Reward: How to Foster a Technology-Innovation Culture within a Large Organization: What You Can Learn from Start-up Companies

Bruce Jacob, *University of Maryland*

INTRODUCTION

Few firms innovate with any degree of consistency. Despite a significant body of in-depth analysis and academic research on the topic of technology innovation, most high-tech businesses are not particularly innovative, and numerous industry watchers and pundits wonder aloud how the truly innovative companies make it look so easy (for example, Burrows 2004; *Economist* 2007; Grossman 2005; Murphy 2008). The problem of how to innovate at the organizational level is clearly not solved.

Although there is more to a successful business venture than just innovation or creating useful stuff (for example, that stuff needs to be manufactured reliably, packaged attractively, marketed successfully, sold at a reasonable profit, maintained over time, protected from any public relations scandals, and so forth), nonetheless, innovation is at the very heart of high tech: it is the necessary, if not sufficient, ingredient for success. Any technology company without innovation at its core is either a commodity, a monopoly, or a vaporfirm (a seller of modern-day snake oil). The intellectual component of production that defines an organization is its technology innovation, its successful high-tech design. Accordingly, a manager in the high-tech sector must produce innovation to be successful, and it is unlikely that one will produce it without first understanding it. The focus of this chapter is to provide some of that understanding: an insight into innovation that will enable an executive to create a culture within the corporate environment that produces innovative results regularly.

The understanding begins with an acceptance that the process of innovation is non-trivial (engineering-speak for "extremely difficult"), or else everyone would be doing it. The natural conclusion of this simple observation is that traditional attitudes toward the innovative process and those who innovate must be reconsidered. To wit, either innovation is important to your organization, or it is not: if it is important, then it should be treated as such.

To occur regularly and to be sustainable, technology innovation must be nurtured: encouraged and rewarded at a level commensurate with its importance. To innovate consistently, an organization must identify who typically innovates within a given organization, understand them well enough to motivate them, and then reward them appropriately for their successes. The remainder of this chapter discusses these issues.

ACCEPT (ADMIT) THAT IT IS DIFFICULT

Recognizing the problem, understanding the difficulty of the task, is half the solution: full awareness enables the executive to see opportunities that others will miss. To become an innovator, one must identify who in the industry is innovating, comprehend what they do differently from you, and emulate their process. Most business articles and research papers focus on individual corporations and individual executives, relaying their success stories and processes. Yet these success stories are the outliers, not the rule, and studying the processes of exceptions is not likely to yield fruit easily. It is perhaps better to understand the processes at work across an entire industry—in particular, the one industry that is known for being innovative: the high-tech startup industry.

Paul Graham, co-founder of 1990s start-up Viaweb and co-developer of its software, which now powers Yahoo! Stores, depicts the inherent tension and challenge of technology innovation in a typical corporate setting:

> . . . In the right kind of business, someone who really devoted himself to work could generate

ten or even a hundred times as much wealth as an average employee. A programmer, for example, instead of chugging along maintaining and updating an existing piece of software, could write a whole new piece of software, and with it create a new source of revenue.

Companies are not set up to reward people who want to do this. You can't go to your boss and say, "I'd like to start working ten times as hard, so will you please pay me ten times as much?" For one thing, the official fiction is that you are already working as hard as you can. But a more serious problem is that the company has no way of measuring the value of your work. . . .

A company that could pay all its employees so straightforwardly [as its executives and salesmen, based upon revenue generated] would be enormously successful. Many employees would work harder if they could get paid for it. More importantly, such a company would attract people who wanted to work especially hard. It would crush its competitors. . . .

That's the real point of startups. Ideally, you are getting together with a group of other people who also want to work a lot harder, and get paid a lot more, than they would in a big company. And because startups tend to get founded by self-selecting groups of ambitious people who already know one another (at least by reputation), the level of measurement [of individual skill and contribution] is more precise than you get from smallness alone. A start-up is not merely ten people, but ten people like you. . . .

Big companies can develop technology. They just can't do it quickly. Their size makes them slow and prevents them from rewarding employees for the extraordinary effort required. So, in practice, big companies only get to develop technology in fields where large capital requirements prevent startups from competing with them, like microprocessors, power plants, or passenger aircraft. And even in those fields, they depend heavily on startups for components and ideas. . . .

