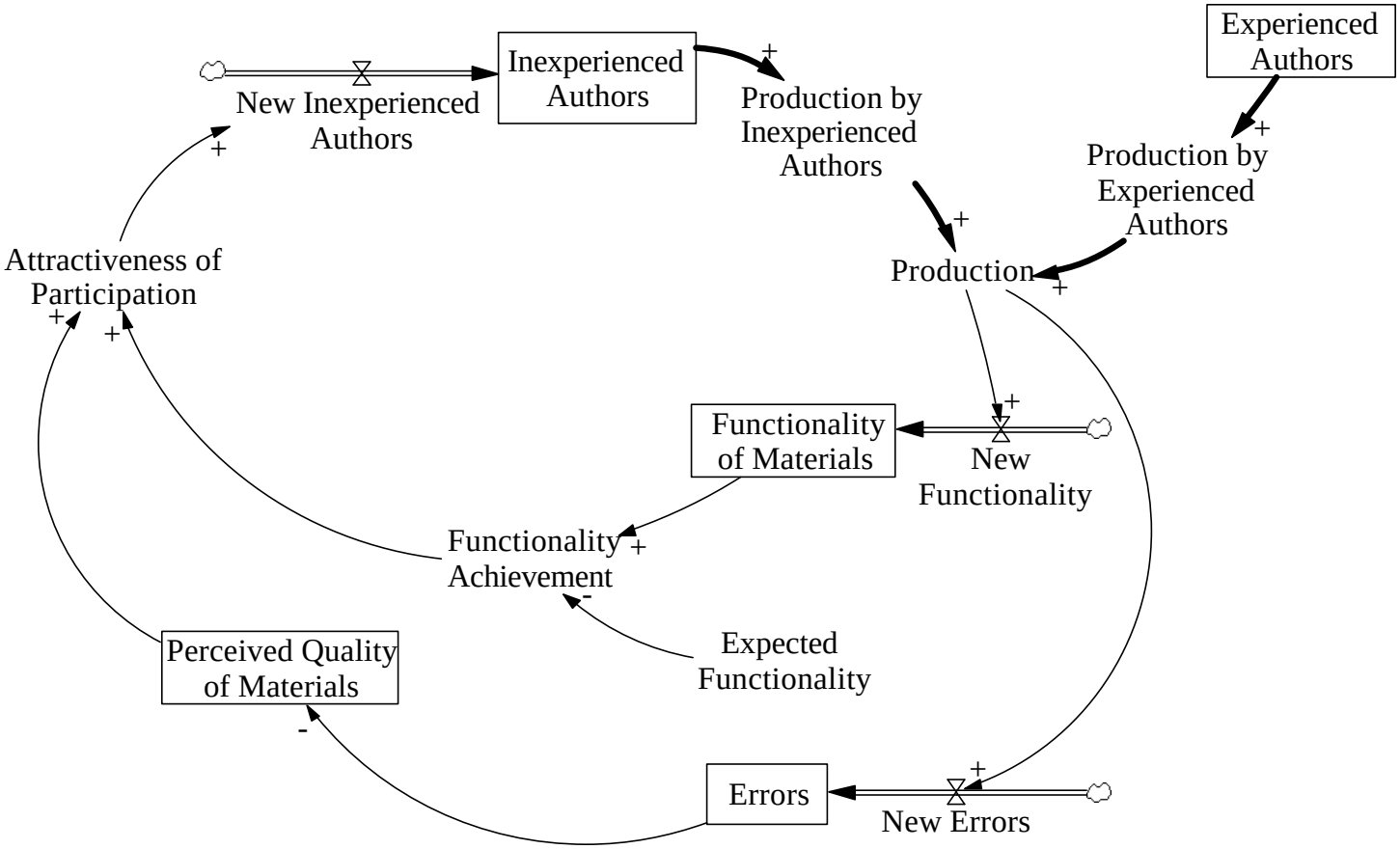
Sketch 1.5.


Motivation for Production
Rejection Ratio
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Rejected
Accepted
Filtering Rate
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

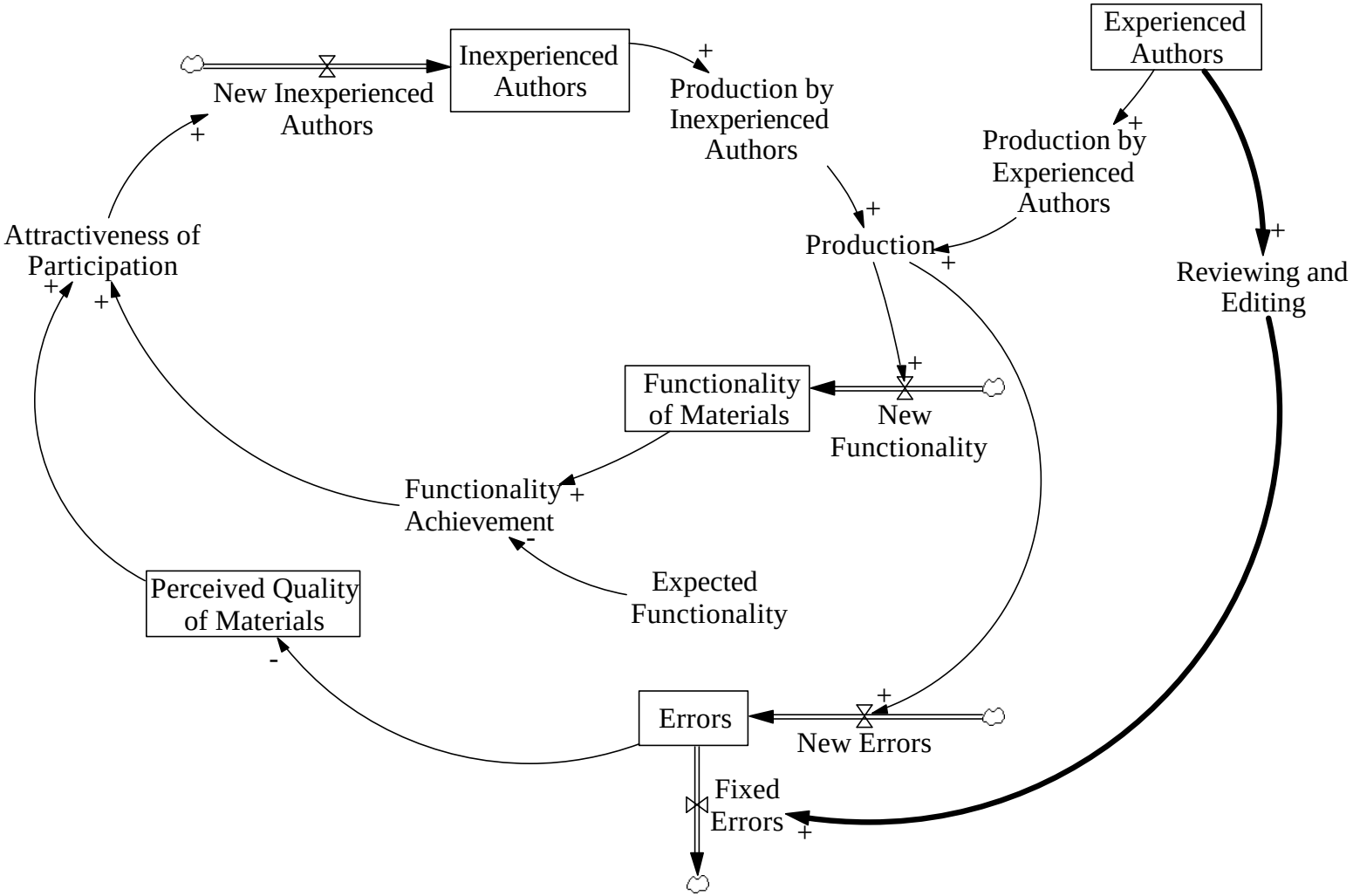
Sketch 1.6.


Motivation for Production
Rejection Ratio
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production to be Filtered
Rejected
Accepted
Filtering Rate
Attractiveness of Participation
Production
Production by Experienced Authors
Experienced Authors
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

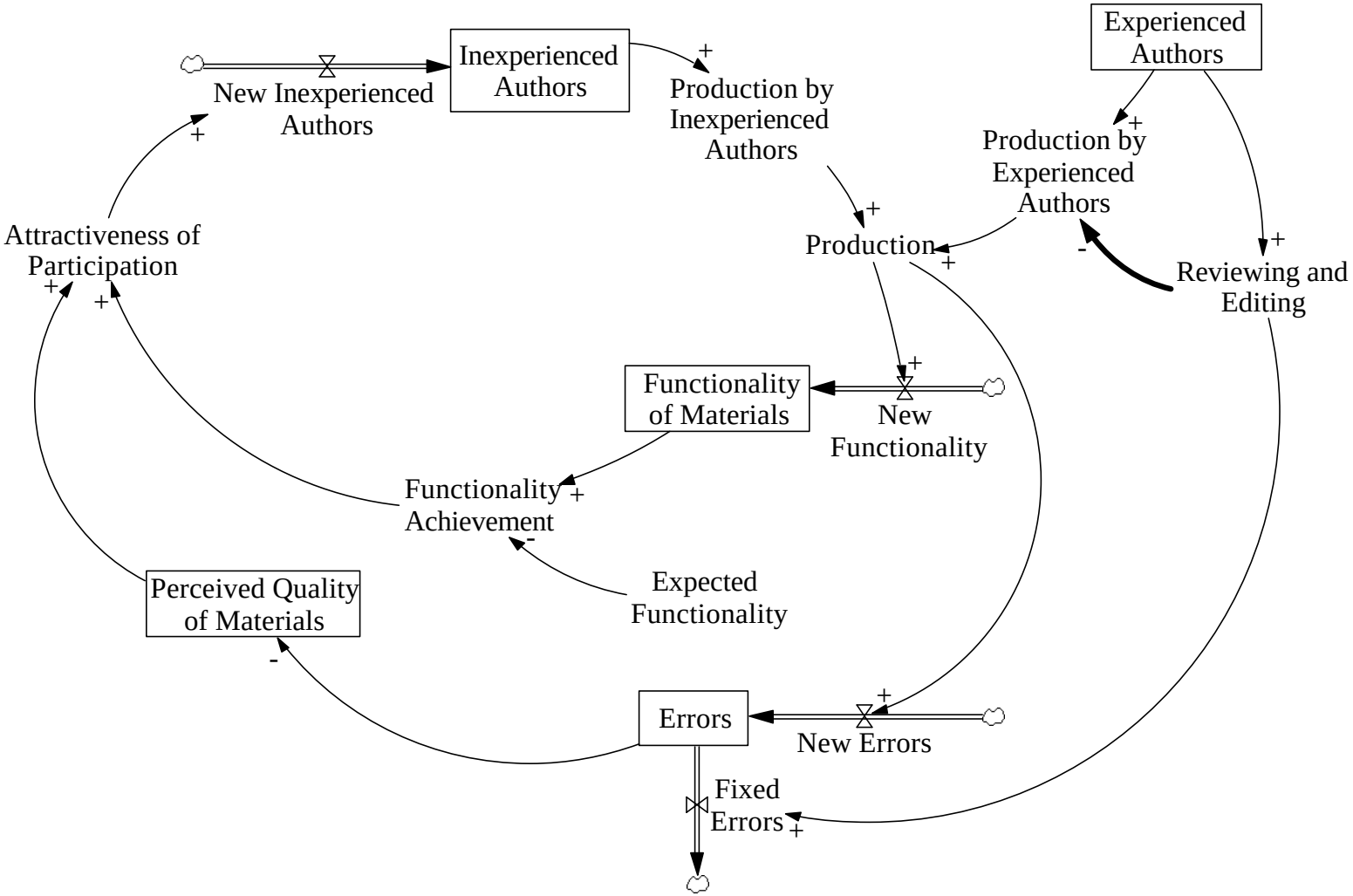
Sketch 2.1.


Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

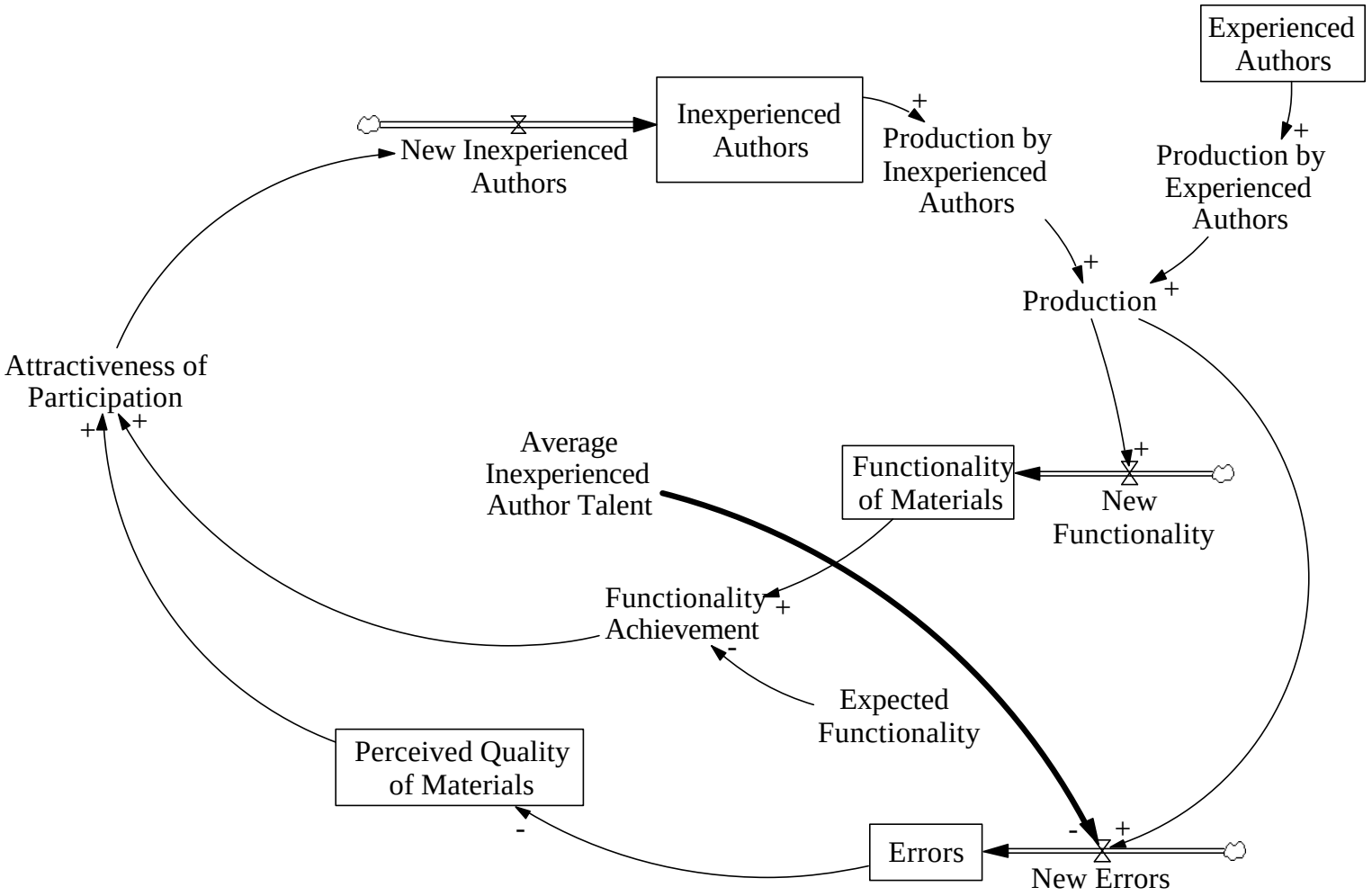
Sketch 2.2.


Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Reviewing and Editing
Attractiveness of Participation
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors
Fixed Errors

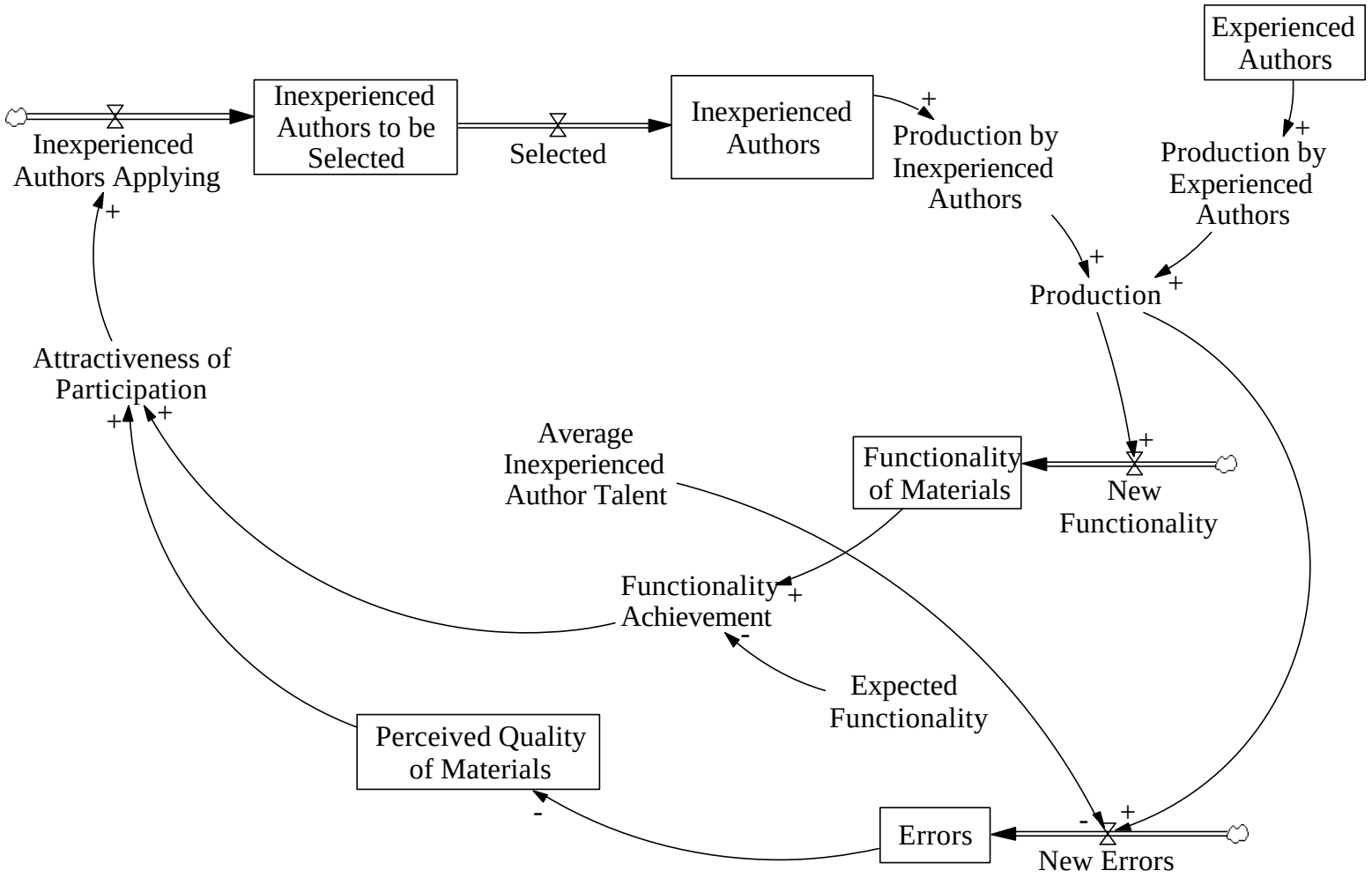
Sketch 2.3.


Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Attractiveness of Participation
Production
Reviewing and Editing
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors
Fixed Errors

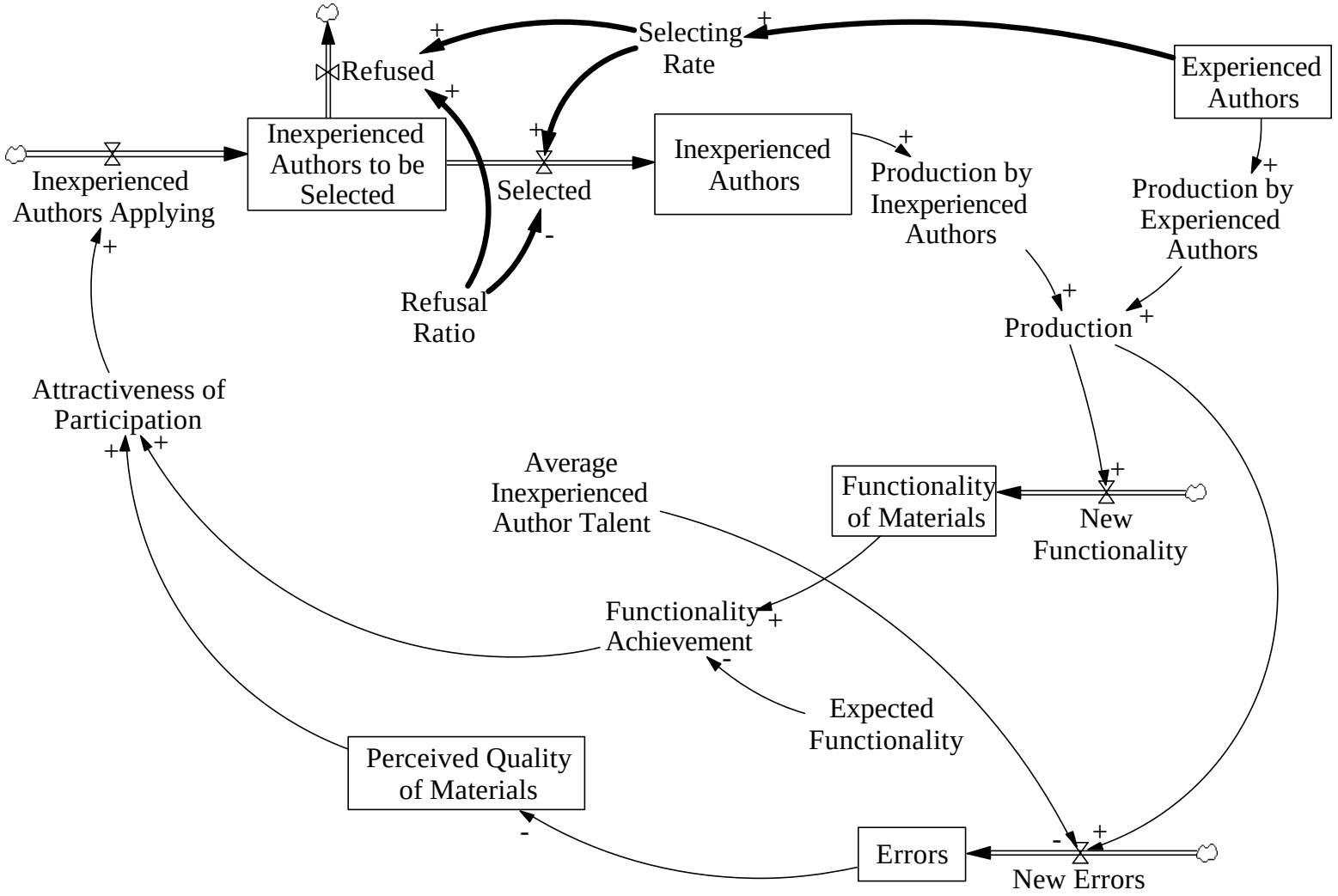
Sketch 3.1.


Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

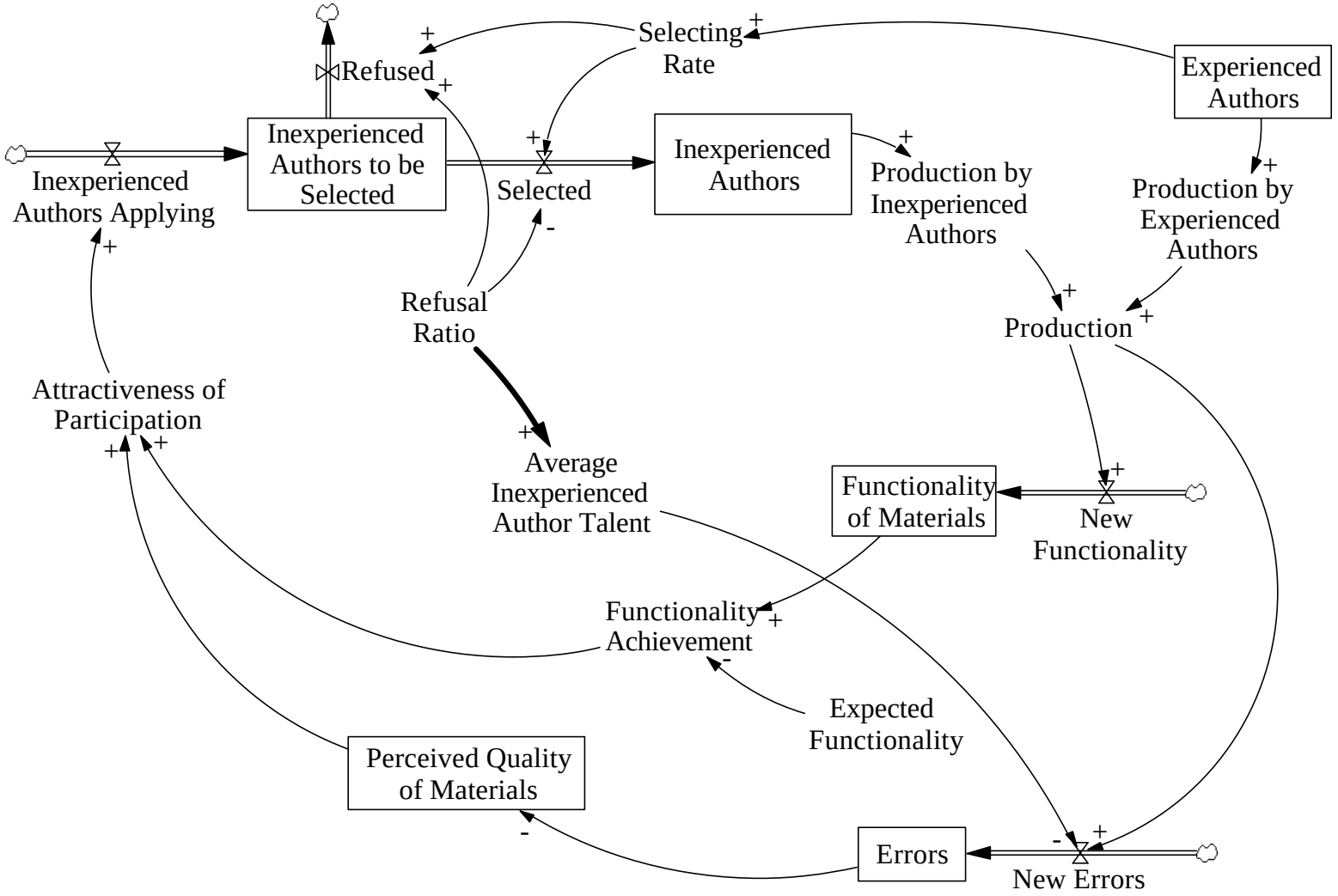
Sketch 3.2.


Experienced Authors
Inexperienced Authors to be Selected
Inexperienced Authors
Inexperienced Authors Applying
Selected
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

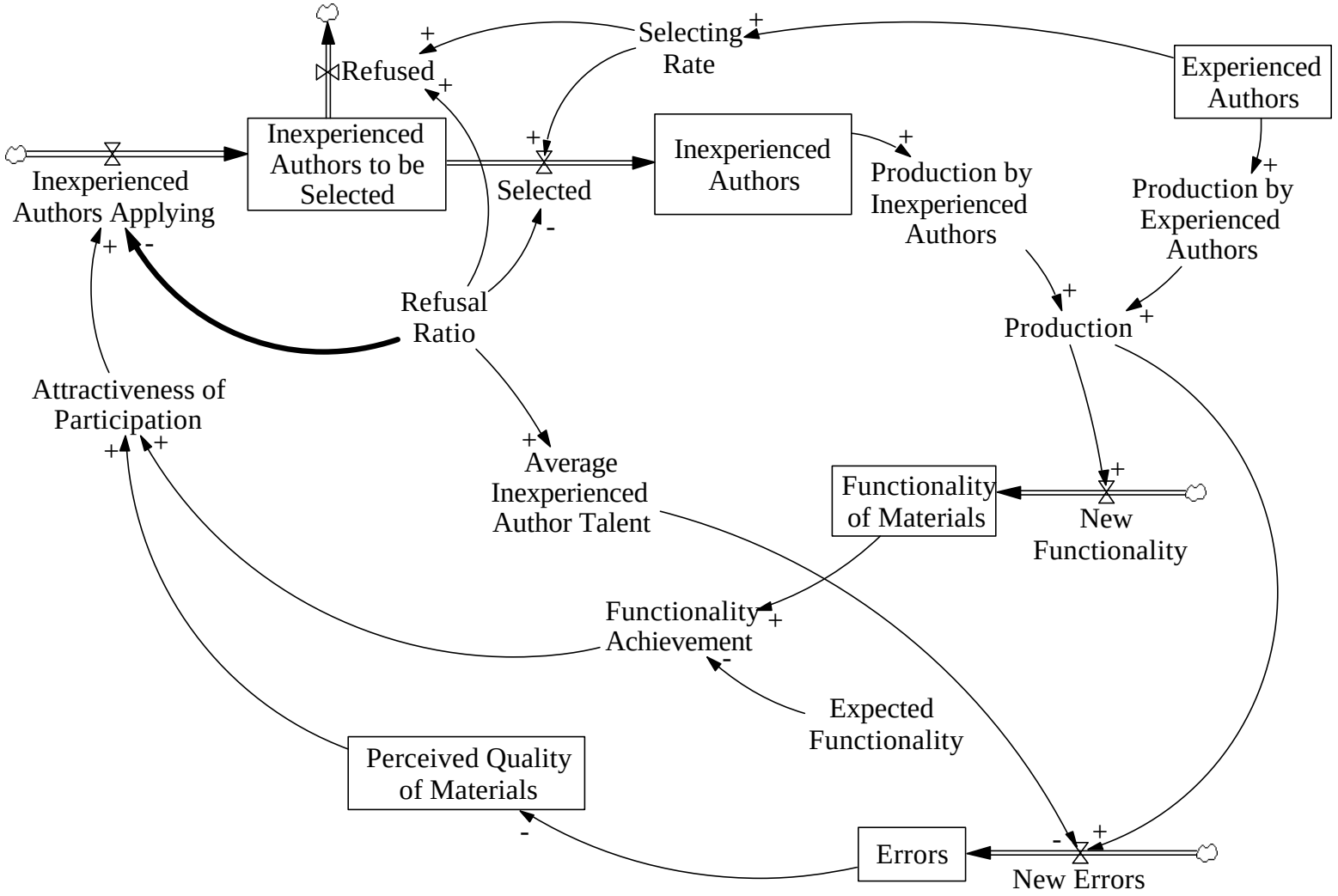
Sketch 3.3.


Refused
Selecting Rate
Experienced Authors
Inexperienced Authors to be Selected
Inexperienced Authors Applying
Selected
Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Refusal Ratio
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

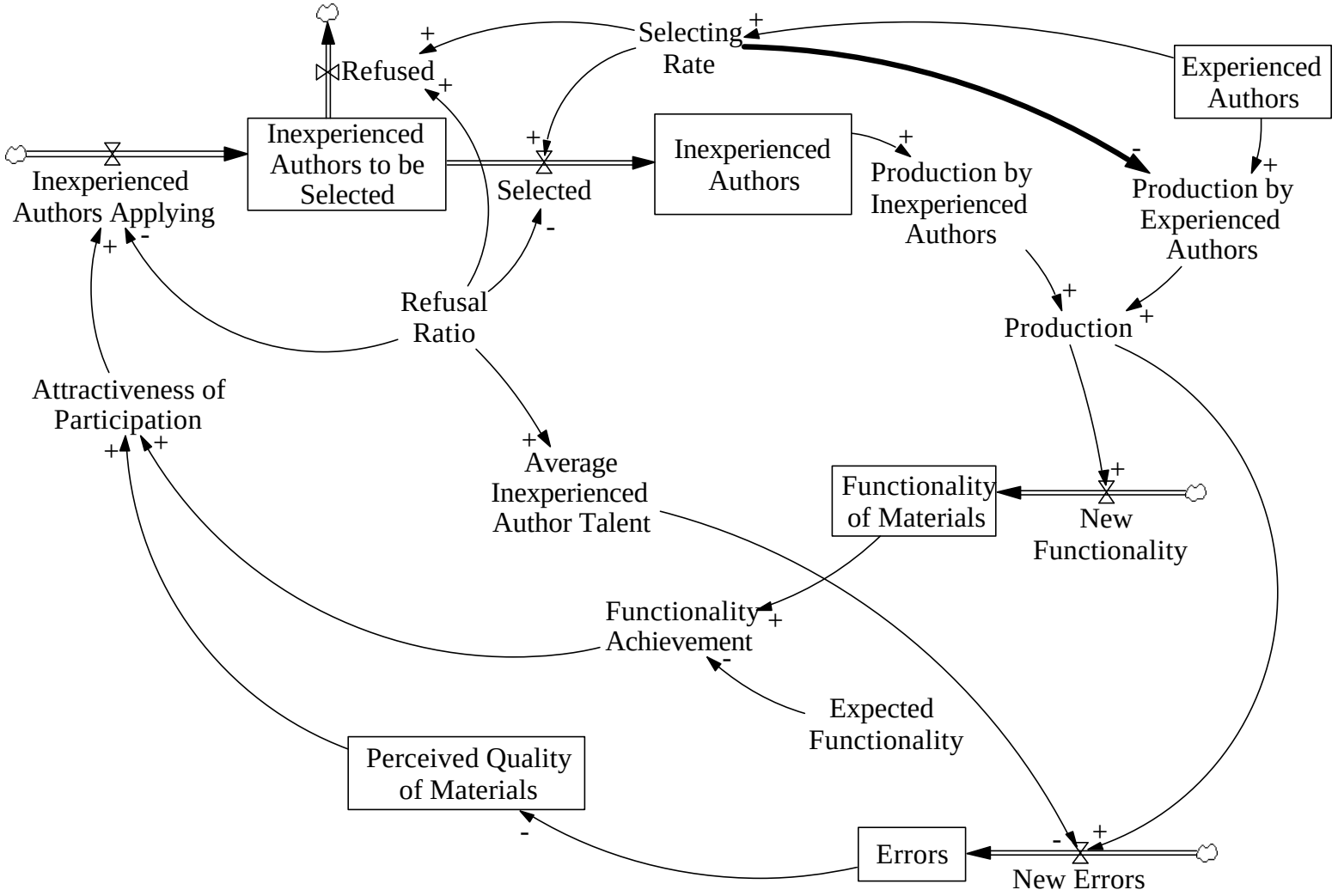
Sketch 3.4.


Refused
Selecting Rate
Experienced Authors
Inexperienced Authors to be Selected
Inexperienced Authors Applying
Selected
Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Refusal Ratio
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

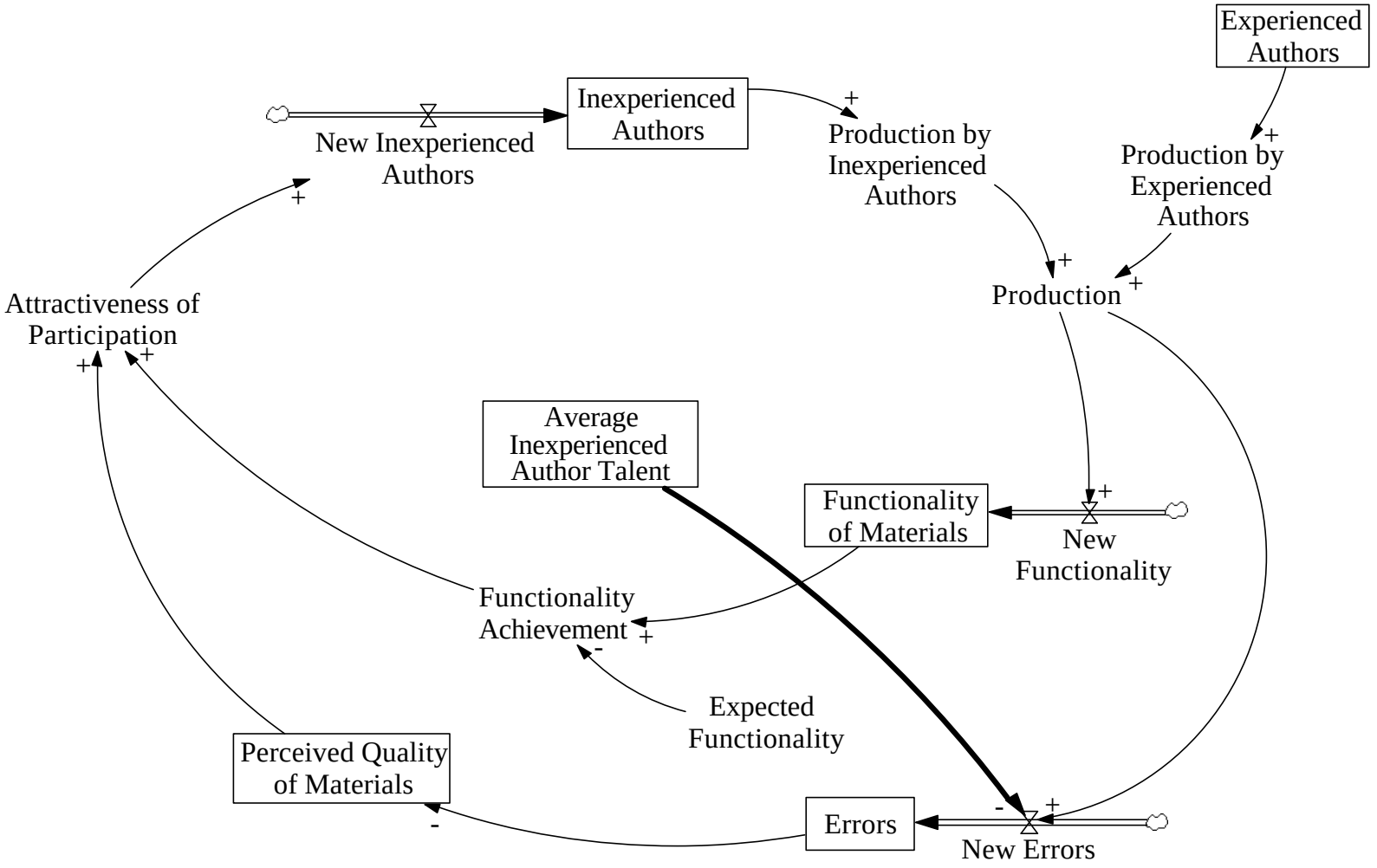
Sketch 3.5.