(pages 96–97, 99, 101, from chapter 6, "How to Make Wealth," in *Hackers and Painters: Big Ideas from the Computer Age*) (Graham 2004)

Your job as an executive is to figure out how to turn your large company into an innovation machine—to generate new ideas and technology well and relatively often. To anyone who creates, the notion itself should smack of incredible hubris: Innovation is the creation of really useful stuff, and so to declare one's company innovation-oriented is essentially to mandate creativity. As any creative person will attest, however, creativity stubbornly refuses to be mandated, or else the term *writer's block* would hold no meaning whatsoever. So how, then, could one possibly center one's business model around something as slippery as innovation and retain any hope of staying alive?

Some Perspective

Let us get a few facts out onto the table to begin with. It would be good to bear these in mind; they are the obstacles that stand in the way of the executive or manager who desires his organization to innovate. Overcoming these obstacles is the focus of this article.

- Technology innovation is ridiculously difficult; it requires extraordinary effort, dedication, time, and focus of attention on the part of extremely talented individuals.
- Technology innovation is driven by individuals, not organizations: If you do not have extremely good people, it simply will not happen. The role of the organization is to help foster innovation, that is, to participate somewhere on the scale from not stifling it (at a bare minimum, compare to Elenkov and Manev 2005) to rewarding it at a level commensurate with its importance to the organization.
- Innovation usually, but not always, comes from engineers; "worldwide, the engineer is a key driver of technological innovation and new venture creation." (Menzel et al. 2007) Good ideas do come from all sources (Kickul and Gundry 2001), but even when an idea originates outside of engineering, it is the engineer who makes the idea work. This chapter, therefore, focuses on understanding and motivating the individual engineer.
- The majority of engineers tend to be good at only one of the following two skills: their job and promoting themselves. Some are good at neither, and it is the rare case in which an individual is good at both. Consequently, in almost every large engineering organization, in which evaluation and reward is typically done by nontechnical managers, there is a rough inverse relationship between people's salaries and their contribution to the organization's bottom line.
- On top of salary issues within engineering (that is, salaries' frequent lack of correlation to skill), a significant disparity exists between engineering salaries and those of management and sales (Salary.com 2008). The direct message to engineers is that innovation is valued by the company far less than management or marketing. Engineers are torn between the desire to innovate and the desire to become wealthy, because within a large company, it is nearly impossible to do both—that is, become wealthy while remaining an engineer in R&D.
- The issues of salary are quite well known to the engineers themselves. This generates unvocalized morale problems and causes the extremely talented individuals to leave large companies (or avoid them to begin with) and instead seek their fortune elsewhere, typically in start-up companies where, despite the 10 percent rule of thumb for start-up success, these individuals still have a far better chance of being rewarded at a level commensurate with their skills.

These may be hard truths to admit, but this is the reality in industry today. By definition, large companies tend not to innovate. The inability of (nontechnical) managers to identify and reward talent causes most employees to do what is expected of them: average, far less than spectacular, work—but without spectacular contributions, innovation

simply will not occur. The most innovative individuals often gravitate toward start-ups where, unlike the environments in large corporations, they tend to be rewarded in proportion to their accomplishments. As King et al. (2003) note, there is a constant conflict because "an employee may obtain greater financial rewards by joining or founding an entrepreneurial firm than by remaining an employee of a large firm."

Start-ups, high-tech firms in their infancy, are often run by engineers or at least have extremely talented engineers directing the entire R&D operation. Thus, start-ups tend to excel at recognizing and rewarding good design, which is why the uber-talented join them: Good engineers can identify good designs and good designers, and, more importantly, they appreciate their value, which is why start-ups reward good designs and good designers in the first place. Good design, the heart of innovation, is infinitely more valuable than money . . . you cannot buy your way into a good design just as you cannot buy the ability to innovate. Good engineers know this. The only way to buy innovation is to purchase an innovative company or hire an accomplished design team—and even then you risk losing all the innovative individuals you just hired if your corporate environment fails to reward them. As an anecdote, many purchased companies experience so-called brain drains as soon as the papers are signed; the purchasing company acquires the innovative company's name and technology but not its innovators.

To sustain innovation, to accomplish the goals of becoming and remaining a technology-innovation company, an organization must foster an environment that retains its best engineers by enabling, recognizing, and rewarding innovation and good design. By and large, the only environments that do this are those of start-up companies. This is not a fact of life, however, only a historical trend.