Refused
Selecting Rate
Experienced Authors
Inexperienced Authors to be Selected
Inexperienced Authors Applying
Selected
Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Refusal Ratio
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

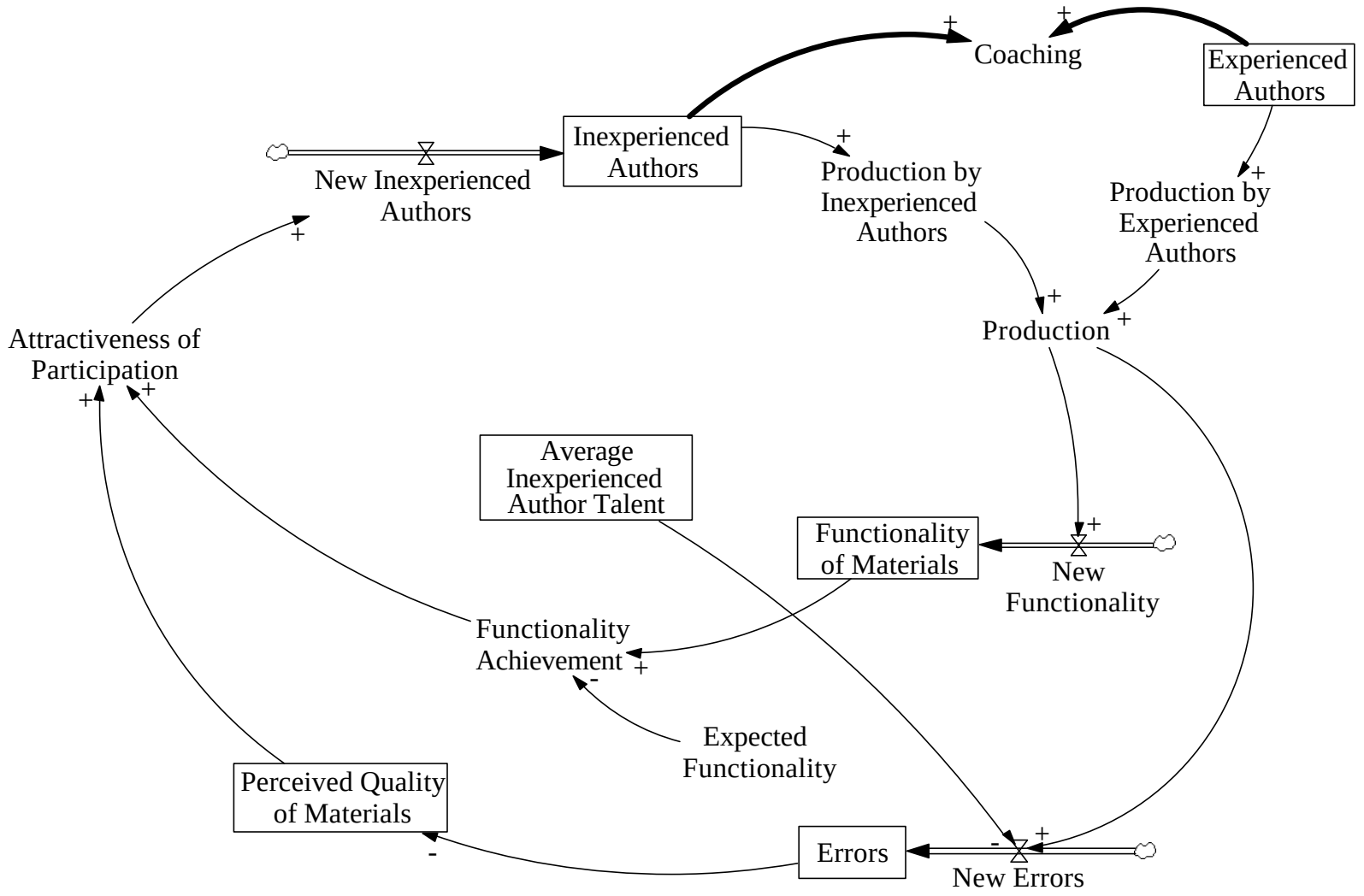
Sketch 3.6.


Refused
Selecting Rate
Experienced Authors
Inexperienced Authors to be Selected
Inexperienced Authors
Inexperienced Authors Applying
Selected
Production by Inexperienced Authors
Production by Experienced Authors
Refusal Ratio
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

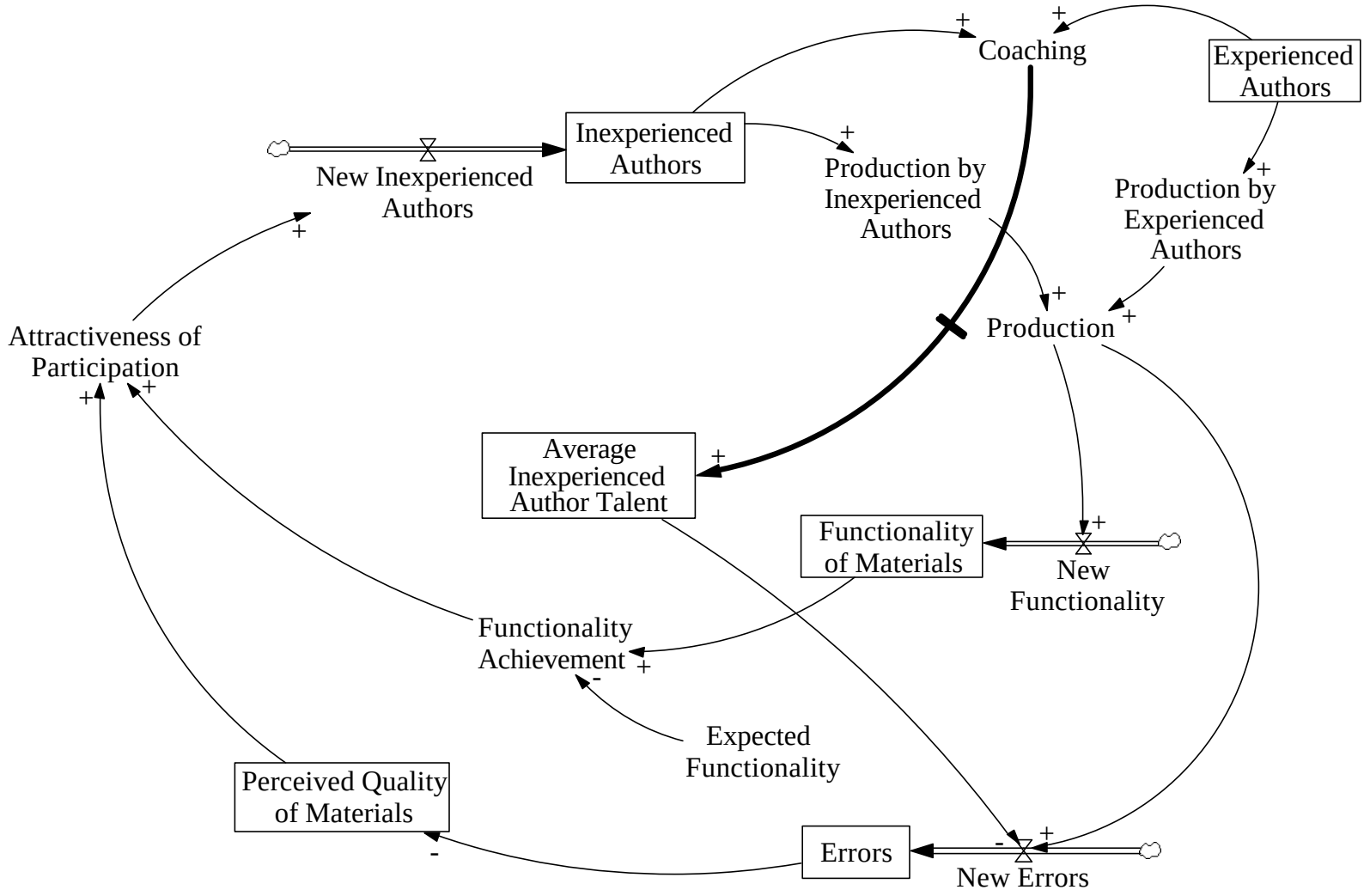
Sketch 4.1.


Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

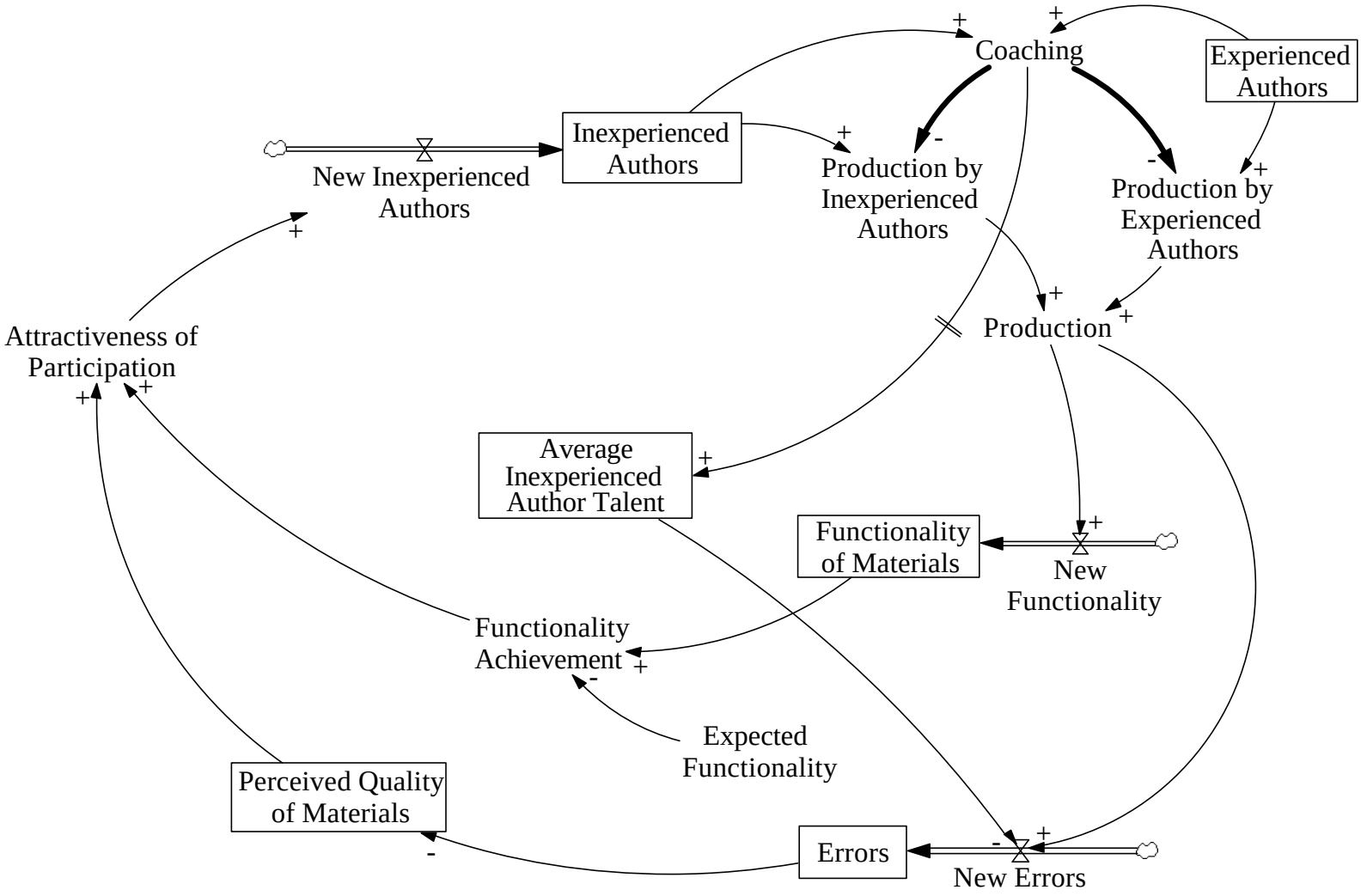
Sketch 4.2.


Coaching
Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

Sketch 4.3.


Coaching
Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

Sketch 4.4.


Coaching
Experienced Authors
Inexperienced Authors
New Inexperienced Authors
Production by Inexperienced Authors
Production by Experienced Authors
Production
Attractiveness of Participation
Average Inexperienced Author Talent
Functionality of Materials
New Functionality
Functionality Achievement
Expected Functionality
Perceived Quality of Materials
Errors
New Errors

A.7. Interview Protocol (Script)

Date: _____ Time Start: _____ Time End: _____

Interview Mode: [ ] Face to face [ ] Telephone [ ] E-mail

Respondent: _____ Title: _____

Affiliation: _____

Phone: _____ E-mail: _____

Interview Start: First of all, thank you for voluntarily accepting to do this interview, and for giving me your time. I am talking to you today to gather information about collaborative efforts to develop teaching materials for introducing system dynamics concepts in K through 12 education. As I mentioned in my e-mail to you, the information I gather today will be used as data in my dissertation in Information Science. Your name and any information that might identify you as an individual will not be used in the dissertation, or elsewhere; and no one else except myself and my dissertation committee will have access to the raw data without your written consent

Before we start, I would like to ask your permission to tape this phone conversation. Do you give your permission for me to tape this conversation? Also, I would like to ask your permission for quoting sections of this conversation anonymously. Do you give your permission for me to quote this conversation anonymously? *[Prompt if not] May I loosely paraphrase your replies?*

Checklist:

1. Intro _____
2. Permission to tape _____
3. Permission to quote _____

Questions:

Part I.

1. How did you get involved in the system dynamics K through 12 community?
2. What is your role in that community?

a. Have you ever worked with others within the community? *[Prompt if necessary] Have you ever worked with a mentor?*

3. How would you describe the efforts to develop and disseminate teaching materials within the system dynamics K through 12 community? *[Prompts, if necessary] How is it organized? Is it coordinated? [Prompt, if yes] How? [Prompt, if necessary] I am particularly interested in efforts for using the Internet for sharing work on developing teaching materials, such as the Creative Learning Exchange, Waters Foundation, CC-STADUS and MIT System Dynamics in Education Project websites.*

4. Do you see any categories that the individuals who contribute to these efforts would fall into? *[Prompt, if necessary] The efforts to develop and disseminate teaching materials within the community?*

[Prompts, if necessary]

a. *How would you categorize them based on expertise?*
b. *How would you categorize them based on contribution levels?*
c. *How would you categorize them based on reasons and motivation for participation?*

5. Is contributing to these efforts open to all? *[Prompt, if necessary] Can anybody who wants to make a contribution do so?*

a. *[Prompt if not] What are the requirements for participation?*

6. Now I would like to talk a little about the teaching material repositories on the Internet such as Creative Learning Exchange, Waters Foundation site, CC-STADUS, and MIT System Dynamics in Education Project. Is contributing to such online repositories open to all?

a. *[Prompt if not] What are the requirements for contributions?*

7. What do you think motivates people to make a contribution?

a. Do you think characteristics of the materials developed have any effect on the motivation of the contributors? *[Prompt if necessary] …characteristics such as quality, functionality, customizability, etc. [Prompt if necessary] Within this context, "materials" might mean documents such as handouts,*

assignment sheets, models, etc. For example I think of the teaching materials posted on the website such as Creative Learning Exchange.

b. Do you think community characteristics have any effect on the motivation of the contributors? *[Prompt if necessary] …such as number or talent level of contributors, number of users of the materials developed, ease of making contributions, probability of one's work being accepted and recognized?*

8. Do the contributors sometimes work together?

[Prompts, if they do]

a. *What is the nature of such collaboration?*

b. *Do they collaborate using the Internet, such as by e-mail or on-line chat?*

[Prompts, if they do]

i. *Do they ever meet face-to-face*

ii. *Do they generally begin to collaborate online before they meet, or vice-versa?*

c. *Do you have an estimate of what percentage of their contribution time the participants devote to collaboration?*

d. *Do you have an estimate of what percentage of collaboration takes place on-line versus face-to-face collaboration?*

e. *Does collaboration generally take place between peers, or between participants with different characteristics?*

i. *[Prompt if necessary] …such as experience level, contribution level, researchers vs. practitioners, etc.*

9. What do you think about the quality of the materials that are developed through these efforts?

a. How would you evaluate the overall quality on a scale of 0 to 10, 0 being the lowest and 10 the highest possible quality?

b. What about the variation of quality?

[Prompts if necessary]

i. *What is the quality level of the top 10%, and top 25% of the materials?*

ii. *What is the quality level of the bottom 10%, and bottom 25% of the materials?*

c. How would you group the materials based on their quality levels?

10. How does good work get recognized or selected?

a. How does work get judged as high or low quality?

b. Is there a filtering mechanism? How is it managed?

c. What happens to work judged to be high quality?

d. What happens to work judged to be low quality?

e. What happens to authors of work judged to be low quality?

11. Do you think anything can be done to improve the quality of the materials, in the short and the long run?

12. What can you say about the quantity of materials produced? [*How would you quantify the work produced? Number of documents, number of models, number of pages, etc.*]

a. Do you have a rough estimate of average work produced per contributor? *[Prompt if necessary] A very rough estimate is OK.*

b. Are there significant variations between the amounts of work produced by contributors with different characteristics?

i. *[Prompt if necessary] …such as experience level, collaboration level, researchers vs. practitioners, etc.*

c. Do you see the teaching material collections such as that of the Creative Learning Exchange as a coherent whole or a set of unconnected documents?

13. Who are the users of those teaching materials? What can you say about their characteristics?

a. What is your estimate of a user/contributor ratio?

14. In what ways do you think the users make use of those materials? *[Prompt if necessary] …such as self-study, classroom exercises, homework assignments.*

a. What features make those materials more or less useful?

15. What do you think makes those materials attractive to the users?

a. *[Prompt if necessary] …such as quality, functionality, ease of access, customizability, existence of other users, etc.*

Reference Modes

16. I would like you to draw some observed reference modes about the concepts (variables) we have discussed so far. Please use the sheet titled "Reference Modes - Observed". *[Prompt if necessary] Behaviors over time; graphs over time.* I would like you to pick the variables you think are key to the issue first.

[Prompt if necessary]

a. What about the number of contributors,…
b. number of users,…
c. number of materials produced,…
d. functionality of materials,…
e. quality of materials?

Policy Problems & Scenario

17. Are there large problems or issues within the community?

a. *[Prompt if there are problems] Do you think anything can be done about these problems in the short or the long run?*

18. What do you think the future holds for the system dynamics K through 12 community? *[Will the community grow? Decline? Split? Divide?]*

a. What is your "best case" scenario?
b. What is your "worst case" scenario?
c. What is your "most likely" scenario?

19. At this stage, I would like you to draw some more reference modes. This time let's focus on projected reference modes, which you think may happen in the future. Please use the sheet titled "Reference Modes - Projected".

[Prompt if necessary]

a. What about the number of contributors,…
b. number of users,…
c. number of materials produced,…
d. functionality of materials,…
e. quality of materials?

Part II,

20. In the previous phases of this study I built a system dynamics model of a hypothetical open online collaboration community. I would like to show you some sketches from that model. I will explain the variables and loops in the sketches, and then ask you whether they apply to the case of system dynamics K through 12 community.

a. Please look at Sketch 0.1. Here, participating authors produce content in the form of documents, models, visuals, etc. and thus add new functionality to the teaching materials collection. Here, functionality means a general level of usefulness of the materials for teaching purposes. As new functionality is added, functionality of the materials approaches the level expected by possible users, and thus functionality achievement increases. Increased functionality achievement increases the attractiveness of participation for authors, and thus new authors become active in the community faster. Do you think such a positive loop reinforces the growth of the number of authors, and the level of functionality of the materials in the case of this community?

b. Sketch 0.2 shows that a higher level of functionality achievement attracts more users. In Sketch 0.3 a higher number of users increases the attractiveness of participation for the authors, thus attracting more new authors. Do you think such a positive loop reinforces the growth of the number of authors, the level of functionality of the materials, and the number of users in the case of this community?

c. Do you see any other influence that might reinforce the growth of the number of authors, the level of functionality of the materials, and the number of users in the case of this community?

d. Please look at Sketch 0.4. Here as the materials approach the expected level of functionality, opportunities for contribution decrease. Due to decreased opportunity, a smaller number of new authors are attracted to participate. Do you think such a negative loop limits the growth of the number of authors, and the level of functionality of the materials in the

case of this community at the time being? *[Prompt if not] Do you think there is a probability that such a negative loop may limit growth in the future?*

e. Please look at Sketch 0.5. Here, as authors produce content and add functionality to the materials, they also generate errors or weaknesses in the materials. In Sketch 0.6 the number of errors decrease the perceived quality of the materials. This is represented as a "smooth", since the perception of quality would change gradually (with a delay). A decreased perception of quality decreases the attractiveness of participation for the authors, thus forming another negative loop. Do you think such a negative loop, which runs through errors and weaknesses, limits the growth of the number of authors, and the level of functionality of the materials in the case of this community? *[Prompt if not] Do you think there is a probability that such a negative loop may limit growth in the future?*

f. Sketch 0.7 shows that a decreased Perceived Quality of Materials has a decreasing effect on the number of new users, thus forming another negative feedback loop. Do you think such a negative loop limits the growth of the number of authors, the level of functionality of the materials, and the number of users in the case of this community? *[Prompt if not] Do you think there is a probability that such a negative loop may limit growth in the future?*

g. Do you see any other influence that might limit the growth in this community?

21. As you can see, when conceptualizing the model, I laid out the main problem as the dichotomy between building functionality and maintaining quality; or put in another way, producing materials vs. improving materials. Many times these act against each other. When you try to build functionality too fast, you may hurt quality. On the other hand, trying to increase quality above a certain level may bring about too slow a functionality growth. Do you observe such a problem in the case of this community?

22. Having laid out the problem about building functionality while maintaining quality, I tried to sketch some policy options. I would like to show you these sketches and ask you whether any of these processes have been implemented, or at least suggested as a remedy for the problems of this community? Before moving on to the policy options, please look at Sketch 1.0, where I divided the authors into two groups, experienced authors and inexperienced authors. These two groups add functionality to the materials by producing content, and while doing that they generate new errors or weaknesses in materials. Now the policy options…

23. The first policy option is filtering materials that are produced by inexperienced authors. This option is based on the premises that inexperienced authors generate more errors per production, and by filtering the materials that are produced by inexperienced authors, it may be possible to decrease the number of new errors or weaknesses in materials. In Sketch 1.1, materials produced by inexperienced authors are not added directly to the overall materials produced, but instead diverted to a backlog to be filtered. As Sketch 1.2 shows, a certain portion of this backlog would be accepted and added to the overall production, while the rest is rejected. Sketch 1.3 shows that filtering would be done by experienced developers, with a certain filtering rate per time unit, and an average rejection ratio would determine the amount of materials that are accepted or rejected. The rejection ratio would depend on the level of scrutiny experienced developers apply during filtering, and thus decrease the number of new errors that go into the materials collection. As Sketch 1.4 shows, a higher rejection ratio, which means a higher scrutiny level, would reduce the number of new errors. As portrayed in Sketch 1.5 a possible adverse effect of this policy would be decreasing motivation for production on the part of the inexperienced authors. It is possible that as the rejection rate increases, motivation for producing materials would decrease. Sketch 1.6 shows another adverse effect of this policy: Materials produced by experienced authors would decrease, since they would dedicate a portion of their time to filtering.

a. Have you observed such processes operating in the case of this community? *[Prompt if yes] What were the consequences of these filtering processes? [Prompt if not] Has such processes ever been suggested? [Prompt if not] Do you think such processes would remedy certain problems in this community? [Prompt if yes] What do you think the consequences of such a filtering approach would be?*

24. The second policy option is reviewing and editing content in order to fix existing errors. Please look at Sketch 2.1. Here again, experienced authors and inexperienced authors build functionality by producing materials and while doing that they generate errors and weaknesses in materials. Sketch 2.2 shows that experienced authors would spend time on reviewing and editing content and thus fix a portion of existing errors. As Sketch 1.3 shows, reviewing and editing would decrease production by experienced authors. This decrease would probably be greater than that would happen under the filtering option, since reviewing and editing existing content would take more time than filtering new production.

a. Have you observed such processes operating in the case of this community? *[Prompt if yes] What were the consequences of these reviewing and editing processes? [Prompt if not] Has such processes ever been suggested? [Prompt if not] Do you think such processes would remedy certain problems in this community? [Prompt if yes] What do you think the consequences of such a reviewing and editing approach would be?*

25. Before moving on to the third policy option, I would like to introduce another concept, namely the average talent level of the inexperienced authors. Please look at Sketch 3.1. I suggest that the number of errors generated by inexperienced authors would depend on their talent level. The higher the average inexperienced author talent, the fewer new errors generated by the inexperienced authors.

Based on this talent concept, the third policy option is selecting new inexperienced authors according to their talents. Please look at Sketch 3.2. Here, new inexperienced authors are not directly accepted into the existing inexperienced authors pool. Rather, they apply and wait to be selected. As Sketch

3.3 shows, selecting is carried out by experienced authors, and of course some applicants are refused, based on an average refusal ratio. Refusal ratio would be based on the scrutiny level of the selection process. A higher level of scrutiny would mean a higher refusal rate, and that in turn would mean a higher average inexperienced author talent level, as Sketch 3.4 shows. One possible adverse effect here would be a decrease in the number of inexperienced authors applying. As shown in Sketch 3.5, I suggest that as refusal ratio increases, number of inexperienced authors applying would decrease. Sketch 3.6 shows another adverse effect of this policy: Materials produced by experienced authors would decrease, since they would dedicate a portion of their time to selecting.

a. Have you observed such processes operating in the case of this community? *[Prompt if yes] What were the consequences of these selecting processes? [Prompt if not] Has such processes ever been suggested? [Prompt if not] Do you think such processes would remedy certain problems in this community? [Prompt if yes] What do you think the consequences of such a selecting approach would be?*

26. The fourth policy option is also geared toward increasing the average inexperienced author talent level. However, this time not by selecting the incoming inexperienced authors, but coaching the existing inexperienced authors. Please look at Sketch 4.2. Here experienced authors coach inexperienced authors, and as Sketch 4.3 shows, coaching increases average inexperienced author talent gradually over time (with a delay). Accordingly, average inexperienced author talent is defined as a "smooth" in this context. Both experienced and inexperienced authors would dedicate a portion of their time to coaching under this policy. So, Sketch 4.4 shows that coaching would decrease materials produced by experienced and inexperienced authors, thus affecting the functionality growth negatively in the short run.

a. Have you observed such processes operating in the case of this community? *[Prompt if yes] What were the consequences of these coaching processes? [Prompt if not] Has such processes ever been suggested? [Prompt if not] Do you think such processes would remedy*

certain problems in this community? [Prompt if yes] What do you think the consequences of such a coaching approach would be?

27. At this stage, I would like you to compare these four policy options in the context of system dynamics K through 12 community.

a. Which of these four policy options do you think would be beneficial in the case of this community?

b. Which of these four policy options do you think could be implemented?

Part III.

28. Is there anything you would like to add that might help me get a better understanding of the system dynamics K through 12 community?

29. Is there anything you are surprised I have not brought up about the community?

30. Who else would you recommend I talk to about these issues?

31. What else do you think I should be asking during the interviews to get a better understanding of these issues?

APPENDIX B -- OPEN SOURCE SOFTWARE DEVELOPMENT MODEL (ITERATION V VERSION) EQUATIONS AND SECTOR VIEWS[7]

B.1. Model Equations (Iteration V Version)

(001) Acceptable Level of Known Bugs per Functionality = 0.1 [*Units*: bugs/UF]

(002) Acceptable Level of Total Bugs per Functionality = 0.3 [*Units*: bugs/UF]

(003) Accepted Production = Production to be Filtered * Filtering Rate * (1 - Rejection Ratio) [*Units*: lines/Month]

(004) Achieved Functionality Ratio = Product Functionality / Limit on Product Functionality [*Units*: Dmnl]

(005) "Achieved/Expected Functionality Ratio" = Achieved Functionality Ratio / Expected Functionality Ratio [*Units*: Dmnl]

(006) Attractiveness of Product for Developers Due to Achieved Functionality = f Attractiveness for Developers vs Achieved Functionality ("Operative/Expected Functionality Ratio") [*Units*: Dmnl]

(007) Attractiveness of Product for Developers Due to Potential Functionality = f Attractiveness for Developers vs Potential Functionality (Achieved Functionality Ratio) [*Units*: Dmnl]

(008) Attractiveness of Product for Developers Due to Users = f Attractiveness for Developers vs Success in Attracting Users (Success in Attracting Users) [*Units*: Dmnl]

(009) Attractiveness of Product for Users = f Attractiveness for Users vs Achieved Functionality (Achieved Functionality Ratio) [*Units*: Dmnl]

(010) Average Developer Participation = f Average Developer Participation vs Rejection Ratio (Rejection Ratio) * Average Developer Participation Normal [*Units*: hours/(Month*people)]

(011) Average Developer Participation Normal = 30 [*Units*: hours/(Month*people)]

(012) Average Developer Productivity = Average Developer Productivity Normal * f Average Developer Productivity vs Participant Population Intensity (Participant Population Intensity) [*Units*: lines/hour]

(013) Average Developer Productivity Normal = 5 [*Units*: lines/hour]

(014) Average Developer Talent = IF THEN ELSE (Developers = 0, 0, (Developer Talent Pool / Developers)) [*Units*: RTU/people]

[7] The model file for the Iteration V version of the OSSD model, as well as the prior versions (Iteration I through Iteration IV) can be downloaded from http://www,glue,umd,edu/~diker. The page also contains a link to download a loyalty free personal version of Vensim, the system dynamics modeling and simulation package, which can be used to view and simulate the model.

(015) Average Developer Talent Building Opportunity = Maximum Developer Talent - Average Developer Talent [*Units*: RTU/people]

(016) Average Developer Talent Building Ratio = f Average Developer Talent Building Ratio vs Coaching Hours Coverage (Coaching Hours Coverage) * Maximum Developer Talent Building Ratio [*Units*: 1/Month]

(017) Average Developer Talent Built = Average Developer Talent Building Opportunity * Average Developer Talent Building Ratio [*Units*: RTU/(Month*people)]

(018) Average Incoming Developer Talent = f Average Incoming Developer Talent vs Refusal Ratio (Refusal Ratio) [*Units*: RTU/people]

(019) Average Leader Participation = 30 [*Units*: hours/(Month*people)]

(020) Average Leader Productivity = Average Leader Productivity Normal * f Average Leader Productivity vs Participant Population Intensity (Participant Population Intensity) [*Units*: lines/hour]

(021) Average Leader Productivity Normal = 10 [*Units*: lines/hour]

(022) Average Leader Talent = 1 [*Units*: RTU/people]

(023) Average Relative Developer Talent = Average Developer Talent / Maximum Developer Talent [*Units*: Dmnl]

(024) Average Relative Leader Talent = Average Leader Talent / Maximum Developer Talent [*Units*: Dmnl]

(025) Bug Discovery Rate Normal = 3 [*Units*: bugs/hour]

(026) Bug Fixing Quality = f Debugging Quality vs Average Relative Developer Talent (Average Relative Developer Talent) [*Units*: Dmnl]

(027) Bug Fixing Rate Normal = 1 [*Units*: bugs/hour]

(028) Bug Generating Rate Normal = 0.01 [*Units*: bugs/line]

(029) Bugs Added per Bug Fixed = f Bugs Added per Bug Fixed vs Debugging Quality (Bug Fixing Quality) * Bugs Added per Bug Fixed Normal [*Units*: Dmnl]

(030) Bugs Added per Bug Fixed Normal = 0.075 [*Units*: Dmnl]

(031) Bugs Fixed = (Developer Bug Fixing Rate * Developer Hours Allocated to Bug Fixing) + (Leader Bug Fixing Rate * Leader Hours Allocated to Bug Fixing) [*Units*: bugs/Month]

(032) Bugs Found = (Developer Bug Discovery Rate * Developer Hours Allocated to Bug Detection) + (Leader Bug Discovery Rate * Leader Hours Allocated to Bug Detection) [*Units*: bugs/Month]

(033) Bugs in Accepted Code = Accepted Production * Bugs per Code in Production to be Filtered * (1 - Quality Improvement by Filtering) [*Units*: bugs/Month]

(034) Bugs in Production to be Filtered = INTEG(New Bugs in Production to be Filtered - Bugs in Accepted Code - Bugs in Rejected Code , Initial Bugs in Production to be Filtered) [*Units*: bugs]

(035) Bugs in Rejected Code = Rejected Production * Bugs per Code in Production to be Filtered * (1 + Quality Improvement by Filtering) [*Units*: bugs/Month]

(036) Bugs per Code = Total Bugs in Code / Project Size [*Units*: bugs/line]

(037) Bugs per Code in Production to be Filtered = ACTIVE INITIAL(ZIDZ (Bugs in Production to be Filtered , Production to be Filtered) , 0.0064) [*Units*: bugs/line]

(038) Candidates Applying = Overall Attractiveness of Product for Developers * Potential Developers / Normal Time to Attract All Potential Developers [*Units*: people/Month]

(039) Candidates Refused = Selecting Rate * Refusal Ratio * Developer Candidates [*Units*: people/Month]

(040) Candidates Selected as New Developers = Selecting Rate * (1 - Refusal Ratio) * Developer Candidates [*Units*: people/Month]

(041) Coaching Hours Availability Ratio = ZIDZ (Total Coaching Hours Available , Developer Hours Needed for Coaching) [*Units*: Dmnl]

(042) Coaching Hours Coverage = Coaching Hours per Developer / Maximum Coaching Hours Needed per Developer [*Units*: Dmnl]

(043) Coaching Hours Needed per Developer = Pressure for Talent Building * Maximum Coaching Hours Needed per Developer [*Units*: hours/(Month*people)]

(044) Coaching Hours per Developer = ZIDZ (Developer Hours Allocated to Coaching , Developers) [*Units*: hours/(Month*people)]

(045) Code Added per Bug Fixed = ZIDZ (f Code Added per Bug Fixed vs Debugging Quality (Bug Fixing Quality) , Bugs per Code) [*Units*: lines/bug]

(046) Desired Time to Discover All Bugs = 6 [*Units*: Month]

(047) Desired Time to Fix All Known Bugs = 6 [*Units*: Month]

(048) Developer Bug Discovery Rate = Bug Discovery Rate Normal * f Bug Discovery Rate vs Average Relative Developer Talent (Average Relative Developer Talent) * f Bug Discovery Efficiency vs Unknown Bugs Density (Unknown Bug Density) [*Units*: bugs/hour]

(049) Developer Bug Fixing Rate = Bug Fixing Rate Normal * f Bug Fixing Rate vs Average Relative Developer Talent (Average Relative Developer Talent) [*Units*: bugs/hour]

(050) Developer Bug Generating Rate = Bug Generating Rate Normal * f Bug Generating Rate vs Average Relative Talent (Average Relative Developer Talent) [*Units*: bugs/line]

(051) Developer Candidates = INTEG(Candidates Applying - Candidates Refused - Candidates Selected as New Developers , 0) [*Units*: people]

(052) Developer Hours Allocated to Bug Detection = Developer Hours Revised Allocation Factor * Developer Hours Needed for Bug Detection [*Units*: hours/Month]

(053) Developer Hours Allocated to Bug Fixing = Developer Hours Revised Allocation Factor * Developer Hours Needed for Bug Fixing [*Units*: hours/Month]

(054) Developer Hours Allocated to Coaching = Developer Hours Revised Allocation Factor * Developer Hours Planned for Coaching [*Units*: hours/Month]

(055) Developer Hours Allocated to Production = Total Developer Hours Available - "Total Developer Hours Allocated for Non-Production Tasks" [*Units*: hours/Month]

(056) Developer Hours Allocation Factor = f Developer Hours Allocation Factor vs Developer Hours Coverage Ratio (Developer Hours Coverage Ratio) [*Units*: Dmnl]

(057) Developer Hours Coverage Ratio = ZIDZ (Total Developer Hours Available , Total Developer Hours Needed) [*Units*: Dmnl]

(058) Developer Hours for Bug Detection Gap = Developer Hours Needed for Bug Detection - Developer Hours Allocated to Bug Detection [*Units*: hours/Month]

(059) Developer Hours for Bug Fixing Gap = Developer Hours Needed for Bug Fixing - Developer Hours Allocated to Bug Fixing [*Units*: hours/Month]

(060) Developer Hours for Production Gap = Developer Hours Planned for Production - Developer Hours Allocated to Production [*Units*: hours/Month]

(061) Developer Hours Needed for Bug Detection = (Pressure for Bug Detection * ZIDZ (Unknown Bugs in Code , Developer Bug Discovery Rate)) / Desired Time to Discover All Bugs [*Units*: hours/Month]

(062) Developer Hours Needed for Bug Fixing = (Pressure for Bug Fixing * ZIDZ (Known Bugs in Code , Developer Bug Fixing Rate)) / Desired Time to Fix All Known Bugs [*Units*: hours/Month]

(063) Developer Hours Needed for Coaching = MIN ((Coaching Hours Needed per Developer * Developers) , Total Developer Hours Available) [*Units*: hours/Month]

(064) Developer Hours Planned for Coaching = f Developer Hours Planned for Coaching vs Coaching Hours Availability Ratio (Coaching Hours Availability Ratio) * Developer Hours Needed for Coaching [*Units*: hours/Month]

(065) Developer Hours Planned for Production = Total Developer Hours Available [*Units*: hours/Month]

(066) Developer Hours Revised Allocation Factor = INTEG(Developer Hours Revised Allocation Factor Adjustment , Initial Developer Hours Revised Allocation Factor) [*Units*: Dmnl]

(067) Developer Hours Revised Allocation Factor Adjustment = Developer Hours Revised Allocation Factor Adjustment Discrepancy / Developer Hours Revised Allocation Factor Adjustment Time [*Units*: 1/Month]

(068) Developer Hours Revised Allocation Factor Adjustment Discrepancy = Indicated Developer Hours Revised Allocation Factor - Developer Hours Revised Allocation Factor [*Units*: Dmnl]

(069) Developer Hours Revised Allocation Factor Adjustment Time = 1 [*Units*: Month]

(070) Developer Talent Built = Average Developer Talent Built * Developers [*Units*: RTU/Month]

(071) Developer Talent Gained = Average Incoming Developer Talent * Candidates Selected as New Developers [*Units*: RTU/Month]

(072) Developer Talent Lost = Average Developer Talent * Leaving Developers [*Units*: RTU/Month]

(073) Developer Talent Pool = INTEG(Developer Talent Gained - Developer Talent Lost + Developer Talent Built , Initial Developer Talent Pool) [*Units*: RTU]

(074) Developers = INTEG(Candidates Selected as New Developers - Leaving Developers , Initial Developers) [*Units*: people]

(075) Developers on Other Projects = INTEG(Potential Developers Choosing Other Projects - Leaving Developers from Other Projects , Initial Developers on Other Projects) [*Units*: people]

(076) Expected Funtionality Ratio = f Expected Functionality Ratio vs Patience (Patience) [*Units*: Dmnl]

(077) f Attractiveness for Developers vs Achieved Functionality ([(0,0)-(6,1)],(0,0), (0.165138,0), (0.311927,0.0307018), (0.40367,0.114035), (0.862385,0.758772), (1.04587,0.96), (1.3211,0.986842), (1.54128,0.991228), (1.88379,0.995614), (2.39755,0.995614), (2.88073,0.995614), (3.2844,0.995614), (4,1), (5,1)) [*Units*: Dmnl]

(078) f Attractiveness for Developers vs Potential Functionality ([(0,0)-(1,1)],(0,1), (0.125,0.994737), (0.25,0.987719), (0.3333,0.980263), (0.425,0.972807), (0.5,0.959211), (0.575,0.890351), (0.6666,0.723684), (0.761468,0.486842), (0.810398,0.328947), (0.862385,0.184211), (0.905199,0.0877193), (0.941896,0.0394737), (0.97,0), (1,0)) [*Units*: Dmnl]

(079) f Attractiveness for Developers vs Success in Attracting Users ([(0,0)-(1,4)],(0,1), (0.125382,1.07018), (0.24159,1.22807), (0.348624,1.47368), (0.428135,1.82456), (0.5,2.31579), (0.575,2.96491), (0.6666,3.5614), (0.75,3.78947), (0.875,3.92982), (1,3.96491)) [*Units*: Dmnl]