The Opportunity

There exists a powerful opportunity to turn this reality to one's advantage: the flip side of the trend is that any large company that does learn how to identify, retain, motivate, and reward its best engineers would position itself successfully as an innovator and would most surely dominate its industry. Witness Apple's original emergence and recent re-emergence as a technology-innovation company. Witness Google's rise; their treatment of their engineers is legendary.

So how can an organization become innovative? How can an executive successfully mandate creativity? Though one may not be able to guarantee creativity, there is much an executive can do to foster it, which is certainly an attainable goal and quite possibly "the least one can do" as the person in charge, as this is something of a sine qua non, an essential component (Menzel et al. 2007). As you will see, motivating engineers is relatively easy: give them challenging problems to solve. The real issue for the executive is that of reward—how do you convince your engineering staff that it is worth their while to go above and beyond on a daily basis?

MOTIVATION

There are two questions to address: how to motivate engineers to build innovative products, and how to keep the most innovative from leaving to join a start-up company.

A Page from the Graphing Calculator Story

First of all, what motivates engineers? The story of Apple's graphing calculator application provides good insight. The software was developed by two contractors whose projects were terminated prematurely but who nonetheless remained for months afterward, unpaid, sneaking into the facility to finish (Avitzur 2004). Here is some insight into the engineers' motivation:

> Why did Greg and I do something so ludicrous as sneaking into an eight-billion-dollar corporation to do volunteer work? Apple was having financial troubles then, so we joked that we were volunteering for a nonprofit organization. In reality, our motivation was complex. Partly, the PowerPC was an awesome machine, and we wanted to show off what could be done with it; in the Spinal Tap idiom, we said, "OK, this one goes to eleven." Partly, we were thinking of the storytelling value. Partly, it was a macho computer guy thing—we had never shipped a million copies of software before. Mostly, Greg and I felt that creating quality educational software was a public service. . . .
>
> I view the events as an experiment in subverting power structures. I had none of the traditional power over others that is inherent to the structure of corporations and bureaucracies. I had neither budget nor headcount. I answered to no one, and no one had to do anything I asked. Dozens of people collaborated spontaneously, motivated by loyalty, friendship, or the love of craftsmanship. We were hackers, creating something for the sheer joy of making it work.

The story might well be read by nonengineering types with a kind of horror; I don't know. It is certainly read by engineers as a modern Robin Hood, as an example of heroes to honor and emulate.

Engineers want to create superbly beautiful things; the less encumbered by bureaucracy, the better (compare to Menzel et al. 2007). As the story suggests, engineers will work ridiculous hours and go to absurd lengths to ensure that what they create is enviably good design; all they need is the proper motivation—and in this example, the work itself was its own motivation. Though this is an extreme example, it is indicative of the mindset of an engineer: A tough technical challenge is often the best motivation (compare to Kostoff 1999).

A Page from the Start-Up Story

How to attract and retain the top engineers is the next issue; start-ups attract top engineers but ultimately lose them as the successful start-ups transition into large corporations. This is the heart of the issue, the problem that must be understood and addressed to retain engineering talent.

Consider the normal life cycle of a company that starts out as a technology innovator and, in this case, succeeds. Normally, start-ups begin as a small group of like-minded

individuals who solve an important problem that other people want solved badly enough to pay for the solution. This is innovation. Start-ups by definition must innovate, or else they fail to thrive. Innovation is the one thing that allows them to compete with established companies.

At the outset, all tasks are handled by the individuals in this small group, but that at some point the start-up becomes successful enough to warrant additional employees to handle the noninnovative tasks considered mundane by the innovators but are nonetheless essential: answering the phones, manufacturing and packaging the product, taking orders, handling customer service, providing quality assurance, maintaining and refining the existing product line, and so forth. The company's focus, as measured by the number of staff hours spent doing the various tasks, shifts from innovating to staying profitable. As soon as there are more people in the company spending more time doing anything other than innovating, the company has changed, and the shift is palpable to anyone there since the beginning.

Meanwhile the original innovators often do one of two things: they remain focused on innovation, either by hiding themselves in their offices and developing the next-generation product, or by leaving the company to start up another.