(080) f Attractiveness for Users vs Achieved Functionality ([(0,0)-(1,1)], (0,0), (0.131498,0.00877193), (0.253823,0.0263158), (0.357798,0.0701754), (0.477064,0.131579), (0.574924,0.241228), (0.651376,0.390351), (0.706422,0.557018), (0.749235,0.697368), (0.807339,0.837719), (0.892966,0.95614), (1,1)) [*Units*: Dmnl]

(081) f Average Developer Participation vs Rejection Ratio ([(0,0)-(1,1.4)],(0,1.3333), (0.1,1.1667), (0.2,1), (0.35,0.8), (0.5,0.65), (0.6,0.5667), (0.75,0.5), (0.85,0.4333), (0.9,0.36667), (0.95,0.2667), (1,0)) [*Units*: Dmnl]

(082) f Average Developer Productivity vs Participant Population Intensity ([(0,0)-(3,1)],(0,1), (0.2,1), (0.4,0.99), (0.62,0.95), (0.95,0.85), (1.5,0.6), (2,0.4), (2.3,0.29), (2.5,0.22), (3,0.2)) [*Units*: Dmnl]

(083) f Average Developer Talent Building Ratio vs Coaching Hours Coverage ([(0,0)-(1,1)],(0,0), (0.1,0.02), (0.2,0.08), (0.3,0.19), (0.4,0.36), (0.5,0.6), (0.6,0.78), (0.7,0.87), (0.8,0.94), (0.9,0.98), (1,1)) [*Units*: Dmnl]

(084) f Average Incoming Developer Talent vs Refusal Ratio ([(0,0)-(1,1)], (0,0.3), (0.045,0.4), (0.1,0.5), (0.2,0.65), (0.3,0.75), (0.45,0.85), (0.6,0.9), (0.8,0.95), (1,1)) [*Units*: RTU/people]

(085) f Average Leader Productivity vs Participant Population Intensity ([(0,0)-(3,1)],(0,1), (0.2,1), (0.4,0.99), (0.62,0.95), (0.95,0.85), (1.5,0.6), (2,0.4), (2.3,0.29), (2.5,0.22), (3,0.2)) [*Units*: Dmnl]

(086) f Bug Discovery Efficiency vs Unknown Bugs Density ([(0,0)-(1,1)],(0,0), (0.1,0.2), (0.2,0.37), (0.3,0.52), (0.4,0.65), (0.5,0.77), (0.6,0.85), (0.7,0.91), (0.8,0.96), (0.9,0.98), (1,1), (2,1), (5,1)) [*Units*: Dmnl]

(087) f Bug Discovery Rate vs Average Relative Developer Talent ([(0,0)-(1,1)],(0,0), (0.1,0.02), (0.2,0.05), (0.3,0.15), (0.4,0.3), (0.5,0.5), (0.6,0.7), (0.7,0.85), (0.8,0.95), (0.9,0.98), (1,1)) [*Units*: Dmnl]

(088) f Bug Fixing Rate vs Average Relative Developer Talent ([(0,0)-(1,1)],(0,0), (0.1,0.02), (0.2,0.05), (0.3,0.15), (0.4,0.3), (0.5,0.5), (0.6,0.7), (0.7,0.85), (0.8,0.95), (0.9,0.98), (1,1)) [*Units*: Dmnl]

(089) f Bug Generating Rate vs Average Relative Talent ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.89), (0.4,0.78), (0.5,0.65), (0.6,0.48), (0.7,0.28), (0.8,0.14), (0.9,0.07), (1,0.05)) [*Units*: Dmnl]

(090) f Bugs Added per Bug Fixed vs Debugging Quality ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.85), (0.4,0.7), (0.5,0.5), (0.6,0.3), (0.7,0.15), (0.8,0.05), (0.9,0.02), (1,0)) [*Units*: Dmnl]

(091) f Code Added per Bug Fixed vs Debugging Quality ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.85), (0.4,0.7), (0.5,0.5), (0.6,0.3), (0.7,0.15), (0.8,0.05), (0.9,0.02), (1,0)) [*Units*: Dmnl]

(092) f Debugging Quality vs Average Relative Developer Talent ([(0,0)-(1,1)],(0,0), (0.1,0.03), (0.2,0.1), (0.3,0.2), (0.4,0.36), (0.5,0.58), (0.6,0.8), (0.7,0.94), (0.8,0.98), (0.9,0.995), (1,1)) [*Units*: Dmnl]

(093) f Developer Hours Allocation Factor vs Developer Hours Coverage Ratio ([(0,0)-(1,1)],(0,0), (0.01,0), (0.25,0.24), (0.5,0.49), (0.75,0.74), (1,0.99), (20,0.99)) [*Units*: Dmnl]

(094) f Developer Hours Planned for Coaching vs Coaching Hours Availability Ratio ([(0,0)-(1,1)],(0,0), (1,1), (20,1)) [*Units*: Dmnl]

(095) f Developer Hours Revised Allocation Factor vs Pressure for Production ([(0,0)-(1,1)],(0,1), (0.1,0.97), (0.2,0.9), (0.3,0.767544), (0.4,0.6), (0.5,0.416667), (0.6,0.280702), (0.7,0.2), (0.8,0.15), (0.9,0.12), (1,0.1)) [*Units*: Dmnl]

(096) f Expected Functionality Ratio vs Patience ([(0,0)-(1,1)],(0,1), (0.1,0.97), (0.2,0.92), (0.3,0.833333), (0.4,0.7), (0.5,0.5), (0.6,0.3), (0.7,0.166667), (0.8,0.08), (0.9,0.03), (1,0.0001)) [*Units*: Dmnl]

(097) f Functionality Adding Efficiency vs Achieved Ratio ([(0,0)-(1,1)],(0,1), (0.140673,0.99), (0.25,0.98), (0.38,0.9693), (0.5,0.9561), (0.65,0.9386), (0.75,0.9211), (0.85,0.8947), (0.9,0.8553), (0.929664,0.7763), (0.95,0.5877), (1,0)) [*Units*: Dmnl]

(098) f Functionality Lost per Bug Fixed vs Debugging Quality ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.85), (0.4,0.7), (0.5,0.5), (0.6,0.3), (0.7,0.15), (0.8,0.05), (0.9,0.02), (1,0)) [*Units*: Dmnl]

(099) f Initial Developer Hours Revised Allocation Factor vs initial Average Developer Participation ([(0,0)-(8,0.6)],(0,0), (7,0), (8,0.6), (300,0.6), (1000,0.6)) [*Units*: Dmnl]

(100) f Initial Leader Hours Revised Allocation Factor vs initial Leader Hours Coverage Ratio ([(0,0)-(1,0.6)],(0,0), (0.01,0), (0.015,0.6), (1,0.6), (11,0.6), (300,0.6), (1000,0.6)) [*Units*: Dmnl]

(101) f Leader Hours Allocation Factor vs Leader Hours Coverage Ratio ([(0,0)-(1,1)],(0,0), (1,1), (20,1)) [*Units*: Dmnl]

(102) f Leader Hours Planned for Coaching vs Leader Hours Availability for Coaching ([(0,0)-(100,1)],(0,0), (0.001,1), (1,1), (2,0.5), (2.5,0.4), (4,0.25), (5,0.2), (8,0.125), (10,0.1), (12.5,0.08), (20,0.05), (40,0.025), (100,0.01)) [*Units*: Dmnl]

(103) f Leader Hours Revised Allocation Factor vs Pressure for Production ([(0,0)-(1,1)],(0,1), (0.1,0.97), (0.2,0.9), (0.3,0.767544), (0.4,0.6), (0.5,0.416667), (0.6,0.280702), (0.7,0.2), (0.8,0.15), (0.9,0.12), (1,0.1)) [*Units*: Dmnl]

(104) f Leaving Accelaration vs Achieved Functionality ([(0,0)-(1,10)],(0,10), (0.088685,9.76316), (0.159021,9.36842), (0.214067,8.85526), (0.262997,8.22368), (0.308868,7.5), (0.342508,6.7), (0.379205,5.53947), (0.40367,4.55263), (0.428135,3.48684), (0.470948,2.53947), (0.510703,1.78947), (0.562691,1.23684), (0.657492,1.01754), (0.75,1), (1,1), (5,1)) [*Units*: Dmnl]

(105) f Leaving Accelaration vs Perceived Product Quality ([(0,0)-(1,10)], (0,10), (0.088685,9.76316), (0.159021,9.36842), (0.214067,8.85526), (0.262997,8.22368), (0.308868,7.5), (0.342508,6.7), (0.379205,5.78947), (0.425076,4.60526), (0.470948,3.90351), (0.510703,3.20175), (0.562691,2.58772), (0.657492,1.97368), (0.75,1.5), (1,1)) [*Units*: Dmnl]

(106) f Leaving Accelaration vs Potential Functionality ([(0,0)-(1,20)],(0,1), (0.15,1.1), (0.3,1.22), (0.45,1.35), (0.6,1.6), (0.75,1.95), (0.85,2.6), (0.91,4), (0.944954,6.15), (0.97,8.8), (0.985,12.5), (1,20)) [*Units*: Dmnl]

(107) "f Leaving Leaders Coefficient vs Operative/Expected Functionality Ratio" ([(0,0)-(5,1)],(0,0.5), (0.1,0.3), (0.2,0.18), (0.3,0.12), (0.4,0.08), (0.5,0.04), (0.6,0.01), (0.7,0), (0.8,0), (0.9,0), (1,0), (5,0)) [*Units*: 1/Month]

(108) f Leaving Users Acceleration vs Achieved Functionality ([(0,0)-(5,20)],(0,20), (0.0764526,13.6842), (0.129969,9.91228), (0.197248,6.66667), (0.3,4), (0.42,2.4), (0.6,1.15), (1,1), (5,1)) [*Units*: Dmnl]

(109) f Leaving Users Acceleration vs Perceived Product Quality ([(0,0)-(1,10)],(0,10), (0.088685,9.76316), (0.159021,9.36842), (0.214067,8.85526), (0.262997,8.22368), (0.308868,7.5), (0.342508,6.7), (0.379205,5.78947), (0.425076,4.60526), (0.470948,3.90351), (0.510703,3.20175), (0.562691,2.58772), (0.657492,1.97368), (0.75,1.5), (1,1)) [*Units*: Dmnl]

(110) f New Users Acceleration vs Success in Attracting ([(0,0)-(1,10)],(0,1), (0.0458716,3.91667), (0.0764526,4.91667), (0.143731,6.21053), (0.272171,7.51316), (0.434251,8.73684), (0.6,9.28947), (0.75,9.56579), (0.85,9.72368), (0.944954,9.88158), (1,10)) [*Units*: Dmnl]

(111) f Normal Time to Attract All Potential Developers vs Refusal Ratio ([(0,0)-(1,10)],(0,0.8), (0.1,1), (0.2,1.2), (0.3,1.5), (0.4,2), (0.5,2.5), (0.6,3.4), (0.7,4.5), (0.8,6), (0.9,7.8), (1,10)) [*Units*: Dmnl]

(112) f Optimal Filtering Rate vs Optimal Filtering Horizon ([(0,0)-(5,2)], (0,0), (1e-006,2), (0.5,1.4), (1,1), (2,0.5), (3,0.333), (4,0.25), (5,0.2), (6,0.166), (7,0.143), (8,0.125), (9,0.111), (10,0.1), (20,0.05), (100,0.01)) [*Units*: 1/Month]

(113) f Perceived Product Quality vs Severity of Total Bugs Problem ([(0,0)-(40,1)],(0,1), (0.1,1), (0.963303,0.964912), (1.36086,0.921053), (1.80428,0.79386), (2.15596,0.600877), (2.53823,0.381579), (3.0581,0.22807), (3.59327,0.135965), (4.29664,0.0745614), (5,0.05), (10,0.02), (30,0)) [*Units*: Dmnl]

(114) f Pressure for Bug Detection vs Perceived Product Quality ([(0,0)-(1,1)],(0,1), (0.214067,0.982456), (0.360856,0.942982), (0.5,0.846491), (0.59633,0.758772), (0.688073,0.635965), (0.752294,0.45614), (0.788991,0.276316), (0.83792,0.149123), (0.899083,0.0526316), (1,0)) [*Units*: Dmnl]

(115) f Pressure for Bug Fixing vs Severity of Known Bugs Problem ([(0,0)-(5,1)],(0,0), (1,0), (1.46789,0.0657895), (1.92661,0.20614), (2.27829,0.429825), (2.53823,0.618421), (2.98165,0.828947), (3.50153,0.921053), (4.09786,0.986842), (5,0.99), (10,0.99), (50,0.99)) [*Units*: Dmnl]

(116) "f Pressure for Production vs Achieved/Expected Functionality Ratio" ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.9), (0.4,0.8), (0.5,0.6), (0.6,0.4), (0.7,0.23), (0.8,0.11), (0.9,0.03), (1,0), (20,0)) [*Units*: Dmnl]

(117) "f Pressure for Production vs Operative/Expected Functionality Ratio" ([(0,0)-(1,1)],(0,1), (0.1,0.98), (0.2,0.95), (0.3,0.9), (0.4,0.8), (0.5,0.6), (0.6,0.4), (0.7,0.23), (0.8,0.11), (0.9,0.03), (1,0), (20,0)) [*Units*: Dmnl]

(118) f Pressure for Talent Building vs Talent Building Opportunity ([(0,0)-(1,1)],(0,0), (1,1)) [*Units*: Dmnl]

(119) f Quality Improvement by Filtering vs Quality of Filtering ([(0,0)-(1,0.3)],(0,0), (0.125,0.01), (0.25,0.03), (0.375,0.058), (0.5,0.09), (0.625,0.128), (0.75,0.17), (0.875,0.225), (1,0.3)) [*Units*: Dmnl]

(120) f Quality of Filtering vs Relative Filtering Rate ([(0,0)-(10,1)],(0,1), (0.5,1), (1,1), (1.6,0.84), (2.5,0.64), (3.2,0.52), (4,0.4), (5,0.28), (6.2,0.18), (7.5,0.1), (8.5,0.05), (10,0)) [*Units*: Dmnl]

(121) f Time to Lose Patience vs Limit on Product Functionality ([(0,0)-(20000,8)],(0,0), (400,1), (1000,1.68), (2000,2.6), (3000,3.4), (4000,4), (7000,5.2), (10000,6), (20000,7.5)) [*Units*: Dmnl]

(122) f Weight on Expected Functionality Ratio vs Expected Functionality Ratio ([(0,0)-(1,1)],(0,1), (0.1,1), (0.12,0.98), (0.14,0.94), (0.1666,0.85), (0.185,0.72), (0.2,0.5), (0.215,0.28), (0.2333,0.15), (0.26,0.06), (0.28,0.02), (0.3,0), (1,0)) [*Units*: Dmnl]

(123) Filtering Rate = 0.5 [*Units*: 1/Month]

(124) FINAL TIME = 100 [*Units*: Month]

(125) Functionality Lost by Debugging = Functionality Lost per Bug Fixed * Bugs Fixed [*Units*: UF/Month]

(126) Functionality Lost per Bug Fixed = ZIDZ (f Functionality Lost per Bug Fixed vs Debugging Quality (Bug Fixing Quality) , Total Bugs per Functionality) [*Units*: UF/bug]

(127) Functionality per Code = Product Functionality / Project Size [*Units*: UF/line]

(128) Increase in Limit on Product Functionality = Increase in Limit on Product Functionality Coefficient * Limit on Product Functionality [*Units*: UF/Month]

(129) Increase in Limit on Product Functionality Coefficient = 0.002 [*Units*: 1/Month]

(130) Indicated Developer Hours Revised Allocation Factor = f Developer Hours Revised Allocation Factor vs Pressure for Production (Pressure for Production) * Developer Hours Allocation Factor [*Units*: Dmnl]

(131) Indicated Leader Hours Revised Allocation Factor = f Leader Hours Revised Allocation Factor vs Pressure for Production (Pressure for Production on Leaders) * Leader Hours Allocation Factor [*Units*: Dmnl]

(132) Initial Bugs in Production to be Filtered = 0 [*Units*: bugs]

(133) Initial Developer Hours Revised Allocation Factor = f Initial Developer Hours Revised Allocation Factor vs initial Average Developer Participation (Average Developer Participation) [*Units*: Dmnl]

(134) Initial Developer Talent Pool = Average Incoming Developer Talent * Initial Developers [*Units*: RTU]

(135) Initial Developers = 7 [*Units*: people]

(136) Initial Developers on Other Projects = Initial Developers on Other Projects per Limit on Product Functionality * Limit on Product Functionality [*Units*: people]

(137) Initial Developers on Other Projects per Limit on Product Functionality = 0.1 [*Units*: people/UF]

(138) Initial Functionality = 0 [*Units*: UF]

(139) Initial Known Bugs = 0 [*Units*: bugs]

(140) Initial Leader Hours Revised Allocation Factor = f Initial Leader Hours Revised Allocation Factor vs initial Leader Hours Coverage Ratio (Leader Hours Coverage Ratio) [*Units*: Dmnl]

(141) Initial Leaders = 3 [*Units*: people]

(142) Initial Limit on Product Functionality = 400 [*Units*: UF]

(143) Initial Patience = 1 [*Units*: Dmnl]

(144) Initial Potential Developers = Initial Potential Developers per Limit on Product Functionality * Limit on Product Functionality [*Units*: people]

(145) Initial Potential Developers per Limit on Product Functionality = 0.025 [*Units*: people/UF]

(146) Initial Potential Users = Initial Potential Users per Limit on Product Functionality * Limit on Product Functionality [*Units*: people]

(147) Initial Potential Users per Limit on Product Functionality = 20 [*Units*: people/UF]

(148) Initial Production to be Filtered = 0 [*Units*: lines]

(149) Initial Project Size = 0.012 [*Units*: lines]

(150) INITIAL TIME = 0 [*Units*: Month]

(151) Initial Unknown Bugs = 0 [*Units*: bugs]

(152) Initial Users = 0 [*Units*: people]

(153) Initial Users Using Competitor Products = Initial Users Using Competitor Products per Limit on Product Functionality * Limit on Product Functionality [*Units*: people]

(154) Initial Users Using Competitor Products per Limit on Product Functionality = 30 [*Units*: people/UF]

(155) Known Bugs in Code = INTEG(Bugs Found - Bugs Fixed , Initial Known Bugs) [*Units*: bugs]

(156) Known Bugs per Code = Known Bugs in Code / Project Size [*Units*: bugs/line]

(157) Known Bugs per Functionality = ZIDZ (Known Bugs in Code , Product Functionality) [*Units*: bugs/UF]

(158) Leader Bug Discovery Rate = Bug Discovery Rate Normal * f Bug Discovery Efficiency vs Unknown Bugs Density (Unknown Bug Density) [*Units*: bugs/hour]

(159) Leader Bug Fixing Rate = 1 [*Units*: bugs/hour]

(160) Leader Bug Generating Rate = Bug Generating Rate Normal * f Bug Generating Rate vs Average Relative Talent (Average Relative Leader Talent) [*Units*: bugs/line]

(161) Leader Hours Allocated to Bug Detection = Leader Hours Revised Allocation Factor * Leader Hours Needed for Bug Detection [*Units*: hours/Month]

(162) Leader Hours Allocated to Bug Fixing = Leader Hours Revised Allocation Factor * Leader Hours Needed for Bug Fixing [*Units*: hours/Month]

(163) Leader Hours Allocated to Coaching = Leader Hours Revised Allocation Factor * Leader Hours Planned for Coaching [*Units*: hours/Month]

(164) Leader Hours Allocated to Production = Total Leader Hours Available - "Total Leader Hours Allocated for Non-Production Tasks" [*Units*: hours/Month]