This is merely Start-Up 101, the life cycle of nearly all high-tech start-up companies; anyone who has worked at a successful start-up recognizes the story. The life cycle of the typical innovation-based company historically includes a slowdown in innovation and a resultant brain drain, but it is exactly this historical trend that one must overcome to remain innovative as an organization.

How is an executive to reverse this trend? Primarily by maintaining the innovative atmosphere associated with the origins of the company. However simple this may sound, there is no obvious mechanism; companies the world over are scrambling for a successful recipe.

Many companies address the issue with outside people; they either collaborate with smaller start-up firms (King et al. 2003), or they bring in hired guns to do the really innovative work for a new design. Collaboration is both more likely to succeed and more risky than hiring outside help, as there is an obvious loss of control in sharing your IP with an outside firm, NDAs notwithstanding. Hiring outside help can prove successful, but it can also backfire if the existing staff feels passed over. Also, it is just as sensitive to whimsy as relying upon your own staff for innovation if you have not instituted a culture or environment conducive to innovation: The contractor's success or failure is effectively out of your control.

It would be more prudent to maximize the probability of success by creating the right environment, whether contracted help is used or not. One approach would be to emulate the risk-and-reward structure of a start-up environment directly. One can tie financial reward directly to the commercial success of an engineering team's designs, mirroring the financial reward structure of a start-up company. The risk would be that of failing to produce a winning design: the risk of not receiving a large monetary payout (more psychological defeat than a monetary risk, for example, if combined with a *skunkworks* type of competition within R&D). Such a reward structure, if implemented correctly, would appeal to the top engineers in the organization and would likely attract top engineers from other firms as well.

The point of identification is to ensure that the appropriate individuals are rewarded—that is, an organizational structure is needed to answer the questions who is good and which design is best.

IDENTIFICATION AND REWARD: HOW TO BUILD A TECHNOLOGY-INNOVATION CULTURE WITHIN AN ORGANIZATION

In a large engineering organization, the identification of the best designers and best designs is problematic. Identifying quality is difficult, especially for nontechnical managers and executives, and the larger the organization the higher the probability that engineers are evaluated by nontechnical managers or executives (for example, the middle management of Hornsby et al. (2002)). It often takes a competent engineer to adequately evaluate another engineer, but few competent engineers would rather manage than design.

For a company to attract talent, the problem must be solved: To retain good engineers, a company must reward substance, not decoration. Nothing destroys R&D morale faster than good engineers knowing they are paid less than other employees, engineering or otherwise, who don't contribute as much to the bottom line. Exacerbating the problem is the previously mentioned issue that engineers tend to be good at either their job or promoting themselves. Really good engineers often have understated or low-key personalities, wishing for their accomplishments to speak for themselves. They are thus often undervalued by their company, so the issue is how to identify those really good individuals.

One answer is to let them do it themselves; as mentioned earlier, good engineers are very good judges of design. Typically, the most talented denizens of R&D know perfectly well who is good and who is not—let them identify the cream of the crop by self-selection; let them create self-selected teams to work on projects, indirectly identifying the best designers in engineering. This emulates the start-up industry, in which innovation comes from self-selected groups of extremely competent engineers. Let the best engineers form groups that, at least on average, produce the best designs: ecce innovation.

The Proposed Structure

To repeat, the real issue for the executive is that of identification and reward—how do you identify the best of your engineering staff and convince them that it is worth their while to go above and beyond on a daily basis? How do you make them more interested in staying than leaving for a start-up company? How do you convince the exceptionally talented that it is worth their while to work for you instead of working for themselves?

When the question is put that way, it almost answers itself: You persuade talented engineers to work exceptionally hard for you by structuring their work environment

so that they are working for themselves. This is exactly the reward structure of start-up companies: Those who help start the company are given a relatively large portion of the company and thus benefit directly and significantly from their successful innovations. Since many talented engineers leave large corporate environments for start-ups, one obvious approach to enable innovation within large companies is to identify and reward engineers in exactly the same manner as start-up companies.

Envision a skunkworks-type competition within R&D for the next-generation design in which the design teams are self-selected, and the winning team is given a (small but significant) direct, proportional share of the product's revenue. This emulates the start-up environment very closely—by definition, a team must innovate to succeed, and, all else being equal, the team that does the best job is rewarded in proportion to their efforts and skills. The reward is tied directly to revenue generated by the team; this mirrors the reward structure for salesmen and executives, who get paid to produce results. This form of incentive (tying one's reward to the revenue generated) can be extremely effective at motivating people's best efforts, so it is surprising that, outside of the start-up arena, it is rarely used to motivate innovation in engineering.

Because it is important, I will repeat that last bit: Tying one's financial reward to revenue generated has proven to be extremely effective at motivating people's best efforts. It is most commonly used to motivate and reward salesmen and executives, who get paid to produce results. However effective it may be, this approach is rarely used to motivate innovation in engineering. The exception is the start-up arena, in which it is the *primary* tool for motivation, and in which innovation occurs *regularly*.

It stands to reason that giving small design teams a significant, proportional share of the revenue they generate would encourage good design and consistent innovation just as well in a large corporate environment as it does in the start-up environment.

A few of the details in this arrangement are understated but are quite important:

- **The teams must be small**. This is for two reasons; first, the productivity of a team is roughly inversely proportional to the team's size (Brooks 1995). Second, the share of revenue will be split among the entire team, so the larger the team, the smaller the individual reward. In the limiting case, one could reward the entire R&D department for a design success, but that would do little to motivate the most talented.
- **The teams must be self-selected**. This is the solution to the identification problem posed earlier: Nontechnical managers are nowhere near as adept at identifying design talent as are the engineers themselves. Faced with a nontrivial challenge, engineers will want none other than the best on their teams; all else would be dead wood slowing the team down. Note that this will work well only in organizations with some degree of transparency; engineers must be given a good idea of what everyone else is working on, otherwise there is no basis for judgment of individual skill. There do exist R&D organizations in which only management knows what the individual engineers are doing; I suspect it would be extremely difficult to introduce the proposed structure into these environments.
- **The reward must be real**. For quite a while, top executives and salesmen have been paid staggering and highly publicized bonuses for their results (and, in many cases, even their failures). It is hard to get past the surreal juxtaposition of handing a $25 million bonus to an executive for his efforts and handing a congratulatory plaque to an engineering team for their efforts, if the engineering team's efforts affect the company's bottom line just as significantly as the executive's (for example, by developing a new product and thus a new source of revenue). This incredible disparity of reward is one of the primary reasons the exceptionally talented take matters into their own hands by leaving the corporate environment to start something up on their own.
- **The reward must be tied to the product, not the company in general**. The typical reward in high tech takes the form of stock options. While this is an appropriate motivator for new hires (like welcoming someone into your family), it becomes watered down with the size and scope of the company when used to reward technical innovation on a particular product. Give an engineer a piece of the company, and the engineer will work hard, in a vague sense, to ensure the long-term success of the company. Promise an engineer a piece of his product's revenue, and the engineer will work hard on the design and development of that product to ensure its financial success. The carrot dangled dictates the resulting behavior.

This identification-and-reward structure emulates the competitive engineering environment of the start-up industry and, given an appropriate level of reward, would likely attract and retain the same uber-talented engineers as the high-tech start-up industry. The bottom line is that, to remain competitive in the modern economy, firms must now target engineering talent in the same way they have targeted executive and managerial talent in the past. Interestingly, a similar conclusion was reached by a year-long study commissioned to determine how Pittsburgh could revitalize itself (Florida 2000). Florida advises regional transformations appealing to knowledge talent, that is, high-tech engineers, in the same way Pittsburgh transformed itself in the past to attract executive and management talent:

> The Pittsburgh region has many assets and a long tradition of investing in amenities and quality of life from which to build this (proposed) agenda—its spectacular riverfronts, the cultural district, professional sports, the pioneering smoke and flood control measures of the Allegheny Conference and subsequent environmental revitalization, and the downtown renewal of the (1950s) renaissance. These measures were undertaken at least in part to improve the quality of life required to attract high-caliber executive and management talent to the region. To be successful in the new economy, the greater Pittsburgh region must build on its remarkable legacy of achievement in innovation, research and education, and revitalization to create the amenities and lifestyle required to compete effectively in the age of talent.

Also, the self-selective creative aspect mirrors the environment at Disney under Bob Iger, where a six-fold increase in revenue resulted from supporting director-driven movie creation over management-driven movie creation (*Economist,* 19 April 2008).

From an engineer's perspective, this type of identification and reward structure is ideal: It rewards talent and demands of a design team creativity married with competence—in all likelihood, for a design to succeed in this environment, the idea must be innovative, and it must work. There is no better glove to throw down on an engineer's desk.