(165) Leader Hours Allocation Factor = f Leader Hours Allocation Factor vs Leader Hours Coverage Ratio (Leader Hours Coverage Ratio) [*Units*: Dmnl]

(166) Leader Hours Availability for Coaching = ZIDZ (Leader Hours Needed for Coaching , Maximum Total Leader Hours Available for Coaching) [*Units*: Dmnl]

(167) Leader Hours Coverage Ratio = ZIDZ (Total Leader Hours Available , Total Leader Hours Needed) [*Units*: Dmnl]

(168) Leader Hours Needed for Bug Detection = ZIDZ (Developer Hours for Bug Detection Gap , "Leader/Developer Bug Discovery Efficiency Ratio") [*Units*: hours/Month]

(169) Leader Hours Needed for Bug Fixing = ZIDZ (Developer Hours for Bug Fixing Gap , "Leader/Developer Bug Fixing Efficiency Ratio") [*Units*: hours/Month]

(170) Leader Hours Needed for Coaching = Developer Hours Needed for Coaching [*Units*: hours/Month]

(171) Leader Hours Needed for Filtering = 1 [*Units*: hours/Month]

(172) Leader Hours Needed for Selecting = 1 [*Units*: hours/Month]

(173) Leader Hours Planned for Coaching = f Leader Hours Planned for Coaching vs Leader Hours Availability for Coaching (Leader Hours Availability for Coaching) * Leader Hours Needed for Coaching [*Units*: hours/Month]

(174) Leader Hours Planned for Production = Total Leader Hours Available [*Units*: hours/Month]

(175) Leader Hours Revised Allocation Factor = INTEG(Leader Hours Revised Allocation Factor Adjustment , Initial Leader Hours Revised Allocation Factor) [*Units*: Dmnl]

(176) Leader Hours Revised Allocation Factor Adjustment = Leader Hours Revised Allocation Factor Adjustment Discrepancy / Leader Hours Revised Allocation Factor Adjustment Time [*Units*: 1/Month]

(177) Leader Hours Revised Allocation Factor Adjustment Discrepancy = Indicated Leader Hours Revised Allocation Factor - Leader Hours Revised Allocation Factor [*Units*: Dmnl]

(178) Leader Hours Revised Allocation Factor Adjustment Time = 4 [*Units*: Month]

(179) "Leader/Developer Bug Discovery Efficiency Ratio" = IF THEN ELSE (Developer Bug Discovery Rate > 0, (Leader Bug Discovery Rate / Developer Bug Discovery Rate) , 1.308) [*Units*: Dmnl]

(180) "Leader/Developer Bug Fixing Efficiency Ratio" = ZIDZ (Leader Bug Fixing Rate , Developer Bug Fixing Rate) [*Units*: Dmnl]

(181) "Leader/Developer Coaching Ratio" = 1 [*Units*: Dmnl]

(182) Leaders = INTEG(- Leaving Leaders , Initial Leaders) [*Units*: people]

(183) Leaders Coaching Involvement Factor = 0.9 [*Units*: Dmnl]

(184) Leaving Accelaration Due to Low Achieved Functionality = f Leaving Accelaration vs Achieved Functionality ("Operative/Expected Functionality Ratio") [*Units*: Dmnl]

(185) Leaving Accelaration Due to Low Quality = f Leaving Accelaration vs Perceived Product Quality (Perceived Product Quality) [*Units*: Dmnl]

(186) Leaving Acceleration Due to Potential Functionality = f Leaving Accelaration vs Potential Functionality (Achieved Functionality Ratio) [*Units*: Dmnl]

(187) Leaving Developers = Leaving Acceleration Due to Potential Functionality * Leaving Accelaration Due to Low Achieved Functionality * Leaving Accelaration Due to Low Quality * Developers / Normal Time for Developers to Leave [*Units*: people/Month]

(188) Leaving Developers from Other Projects = Developers on Other Projects / Normal Time for Developers to Leave [*Units*: people/Month]

(189) Leaving Leaders = Leaders * Leaving Leaders Coefficient [*Units*: people/Month]

(190) Leaving Leaders Coefficient = "f Leaving Leaders Coefficient vs Operative/Expected Functionality Ratio" ("Operative/Expected Functionality Ratio") [*Units*: 1/Month]

(191) Leaving Users = Leaving Users Acceleration Due to Low Achieved Functionality * Leaving Users Acceleration Due to Low Quality * Users / Normal Time to Lose All Users [*Units*: people/Month]

(192) Leaving Users Acceleration Due to Low Achieved Functionality = f Leaving Users Acceleration vs Achieved Functionality ("Operative/Expected Functionality Ratio") [*Units*: Dmnl]

(193) Leaving Users Acceleration Due to Low Quality = f Leaving Users Acceleration vs Perceived Product Quality (Perceived Product Quality) [*Units*: Dmnl]

(194) Leaving Users from Competitor Products = Users Using Competitor Products / Normal Time to Lose All Users [*Units*: people/Month]

(195) Limit on Product Functionality = INTEG(Increase in Limit on Product Functionality , Initial Limit on Product Functionality) [*Units*: UF]

(196) Maximum Coaching Hours Needed per Developer = 10 [*Units*: hours/(Month*people)]

(197) Maximum Developer Talent = 1 [*Units*: RTU/people]

(198) Maximum Developer Talent Building Ratio = 0.1 [*Units*: 1/Month]

(199) Maximum Talent Building Opportunity = 1 [*Units*: RTU/people]

(200) Maximum Total Leader Hours Available for Coaching = Leaders Coaching Involvement Factor * Total Leader Hours Available [*Units*: hours/Month]

(201) New Bugs Added by Bug Fixes = Bugs Added per Bug Fixed * Bugs Fixed [*Units*: bugs/Month]

(202) New Bugs Added by Production = Bugs in Accepted Code + (Leader Bug Generating Rate * Production by Leaders) [*Units*: bugs/Month]

(203) New Bugs in Production to be Filtered = (Developer Bug Generating Rate * Production by Developers) [*Units*: bugs/Month]

(204) New Product Functionality Added = Product Functionality Adding Efficiency * Total Production [*Units*: UF/Month]

(205) New Users = New Users Acceleration Due to Success in Attracting * Attrractiveness of Product for Users * Potential Users / Normal Time to Attract All Potential Users [*Units*: people/Month]

(206) New Users Acceleration Due to Success in Attracting = f New Users Acceleration vs Success in Attracting (Success in Attracting Users) [*Units*: Dmnl]

(207) Normal Time for Developers to Leave = 96 [*Units*: Month]

(208) Normal Time to Attract All Potential Developers = f Normal Time to Attract All Potential Developers vs Refusal Ratio (Refusal Ratio) * Time to Attract Developers Normal [*Units*: Month]

(209) Normal Time to Attract All Potential Users = 36 [*Units*: Month]

(210) Normal Time to Lose All Potential Developers to Other Projects = 10 [*Units*: Month]

(211) Normal Time to Lose All Potential Users to Competitor Products = 36 [*Units*: Month]

(212) Normal Time to Lose All Users = 60 [*Units*: Month]

(213) Normal Time to Lose Patience = 25 [*Units*: Month]

(214) Operative Functionality Ratio = (Weight on Expected Functionality Ratio * Expected Funtionality Ratio) + (Weight on Achieved Functionality Ratio * Achieved Functionality Ratio) [*Units*: Dmnl]

(215) "Operative/Expected Functionality Ratio" = Operative Functionality Ratio / Expected Funtionality Ratio [*Units*: Dmnl]

(216) Optimal Filtering Amount = Optimal Filtering Amount per Leader * Leaders [*Units*: lines/Month]

(217) Optimal Filtering Amount per Leader = 3000 [*Units*: lines/(Month*people)]

(218) Optimal Filtering Horizon = ZIDZ (Production to be Filtered , Optimal Filtering Amount) [*Units*: Month]

(219) Optimal Filtering Rate = f Optimal Filtering Rate vs Optimal Filtering Horizon (Optimal Filtering Horizon) [*Units*: 1/Month]

(220) Overall Attractiveness of Product for Developers = (Attractiveness of Product for Developers Due to Achieved Functionality * Attractiveness of Product for Developers Due to Potential Functionality * Attractiveness of Product for Developers Due to Users) [*Units*: Dmnl]

(221) Participant Population Intensity = (Developers + Leaders) / Productive Participant Population Limit [*Units*: Dmnl]

(222) Patched Code = Code Added per Bug Fixed * Bugs Fixed [*Units*: lines/Month]

(223) Patience = INTEG(- Patience Lost , Initial Patience) [*Units*: Dmnl]

(224) Patience Lost = Patience / Time to Lose Patience [*Units*: 1/Month]

(225) Perceived Product Quality = f Perceived Product Quality vs Severity of Total Bugs Problem (Severity of Total Bugs Problem) [*Units*: Dmnl]

(226) Potential Developers = INTEG(Leaving Developers + Leaving Developers from Other Projects + Candidates Refused - Candidates Applying - Potential Developers Choosing Other Projects , Initial Potential Developers) [*Units*: people]

(227) Potential Developers Choosing Other Projects = Potential Developers / Normal Time to Lose All Potential Developers to Other Projects [*Units*: people/Month]

(228) Potential Users = INTEG(Leaving Users + Leaving Users from Competitor Products - New Users - Potential Users Choosing Competitor Products , Initial Potential Users) [*Units*: people]

(229) Potential Users Choosing Competitor Products = Potential Users / Normal Time to Lose All Potential Users to Competitor Products [*Units*: people/Month]

(230) Pressure for Bug Detection = f Pressure for Bug Detection vs Perceived Product Quality (Perceived Product Quality) [*Units*: Dmnl]

(231) Pressure for Bug Fixing = f Pressure for Bug Fixing vs Severity of Known Bugs Problem (Severity of Known Bugs Problem) [*Units*: Dmnl]

(232) Pressure for Production = "f Pressure for Production vs Operative/Expected Functionality Ratio" ("Operative/Expected Functionality Ratio") [*Units*: Dmnl]

(233) Pressure for Production on Leaders = "f Pressure for Production vs Achieved/Expected Functionality Ratio" ("Achieved/Expected Functionality Ratio") [*Units*: Dmnl]

(234) Pressure for Talent Building = f Pressure for Talent Building vs Talent Building Opportunity (Relative Average Talent Building Opportunity) [*Units*: Dmnl]

(235) Product Functionality = INTEG(New Product Functionality Added - Functionality Lost by Debugging , Initial Functionality) [*Units*: UF]

(236) Product Functionality Adding Efficiency = f Functionality Adding Efficiency vs Achieved Ratio (Achieved Functionality Ratio) * Product Functionality Adding Efficiency Normal [*Units*: UF/line]

(237) Product Functionality Adding Efficiency Normal = 0.006 [*Units*: UF/line]

(238) Production by Developers = Average Developer Productivity * Developer Hours Allocated to Production [*Units*: lines/Month]

(239) Production by Leaders = Average Leader Productivity * Leader Hours Allocated to Production [*Units*: lines/Month]

(240) Production to be Filtered = INTEG(Production by Developers - Accepted Production - Rejected Production , Initial Production to be Filtered) [*Units*: lines]

(241) Productive Participant Population Limit = 100 [*Units*: people]

(242) Project Size = INTEG(Patched Code + Total Production , Initial Project Size) [*Units*: lines]

(243) Quality Improvement by Filtering = f Quality Improvement by Filtering vs Quality of Filtering (Quality of Filtering) [*Units*: Dmnl]

(244) Quality of Filtering = f Quality of Filtering vs Relative Filtering Rate (Relative Filtering Rate) [*Units*: Dmnl]

(245) Refusal Ratio = 0.1 [*Units*: Dmnl]

(246) Rejected Production = Production to be Filtered * Filtering Rate * Rejection Ratio [*Units*: lines/Month]

(247) Rejection Ratio = 0.2 [*Units*: Dmnl]

(248) Relative Average Talent Building Opportunity = Average Developer Talent Building Opportunity / Maximum Talent Building Opportunity [*Units*: Dmnl]

(249) Relative Filtering Rate = ZIDZ (Filtering Rate , Optimal Filtering Rate) [*Units*: Dmnl]

(250) SAVEPER = TIME STEP [*Units*: Month]

(251) Selecting Rate = 0.5 [*Units*: 1/Month]

(252) Severity of Known Bugs Problem = Known Bugs per Functionality / Acceptable Level of Known Bugs per Functionality [*Units*: Dmnl]

(253) Severity of Total Bugs Problem = Total Bugs per Functionality / Acceptable Level of Total Bugs per Functionality [*Units*: Dmnl]

(254) Success in Attracting Users = Users / Total User Population [*Units*: Dmnl]

(255) TIME STEP = 0.125 [*Units*: Month]

(256) Time to Attract Developers Normal = 10 [*Units*: Month]

(257) Time to Lose Patience = f Time to Lose Patience vs Limit on Product Functionality (Limit on Product Functionality) * Normal Time to Lose Patience [*Units*: Month]

(258) Total Bugs in Code = Known Bugs in Code + Unknown Bugs in Code [*Units*: bugs]

(259) Total Bugs per Functionality = ACTIVE INITIAL(ZIDZ (Total Bugs in Code , Product Functionality) , 0.6) [*Units*: bugs/UF]

(260) Total Coaching Hours Available = "Leader/Developer Coaching Ratio" * Leader Hours Allocated to Coaching [*Units*: hours/Month]

(261) "Total Developer Hours Allocated for Non-Production Tasks" = Developer Hours Allocated to Bug Detection + Developer Hours Allocated to Bug Fixing + Developer Hours Allocated to Coaching [*Units*: hours/Month]

(262) Total Developer Hours Available = Average Developer Participation * Developers [*Units*: hours/Month]

(263) Total Developer Hours Needed = Developer Hours Planned for Production + "Total Developer Hours Needed for Non-Production Tasks" [*Units*: hours/Month]

(264) "Total Developer Hours Needed for Non-Production Tasks" = Developer Hours Needed for Bug Detection + Developer Hours Needed for Bug Fixing + Developer Hours Planned for Coaching [*Units*: hours/Month]

(265) "Total Leader Hours Allocated for Non-Production Tasks" = Leader Hours Allocated to Bug Detection + Leader Hours Allocated to Bug Fixing + Leader Hours Allocated to Coaching [*Units*: hours/Month]

(266) Total Leader Hours Available = Average Leader Participation * Leaders [*Units*: hours/Month]

(267) Total Leader Hours Needed = Leader Hours Planned for Production + "Total Leader Hours Needed for Non-Production Tasks" [*Units*: hours/Month]

(268) "Total Leader Hours Needed for Non-Production Tasks" = Leader Hours Needed for Bug Detection + Leader Hours Needed for Bug Fixing + Leader Hours Planned for Coaching [*Units*: hours/Month]

(269) Total Participants = Leaders + Developers [*Units*: people]

(270) Total Production = Production by Leaders + Accepted Production [*Units*: lines/Month]

(271) Total User Population = Users + Potential Users + Users Using Competitor Products [*Units*: people]

(272) Unknown Bug Density = ZIDZ (Unknown Bugs per Code , Bug Generating Rate Normal) [*Units*: Dmnl]

(273) Unknown Bugs in Code = INTEG(New Bugs Added by Production - Bugs Found + New Bugs Added by Bug Fixes , Initial Unknown Bugs) [*Units*: bugs]

(274) Unknown Bugs per Code = Unknown Bugs in Code / Project Size [*Units*: bugs/line]

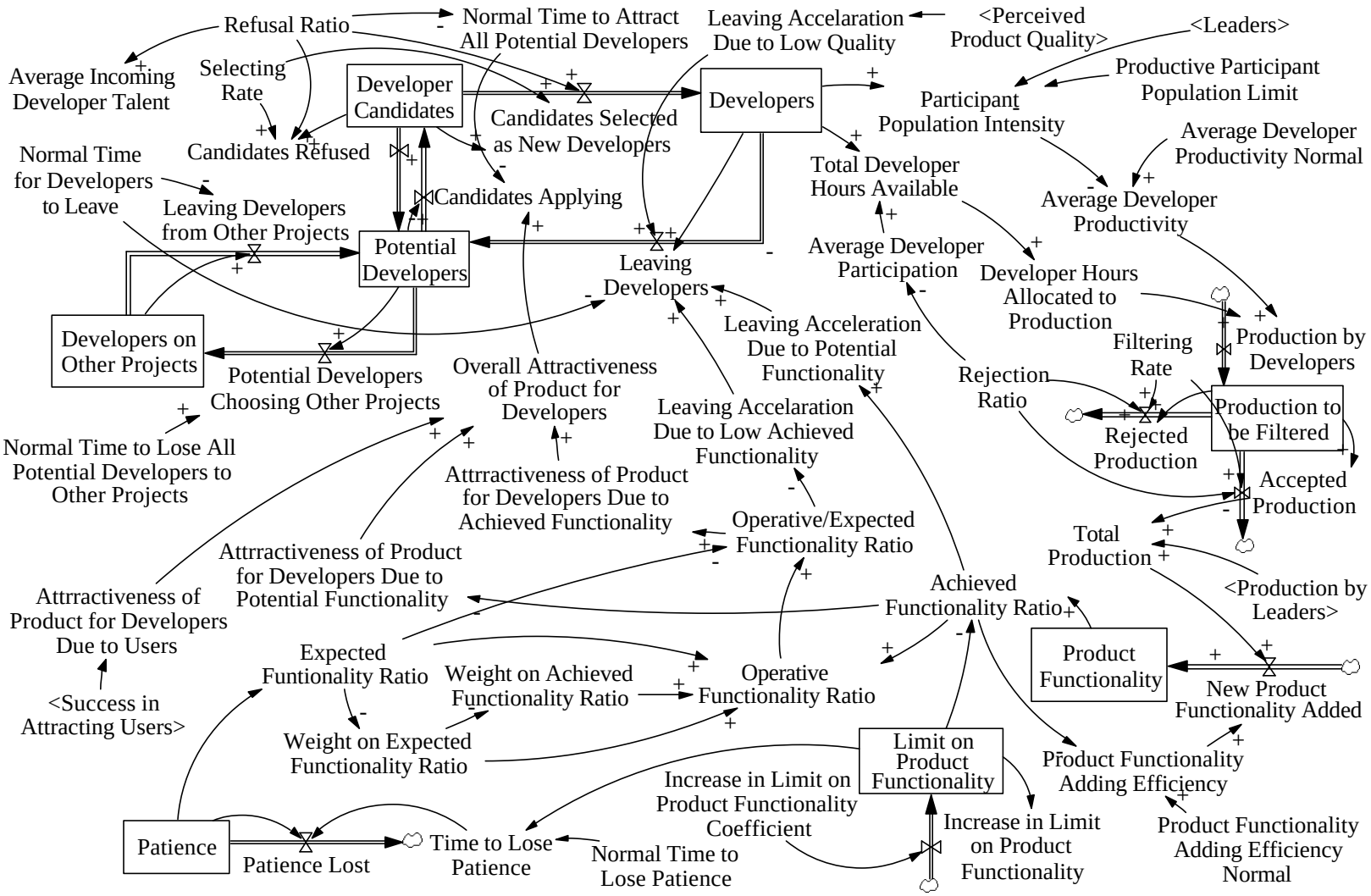
(275) Users = INTEG(New Users - Leaving Users , Initial Users) [*Units*: people]

(276) Users Using Competitor Products = INTEG(Potential Users Choosing Competitor Products - Leaving Users from Competitor Products , Initial Users Using Competitor Products) [*Units*: people]

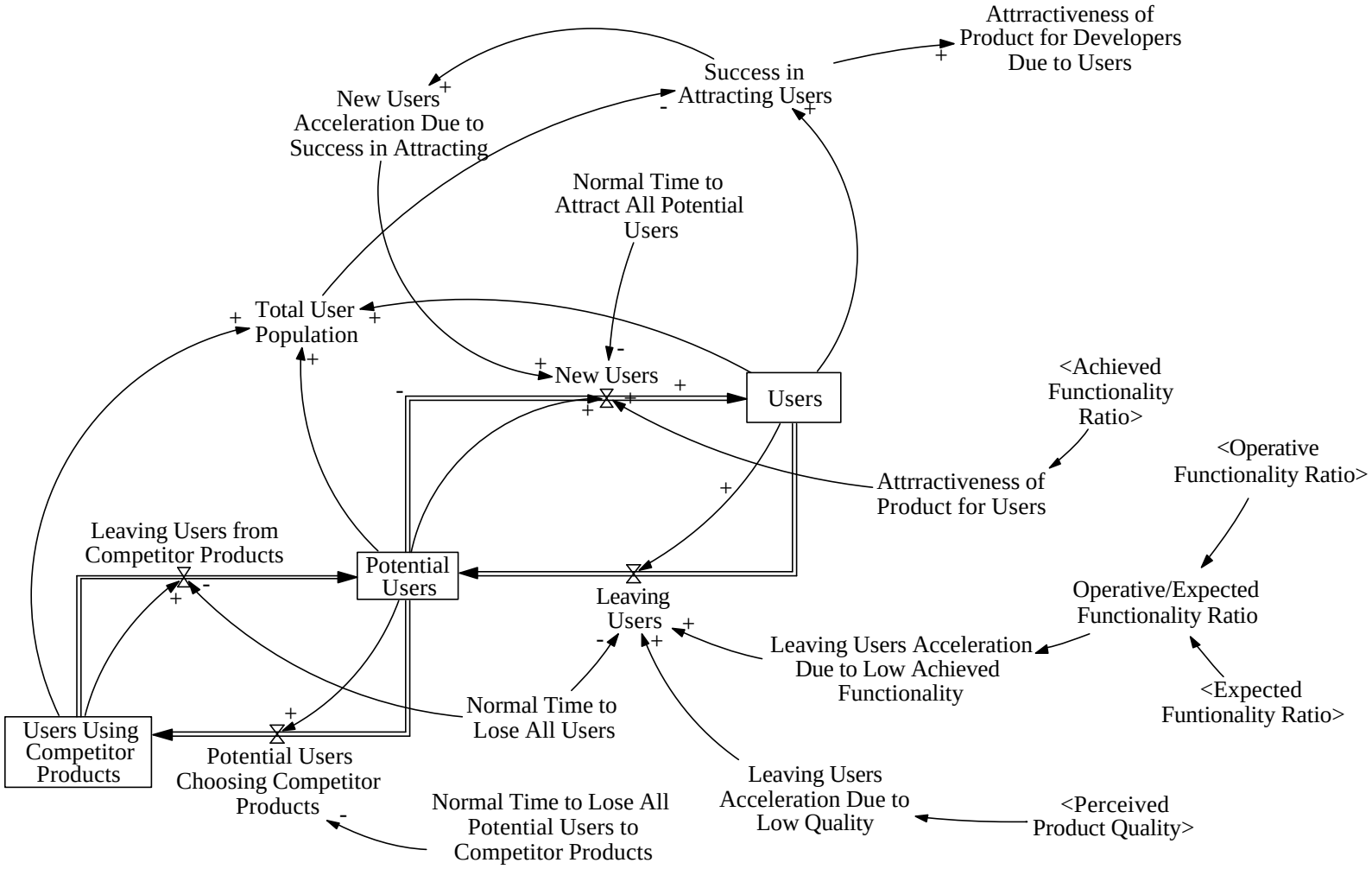
(277) Weight on Achieved Functionality Ratio = 1 - Weight on Expected Functionality Ratio [*Units*: Dmnl]

(278) Weight on Expected Functionality Ratio = f Weight on Expected Functionality Ratio vs Expected Functionality Ratio (Expected Functionality Ratio) [*Units*: Dmnl]

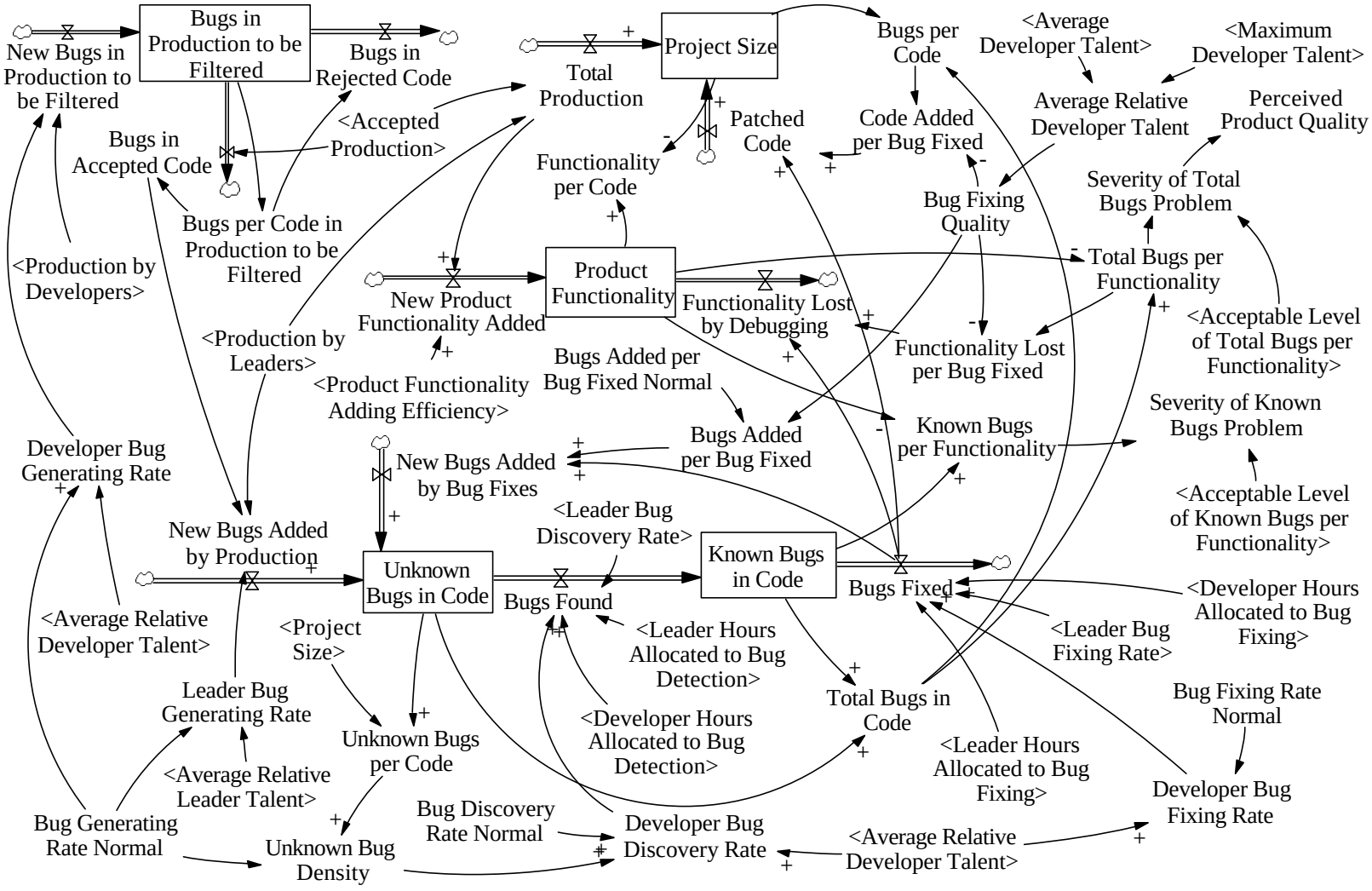
B.2. Model Sector Views (Iteration V Version) **Developers and Production Sector**