If it strikes the reader as if the corporate entity essentially becomes a high-tech VC firm funding its own R&D staff, this is exactly what is proposed. The mechanism is not far from Hornsby's discussion of the role of middle management (2002), but Hornsby's focus is on empowering middle managers who can foster innovation, as opposed to the engineers who actually innovate. This disjunction is found throughout the many studies of *intrapreneurship* (also termed *corporate entrepreneurship* or *corporate venturing*), a concept targeting technological innovation within the corporate environment, organizational renewal and revitalization, and the creation of new business opportunities (Antoncic and Hisrich 2001; Burgelman 1984; Hornsby et al. 2002; Kuratko et al. 1990; MacMillan et al. 1986; Menzel et al., 2007; Pinchot 1985; Tidd et al. 2005). Though literature on the topic universally recognizes that engineering creativity must be fostered within the corporate environment for innovation to take place, few, if any, studies suggest that the best way to foster this creativity is to pay engineers on a par with managers, salesmen, and executives. This is surprising, given the vast number of articles on the topic (for example, several journals are devoted entirely to the study of intrapreneurship and related issues), which would seem to underscore its importance.

Similar precedents to the present proposal do exist in high tech today, for example, Microsoft, Google, and Facebook spurring third-party innovation directly through millions in developer seed funding (Mills 2008; Google 2008; Farber 2007). The only difference is that these examples show companies funding external innovation rather than internal innovation, which benefits the company indirectly rather than directly. Exploring an internally directed scenario would be worthwhile: the costs would be lower (payment is not speculative but based upon success), and the benefits more readily observed and quantified.

Moreover, this is arguably one of the only feasible solutions to the problem of identifying, retaining, motivating, and rewarding the industry's best designers—all of which is prerequisite to innovation taking place. As Brooks states, design is an individual process: "although many fine, useful software systems have been designed by committees and built by multipart projects, those software systems that have excited passionate fans are those that are the products of one or a few designing minds, great designers." (Brooks 1987; compare to Menzel et al. 2007) Brooks goes further, and though he speaks of software design, the sentiment is just as applicable to hardware design and embedded systems design:

> I think the most important single effort we can mount is to develop ways to grow great designers. No software organization can ignore this challenge. Good managers, scarce though they be, are no scarcer than good designers. Great designers and great managers are both very rare. Most organizations spend considerable effort in finding and cultivating the management prospects; I know of none that spends equal effort in finding and developing the great designers upon whom the technical excellence of the products will ultimately depend.
>
> **(Brooks 1987, reprinted in Brooks 1995)**

As Graham suggests (Graham 2004; see earlier quote), were a company to adopt such tactics, it would most likely attract engineers who wanted to work especially hard. It would crush its competitors.

OPPORTUNITIES FOR TECHNOLOGY INNOVATION

This section describes the typical development processes in hardware design, software design, and embedded systems design, with the goal of giving the nontechnical manager an idea of what is going on and pointing out a few opportunities. The discussion leans toward what is arguably the easiest problem to solve: that of improving embedded systems' reliability by importing principles of semiconductor design. The example illustrates that not all innovation produces new and better things; it is often just as valuable to produce new and better techniques.

More on Innovation and Good Design

At this point we have addressed the issues of identifying and rewarding the company's most talented and innovative engineers, but part of the original problem still exists: that is, recognizing good design. The failure of many companies to capitalize on their own innovations (numerous historical examples spring to mind, including graphical user interfaces, laser printing, computer networks) suggests that this is probably the most difficult problem to solve (compare to Candi and Saemundsson 2008).

Clearly, every industry, and every different product within that industry, will have its own set of metrics for success—qualities that make one design better than others—so it is impossible to be comprehensive here. Also, most managers and executives already believe themselves good judges of design (indeed, the term *managerial creativity* is used in many research articles in such a way as to suggest that management often believes itself to be the source of technical innovation and design).

As for good design, Graham (2004) discusses some characteristics common to particularly good examples, and at the very least, his list (which should be read slowly) will provoke thought:

Good design is simple.
Good design is timeless.
Good design solves the right problem.
Good design is suggestive.
Good design is often slightly funny.

Good design is hard.
Good design looks easy.
Good design uses symmetry.
Good design resembles nature.
Good design is redesign.
Good design can copy.
Good design is often strange.
Good design happens in chunks.
Good design is often daring.

(from chapter 9, "Taste for Makers," in *Hackers and Painters: Big Ideas from the Computer Age,* which discusses each in detail)

Among other things, good design is not the same as choosing something safe. A safe technology choice is typically a mediocre design chosen not to inspire but rather to avoid failure (Kriegesmann et al. 2005; Zahra et al. 2000). Committees typically choose safe designs. Executives typically choose safe designs. Innovators typically choose good designs. The executive who asks for innovation must be mindful of what she wishes for, on multiple levels:

- A request for innovation is a tacit acceptance of risk.
- Old habits die hard: Engineers accustomed to choosing safe designs, and management that has historically rewarded safe design choices, will continue down their well-trodden paths until led or driven elsewhere.
- "Industry best practice" is not. Best, that is. It is by definition the technological state of the art, which in high tech is merely the industry-wide status quo, because any advances are quickly adopted by all. More importantly, it is what the innovator is attempting to beat.

To help reduce the risk associated with innovation, and to retain the company's innovators, management must learn how to recognize good design (perhaps enlisting the aid of the company's top engineers), whether the particular item is chosen for production or not. This is similar in nature to corporate efforts at rewarding creative failures, rather than punishing them, so as to avoid risk aversion and thus a stifling of the innovative process (Kriegesmann et al. 2005; Menzel et al. 2007). The difference here is to learn enough about good design so that, beyond avoiding the punishment of honest creative failures, the organization can go out of its way to reward successes as well as particularly good creative failures.

Hardware versus Software

Anecdotal evidence suggests that software and hardware engineers tend not to interact well and that companies would do well to encourage collaborative efforts between the two. In particular, enabling such cross-disciplinary interaction is one of the easiest ways to make significant innovative strides into new terrain.

Traditionally, software engineers are computer scientists, and hardware engineers are electrical engineers, and so the division begins in the education system and is continued in the workplace. In academia, these topics are taught in separate departments, which are usually in separate colleges (that is, they belong to different organizational and administrative hierarchies), and the different departments never interact. In industry, they are different groups usually housed at opposite ends of the building or campus, and they never interact.

The reason they do not interact is often resentment: in both industry and academia (note: I was a software engineer in industry then went to academia and evolved into a hardware engineer), software folks hate hardware folks, and vice versa. Software people consider hardware people unibrow Neanderthals who believe clubs, sharpened rocks, and C programming to be paragons of high-tech. Hardware people consider software people flighty, self-satisfied pansies (think Harvey Korman and Andréas Voutsinas as the bickering French noblemen in *History of the World, Part I*) who complain far out of proportion to the useful work they do.

Ignoring the question of which group is right, and the fact that each group is evaluating the other against its own standards instead of evaluating the other against the other's standards (thus producing the type of cognitive disconnect and misunderstanding that occurs when a person dramatically underestimates the value and difficulty of another person's work), the important question is why do these groups fail to get along? What is going on here? Both groups are engineers; both build extremely complex systems; why is it that they do not appreciate each other's work?

Perhaps it is the nature of the work that creates the divide.

Characteristics of Software as an Engineered Product

First, let us develop a quick understanding of the problem area in which software engineers work: They build extremely large systems of interconnected functions, in which only a relative handful of the functions interact at a given time—meaning at any given point, only a fraction of the system (code) is operative.

Software engineers are held to a standard of correctness that is between 99 percent and 99.9 percent—the general rule of thumb is that any given software program has a bug every 100 to 1,000 lines of code, and this is considered an acceptable level of reliability. The truth is that the sheer complexity of these systems, and the richness of the functions and their interactions, make it extremely difficult to get a software product even to this level of correctness.

The software engineer's task is inherently creative—the software engineer is tasked to generate new concepts, new features, new behaviors . . . system-level capabilities that did not exist previously . . . and then to realize them in code. Often the hardest part is mapping these things, which rarely have words for adequate description, into existing software paradigms that inevitably lack appropriate power of expression.

Characteristics of Hardware as an Engineered Product

Let us extend this understanding to describe what hardware engineers do. Hardware engineers build extremely large systems of interconnected components, in which almost all components interact at a given time—meaning at any given point, almost all of the system is operative. If software is a gigantic, complex system of interacting functions, then hardware (for example, computer hardware) is the equivalent of one, single, incredibly enormous

function. All of it operates with itself, all the time—so, for example, if any of it is broken, the whole thing fails.