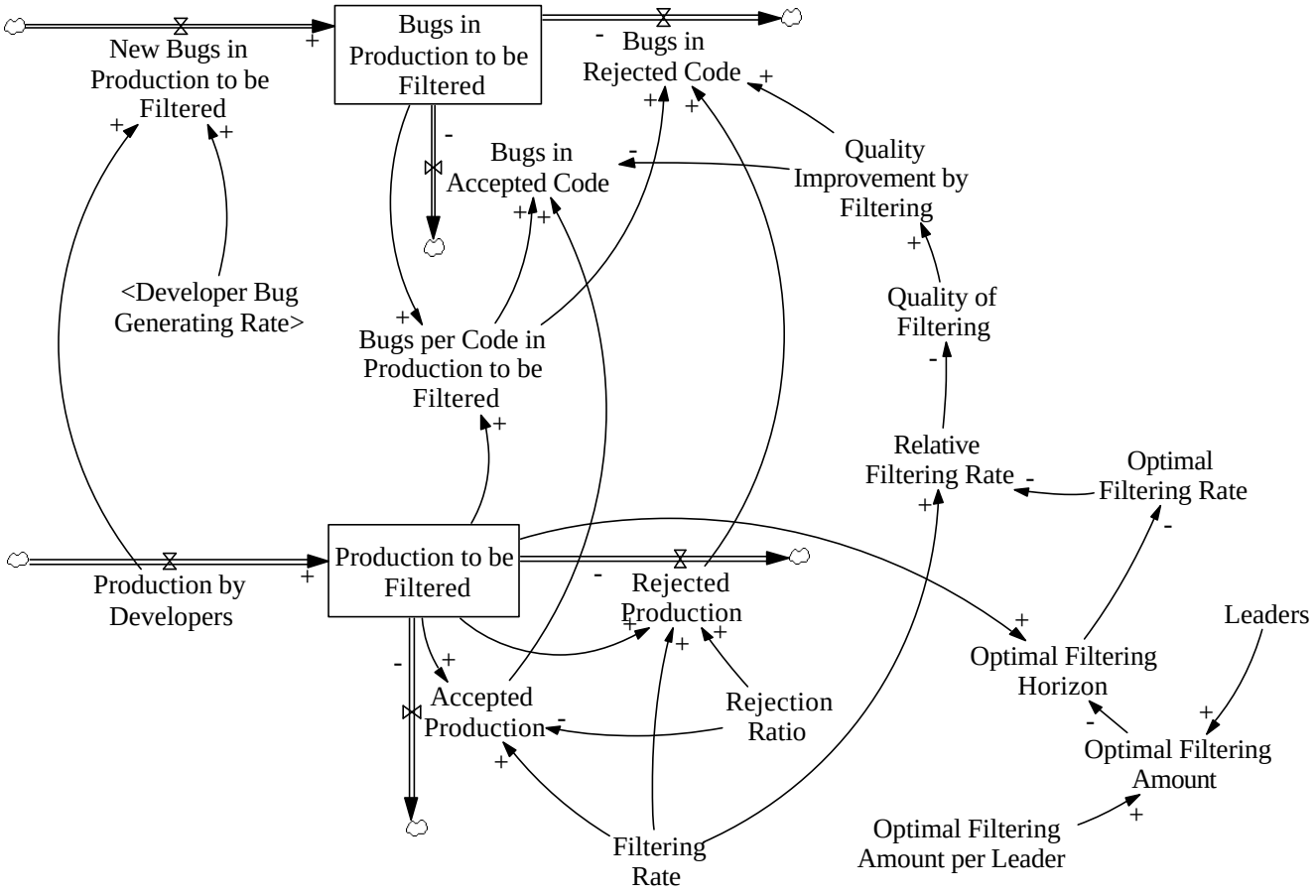
Users Sector



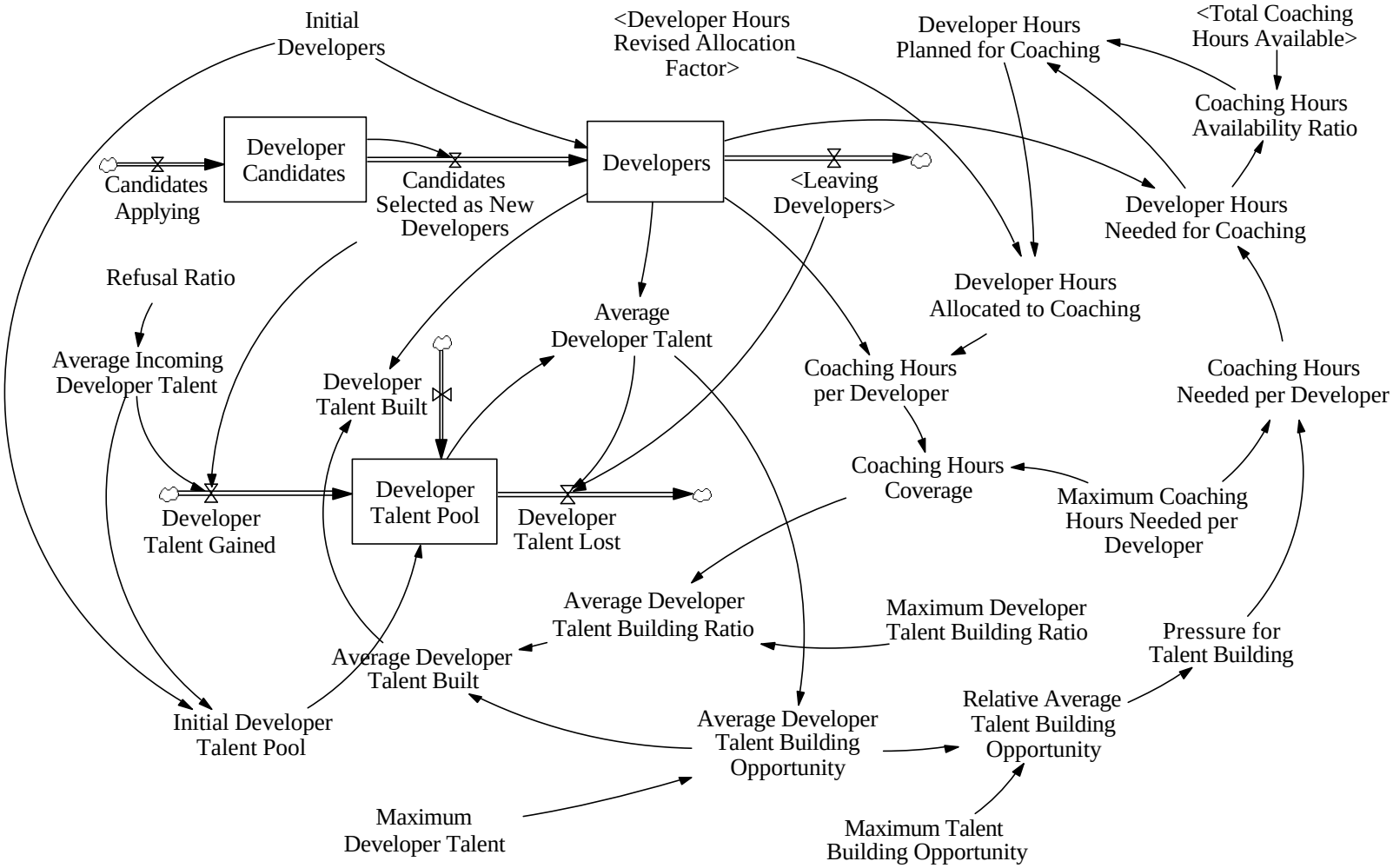
Quality Sector



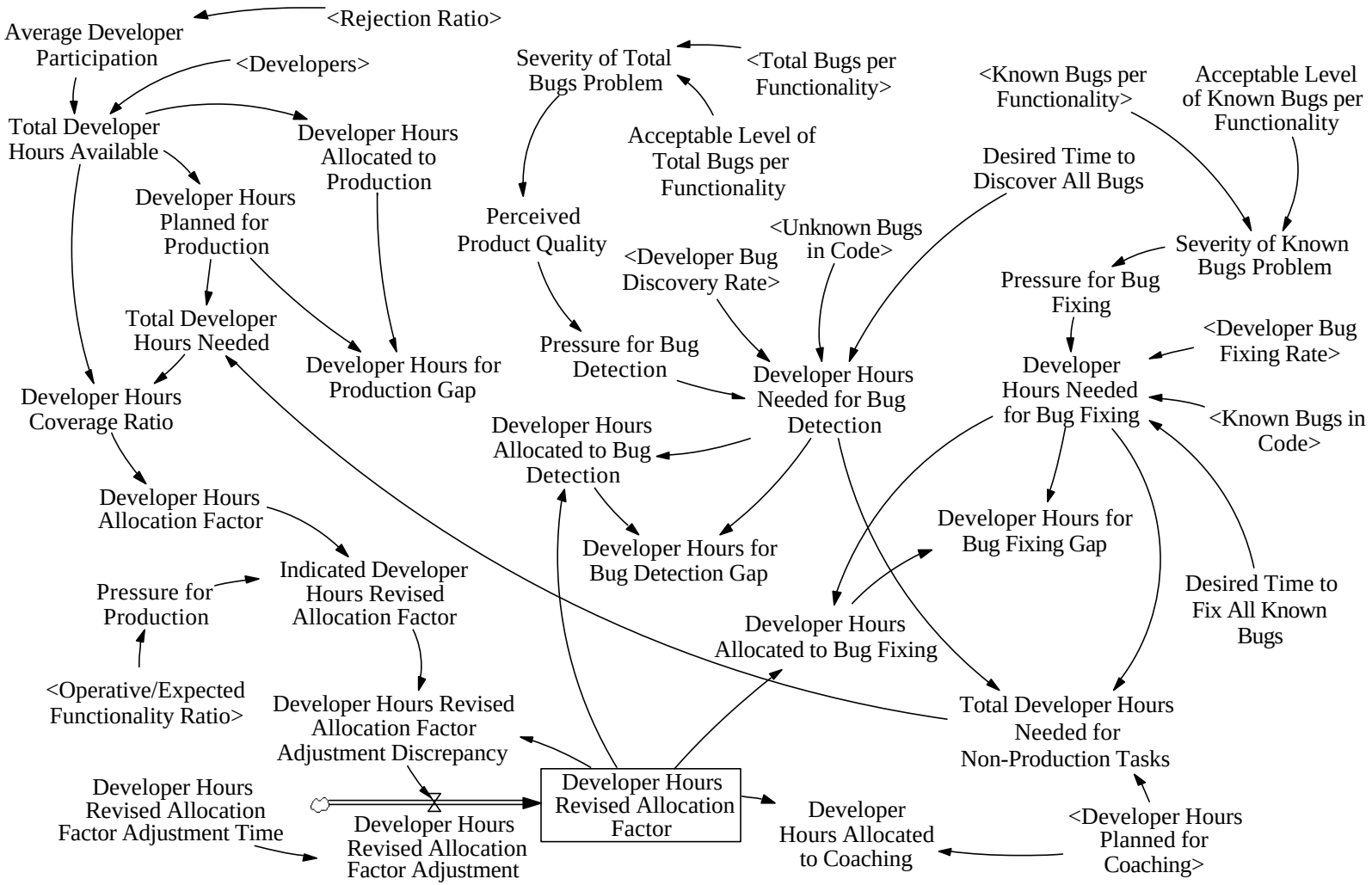
Filtering Sector



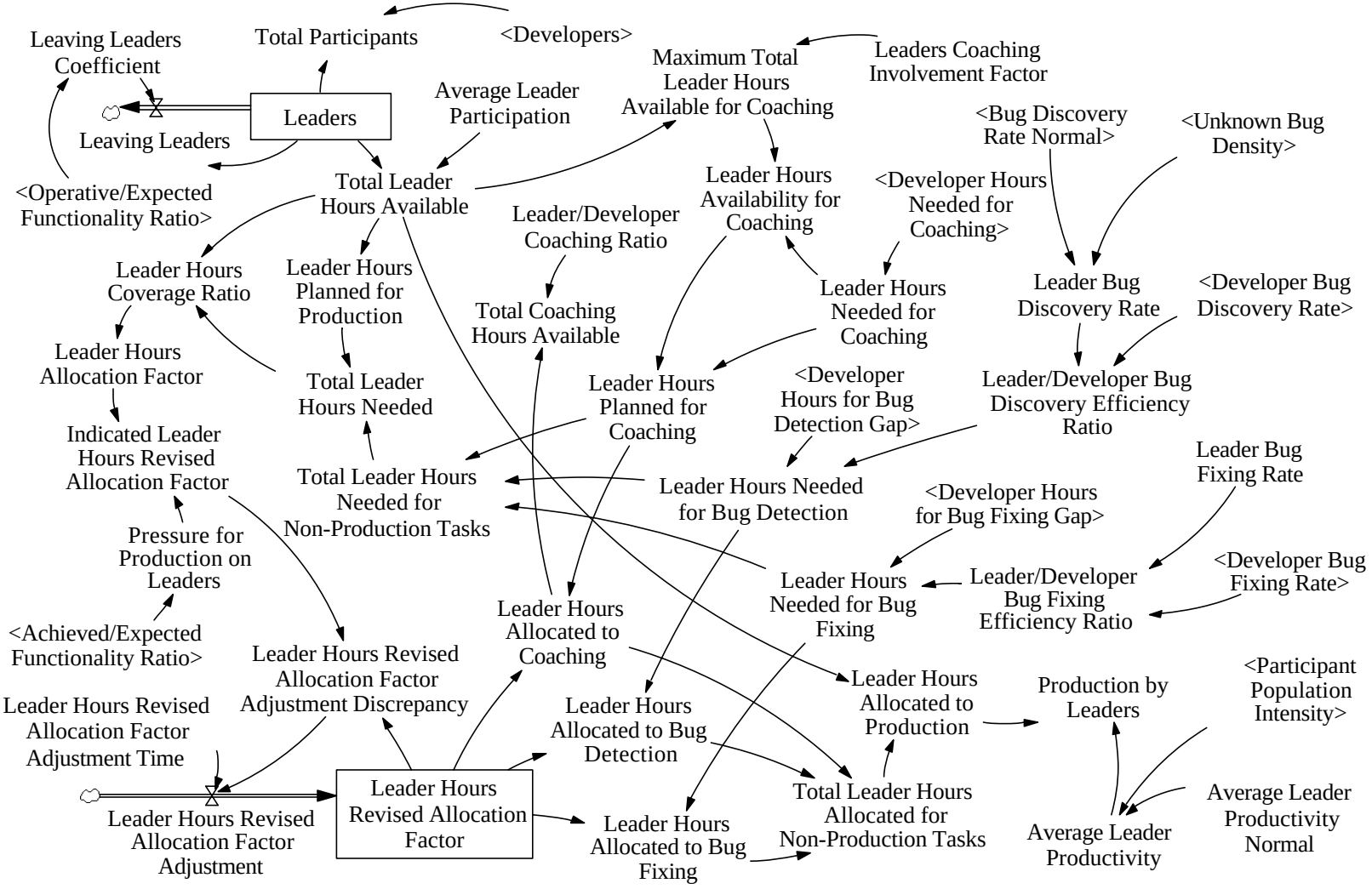
Developer Talent and Coaching Sector



Developer Time Allocation Sector



Leaders Sector



REFERENCES

Abdel-Hamid, T. K. (1984). The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective. Ph.D. Thesis. Massachusetts Institute of Technology.

Abdel-Hamid, T. K. (1989). "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach." IEEE Transactions on Software Engineering **15**(2): 109-119.

Abdel-Hamid, T. K. and S. E. Madnick (1983). "The Dynamics of Software Project Scheduling." Communications of the ACM **26**(5): 340-346.

Abdel-Hamid, T. K. and S. E. Madnick (1989). "Lessons Learned from Modeling the Dynamics of Software Project Management." Communications of the ACM **32**(12): 1426-1438.

Abdel-Hamid, T. K. and S. E. Madnick (1991). Software Project Dynamics: An Integrated Approach. Englewood Cliffs, New Jersey, USA, Prentice Hall.

Abdel-Hamid, T. K. and J. D. W. Morecroft (1983). "A Generic System Dynamics Model of Software Project Management." International System Dynamics Conference, Chestnut Hill, MA.

Andersen, D. F. and G. P. Richardson (1997). "Scripts for Group Model Building." System Dynamics Review **13**(2): 107-129.

Andersen, D. F., G. P. Richardson and J. A. M. Vennix (1997). "Group Model Building: Adding More Science to the Craft." System Dynamics Review **13**(2): 187-201.

Andersen, D. L., M. J. Radzicki, R. L. Spencer and W. S. Trees (1997). "The Dynamics of the Field of System Dynamics." 15th International System Dynamics Conference: "Systems Approach to Learning and Education into the 21st Century", Istanbul, Turkey, Bogazici University Printing Office.

Applegate, L., C. Ellis, C. W. Holsapple, F. J. Radermacher and A. B. Whinston (1991). "Organizational Computing: Definitions and Issues." Journal of Organizational Computing **1**(1): 1-10.

Babbie, E. (1998). The Practice of Social Research. Belmont, CA, Wadsworth Publishing Co.

Barbrook, R. (1998). The Hi-Tech Gift Economy. First Monday **3 (12)** Last Accessed: January 10, 2003
Available:http://www.firstmonday.org/issues/issue3_12/barbrook/index.html

Barlas, Y. (1989). "Multiple Tests for Validation of System Dynamics Type of Simulation Models." European Journal of Operational Research **42**(1): 59-87.

Barlas, Y. and I. Bayraktutar (1992). "An Interactive Simulation Game for Software Project Management (Softsim)." Proceedings of the 1992 International System Dynamics Conference of the System Dynamics Society, Utrecht, the Netherlands, The System Dynamics Society.

Barros, M. d. O., C. M. L. Werner and G. H. Travassos (2000). "Applying System Dynamics To Scenario Based Software Project Management." 18th International Conference of the System Dynamics Society, Bergen, Norway, System Dynamics Society.

Bays, H. and M. Mowbray (2001). Cookies, Gift-Giving, and Online Communities. Online Communities. C. Werry and M. Mowbray. Upper Saddle River, NJ, Prentice Hall PTR.

Bell, D. (1991). "Modes of Exchange: Gift and Commodity." Journal of Socio-Economics **20**(2): 155-167.

Bell, G. A. and J. O. Jenkins (1998). "Methods Chosen to Identify Dominant Feedback Loops that Explain Software Project Cost." 16th International Conference of the System Dynamics Society, Quebec '98, Quebec City, Canada, System Dynamics Society.

Bessen, J. (2002). "Open Source Software: Free Provision of Complex Public Goods." Open Source Software: Economics, Law and Policy, Toulouse, France, Institut d'Economie Industrielle.

Bezroukov, N. (1999). Open Source Software Development as a Special Type of Academic Research. First Monday Last Accessed: April 20, 2002 Available: http://www.firstmonday.org/issues/issue4_10/bezroukov/index.html

Bourdieu, P. (1997). Marginalia--Some Additional Notes on the Gift. The Logic of the Gift: Toward an Ethic of Generosity. A. D. Schrift. New York, NY, Routledge.

Brewer, J. and A. Hunter (1989). Multimethod Research: A Synthesis of Styles. Newbury Park, CA, Sage.

Brooks, F. P. (1995). The Mythical Man-Month - 20th Anniversary Edition. Reading, MA, Addison-Wesley.

Browne, C. B. (1998). Linux and Decentralized Development. First Monday Last Accessed: November 09, 2002 Available: http://www.firstmonday.org/issues/issue3_3/browne/index.html

Carrier, J. (1991). "Gifts, Commodities, and Social Relations: A Maussian View of Exchange." Sociological Forum **6**(1): 119-136.

Cowen, T. (1993). Public Goods and Externalities. The Fortune Encyclopedia of Economics. D. R. Henderson. New York, NY, Warner Books**:** 74-77.

Cox, A. (1998). Cathedrals, Bazaars and the Town Council. Slashdot Last Accessed: November 09, 2002 Available:http://slashdot.org/features/98/10/13/1423253.shtml

Dempsey, B. J., D. Weiss, P. Jones and J. Greenberg (2002). "Who is an open source software developer?" Communications of the ACM **45**(2): 67-72.

Diker, V. G. and H. J. Scholl (1999). "David vs. Goliath: Responses to Domination Strategies in PC and Server OS Markets." 17th International Conference of the System Dynamics Society, Wellington, New Zealand.

Diker, V. G. and H. J. Scholl (2001). "The Art of Leveraging : How Powerful Nonlinear Feedback Processes Can Restructure Rapidly Growing Technology and Knowledge Industries." 34th Annual Hawaii International Conference on System Sciences, Maui, HI.

Donzelli, P. and G. Iazeolla (2001). "Hybrid Simulation Modeling of the Software Process." The Journal of Systems and Software **59**: 227-235.

Eason, K. (1997). Understanding the Organisational Ramifications of Implementing Information Technology Systems. Handbook of Human-Computer Interaction. M. G. Helander, T. K. Landauer and P. V. Prabhu. Amsterdam, Elsevier Science**:** 1475-1495.

Fogel, K. and M. Bar (2001). Open Source Development with CVS. Scottsdale, AZ, Coriolis Technology Press.

Forrester, J. W. (1961). Industrial Dynamics. Cambridge, MA, Productivity Press.

Forrester, J. W. and P. M. Senge (1996). Tests for Building Confidence in System Dynamics Models. Modeling for Management: Simulation in Support of Systems Thinking. G. P. Richardson. Dartmouth, NH, Aldershot. **2:** 414-434.

Fox, R. (1995). "Newstrack." Communications of the ACM **38**(8): 11-12.

Gallaugher, J. M. and Y. Wang (1999). "Network Effects and the Impact of Free Goods: An Analysis of the Web Server Market." International Journal of Electronic Commerce **3**(4): 67-88.

Garton, L., C. Haythornthwaite and B. Wellman (1997). Studying Online Social Networks. Journal of Computer-Mediated Communication **3**(1) Last Accessed: May 29, 2003 Available:http://www.ascusc.org/jcmc/vol3/issue1/garton.html

Gates, W. (1995). The Road Ahead. New York, NY, Viking Penguin.

Ghosh, R. A. (1995). The Problem with Infinity. Electric Dreams **#63 (June 19)** Last Accessed: January 14, 2003 Available:http://dxm.org/dreams/dreams63.html

Ghosh, R. A. (1998). Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet. First Monday **3 (3)** Last Accessed: January 10, 2003 Available:http://www.firstmonday.org/issues/issue3_3/ghosh/index.html

Gregory, C. (1982). Gifts and Commodities. London, Academic Press.

Grudin, J. and M. L. Markus (1997). Organizational Issues in Development and Implementation of Interactive Systems. Handbook of Human-Computer Interaction. M. G. Helander, T. K. Landauer and P. V. Prabhu. Amsterdam, Elsevier Science**:** 1457-1474.

Hagel, J. and A. Armstrong (1997). Net Gain: Expanding Markets through Virtual Communities. Boston, MA, Harvard Business School Press.

Hawkins, R. (2001). "The Economics of Free and Open Source Software." 7th International Conference of the Society for Computational Economics, New Haven, CT.

Hiltz, S. R. (1986). Online Communities: A Case Study of the Office of the Future. New York, NY, Ablex.

Jones, Q. (2000). "Time to Split, Virtually: Expanding Virtual Publics Into Vibrant Virtual Metropolises." 33rd Hawaii International Conference on System Sciences, Maui, HI.

Kahen, G., M. M. Lehman, J. F. Ramil and P. Wernick (2001). "System Dynamics Modeling of Software Evolution Processes for Policy Investigation: Approach and Example." The Journal of Systems and Software **59**: 271-281.

Katz, M. and C. Shapiro (1985). "Network Externalities, Competition and Compatibility." American Economic Review **75**(3): 424-440.

Kling, R. (1999). What is Social Informatics and Why Does It Matter? D-Lib Magazine **5 (1)** Last Accessed: January 12, 2003 Available:http://www.dlib.org/dlib/january99/kling/01kling.html

Kollock, P. (1999). The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. Communities in Cyberspace. M. Smith and P. Kollock. London, Routledge**:** 220-239.

Kvale, S. (1996). Interviews: An Introduction to Qualitative Research Interviewing. Thousand Oaks, CA, Sage.

Lazar, J. and J. Preece (1998). "Classification Schema for Online Communities." AMCIS - Americas Conference on Information Systems, Baltimore, MD.

Lin, N. (2001). Social Capital: A Theory of Socail Structure and Action. Cambridge, Cambridge University Press.

Luna, L. F. and D. L. Andersen (2002). "Using Qualitative Methods in the Conceptualization and Assessment of System Dynamics Models." 20th International System Dynamics Conference, Palermo, Italy, System Dynamics Society.

Madachy, R. (1994). A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment. Ph.D. Dissertation. Los Angeles, CA, University of Southern California.

Madachy, R. (1996). "Modelling Software Processes with System Dynamics: Current Developments." 14th International System Dynamics Conference, Cambridge, MA, System Dynamics Society.

Madachy, R. (2000). "Recent Results In Software Process Modeling." 18th International Conference of the System Dynamics Society, Bergen, Norway, System Dynamics Society.

Madachy, R. (2002). "Software Process Concurrence." Proceedings of the 20th International Conference of the System Dynamics Society, Palermo, Italy, The System Dynamics Society.

Madachy, R. J. (1996). "System Dynamics Modeling of an Inspection-Based Process." Eighteenth International Conference on Software Engineering, Berlin, Germany.

Madachy, R. J. and B. W. Boehm (2003). Software Process Modeling With System Dynamics, John Wiley & Sons.

Markus, M. L., B. Manville and C. E. Agres (2000). "What Makes a Virtual Organization Work?" Sloan Management Review **42**(1): 13-26.

Martin, R. and D. Raffo (2001). "Application of a Hybrid Process Simulation Model to a Software Development Project." The Journal of Systems and Software **59**: 237-246.

Martinez-Moyano, I. J. and G. P. Richardson (2002). "An Expert View of the System Dynamics Modeling Process: Concurrences and Divergences Searching for Best Practices in System Dynamics Modeling." Proceedings of the 20th International Conference of the System Dynamics Society, Palermo, Italy, The System Dynamics Society.

Mauss, M. (1990). The Gift: The Form and Reason for Exchange in Archaic Societies. London, Routledge.

Millen, D. R. (2000). "Community Portals and Collective Goods: Conversation Archives as an Information Resource." 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, IEEE.

Olson, G. M. and J. S. Olson (1997). Research on Computer Supported Cooperative Work. Handbook of Human-Computer Interaction. M. G. Helander, T. K. Landauer and P. V. Prabhu. Amsterdam, Elsevier Science**:** 1433-1456.

Olson, M. (1965). The Logic of Collective Action. Cambridge, MA, Harvard University Press.

O'Reilly, T. (1999). "Lessons from Open-Source Software Development." Communications of the ACM **42**(4): 33-37.

Pfahl, D., M. Klemm and G. Ruhe (2001). "A CBT Module with Integrated Simulation Component for Software Project Management Education and Training." The Journal of Systems and Software **59**: 283-298.

Preece, J. (2000). Online Communities: Designing Usability, Supporting Sociability. New York, NY, John Wiley & Sons Inc.

Preece, J. (2000). Online Cumminites: Designing Usability, Supporting Sociability. New York, NY, John Wiley & Sons Inc.

Putnam, R. (1995). "Bowling Alone: America's Declining Social Capital." Journal of Democracy **6**(1): 65-78.

Rai, V. K. and B. Mahanty (2002). "Dynamics of Schedule Pressure in Software Projects." Proceedings of the 20th International Conference of the System Dynamics Society, Palermo, Italy, The System Dynamics Society.

Raymond, E. S. (2001). The Cathedral and the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary. Sebastopol, CA, O'Reilly and Associates.

Reid, E. (1996). Communication and Community of Internet Relay Chat: Construction Communities. High Noon on the Electronic Frontier: Conceptual Issues in Cyberspace. P. Ludlow. Cambridge, MA, MIT Press**:** 397-411.

Rethemeyer, R. K. (2002). Centralization or Democratization: Assesing the Internet's Impact on Policy Networks - A Theoretical and Empirical Inquiry. Ph.D. Thesis. Boston, MA, Harvard University.

Richardson, G. P. and A. L. Pugh (1981). Introduction to System Dynamics Modeling with DYNAMO. Cambridge, MA, Productivity Press.

Rodrigues, A. G. and T. M. Williams (1997). "System Dynamics in Software Project Management: Towards the Development of a Formal Integrated Framework." European Journal of Information Systems **6**(1): 51-66.

Ruiz, M., I. Ramos and M. Toro (2001). "A Simplified Model of Software Project Dynamics." The Journal of Systems and Software **59**: 299-309.

Sandred, J. (2001). Managing Open Source Projects. New York, NY, John Wiley and Sons.

Scott, J. (2000). Social Network Analysis: A Handbook. London, Sage.

Slouka, M. (1995). War of the Worlds: Cyberspace and the High-Tech Assault on Reality. New York, NY, Basic Books.

Smith, M. J. and F. T. Conway (1997). Psychosocial Aspects of Computerized Office Work. Handbook of Human-Computer Interaction. M. G. Helander, T. K. Landauer and P. V. Prabhu. Amsterdam, Elsevier Science**:** 1497-1517.

Spector, J. M. (1995). "Using System Dynamics to Model Courseware Development: The Project Dynamics of Complex Problem-Solving." Proceedings of the 1995 ACM Symposium on Applied Computing, Nashville, TN.

Stallinger, F. and P. Gruenbacher (2001). "System Dynamics Modeling and Simulation of Collaborative Requirements Engineering." The Journal of Systems and Software **59**: 311-321.

Stanoevska-Slabeva, K. and B. F. Schmid (2001). "A Typology of Online Communties and Community Supporting Platforms." 34th Hawaii International Conference on System Sciences, Maui, HI.

Sterman, J. D. (2000). Business Dynamics : Systems Thinking and Modeling for a Complex World. Boston, MA, Irwin/McGraw-Hill.

Torvalds, L. (1999). "The Linux edge." Communications of the ACM **42**(4): 38-39.

Turoff, M. (1991). "Computer-Mediated Communication Requirements for Group Support." Journal of Organizational Computing **1**(1): 1-10.

Turoff, M. (1997). "Virtuality." Communications of the ACM **40**(9): 38-43.

Turoff, M. and S. R. Hiltz (1982). "The Electronic Journal: A Progress Report." Journal of The American Society for Information Science **33**(4).

Valloppillil, V. and E. Raymond (annotations) (1998). Halloween I Memo. Opensource.org Last Accessed: November 09, 200 2
Available:http://www.opensource.org/halloween/halloween1.php

Valloppillil, V., J. Cohen and E. Raymond (annotations) (1998). Halloween II Memo. Opensource.org Last Accessed: November 09, 2002
Available:http://www.opensource.org/halloween/halloween2.php

Wasko, M. M. and R. Teigland (2002). "The Provision of Online Public Goods: Examining Social Structure in a Network of Practice." 23th International Conference on Information Systems, Barcelona, Spain, AIS.

Wasserman, S. and K. Faust (1994). Social Network Analysis : Methods and Applications. Cambridge, Cambridge University Press.

Wellman, B. (1997). An Electronic Group is Virtually a Social Network. Culture of the Internet. S. Kiesler. Mahwah, NJ, Lawrence Erlbaum**:** 179-205.

Wellman, B. and S. D. Berkowitz, Eds. (1988). Social Structures: A Network Approach. Cambridge, Cambridge University Press.

Wellman, B. and M. Gulia (1999). Virtual Communities as Communities: Net Surfers Don't Ride Alone. Communities in Cyberspace. M. Smith and P. Kollock. London, Routledge**:** 167-194.

Williams, D. (2001). "Towards a System Dynamics Theory of Requirements Engineering Process." The 19th International Conference of the System Dynamics Society, Atlanta, Georgia, System Dynamics Society.

Williams, R. L. and J. Cothrel (2000). "Four Smart Ways to Run On-line Communities." Sloan Management Review **41**(4): 81-91.