If hardware engineers were held up to the same standard of correctness as software engineers (one bug in every 1,000 lines of code), then nearly all hardware systems would be completely inoperative. Hardware simply does not work if it is only 99.9 percent correct—it doesn't even work if it is 99.999 percent correct. Semiconductor chips today have over 1 billion parts in them, and a single broken part can bring the entire system down, because all of the hardware system is being used, all the time. Forget one bug in a hundred or a thousand; a hardware engineer doesn't sleep well until the bugs are one in a million or better.

And Ne'er the Twain Shall Meet?

So that is the difference: the hardware engineer's problem domain is design reliability, design correctness; the hardware engineer is paid to do it right. The software engineer's problem domain is functionality; he is paid to do something cool and implement it acceptably well. The one is an engineer, a scientist; the other is an artist—compare to *Hackers and Painters* (Graham 2004).

Both require smarts in enormous quantities, both contribute to a company's bottom line, both are equal in value. The successful executive will manage to get each side of engineering to understand they will each benefit individually if they can work together. The key is to recognize that both classes of individuals are extremely competitive, mostly with themselves (that is, internally driven), and they thrive on solving difficult problems. Both hardware engineers and software engineers consider it far more personally rewarding to solve a supposedly impossible problem than to do just about anything else. It is mountain-climbing for the techie—a guy who scales a mountain everyone else said was impossible to climb returns with an enormous feather in his cap; ditto with engineers solving problems that nobody else could. All it takes to get a really good engineer to work on a problem is for him to see how difficult it is: Technical challenges to engineers are catnip to cats or bug-zappers to mosquitoes.

Importantly, technical challenges also happen to be significantly more engrossing to a hardware or software engineer than bashing the software and hardware departments.

Bottom Line

Software and hardware engineers will work together famously if the problem is hard and the reward is significant. Why this should be the least bit interesting to an executive is that in most organizations hardware and software engineers do not work together famously. Thus, it stands to reason that most organizations are not living up to their potential for technology innovation, because the boundary between software and hardware is less well understood than are the fields of software and hardware by themselves. Entire industries have emerged when software and hardware are designed together to play off each other's strengths, enabling applications that would not have succeeded if done in hardware or software alone, for example, graphics acceleration, MP3 players, digital imaging (in particular the merging of cameras and camcorders), software-defined radio, and so forth.

As before, the bottom line is that any large company that does solve this problem—that learns how to get its software and hardware groups to interact creatively—would position itself successfully as an innovator and would most surely dominate its industry.

A Tale of Three Design Flows

Both hardware and software are extremely powerful technologies, but both still leave much to be desired, creating a significant opportunity for innovation. In particular, software and hardware engineers can learn from each other—there is plenty of room for software to be more reliable and for hardware to be more exotic. The engineering challenge is to do this without sacrificing the beneficial characteristics.

VLSI Design

To begin with, consider VLSI design, the creation of semiconductor parts. Jan du Preez, former president of Infineon Technologies North America, stated quite flatly that "semiconductor design is possibly the most complex thing that humans do" (du Preez 2002). VLSI manufacturing is relatively expensive: A mask set for a cutting-edge process technology typically runs in the millions of dollars. Any design revision requires new masks, potentially a full set, so this is not a technology conducive to an iterative design-build-test-redesign development cycle. Designers do not build a chip to test their designs; they build the chip only when they are certain it will work. A design must work the first time around or, worst case, the second time around—more than that, and the project is scrapped or the company goes out of business.

The methodology for VLSI design enables such tight tolerances on correctness. The design flow is characterized by strict design rules; the development tools enable a verifiable physical design, meaning that one can verify at the design stage, using CAD tools, whether or not the physical implementation will work. One need not build a chip to verify the chip's design.

A typical design flow is illustrated in Figure 69.1. The engineer begins with a very high-level representation of the final chip: A behavioral design that looks very much like a piece of software. This specification indicates what the chip is supposed to do and how it is supposed to behave, as opposed to what circuits to use. As Brooks asserts (1987), the hardest part of designing a product is "arriving at a complete and consistent specification, and much of the essence of building (the product) is in fact the debugging of the specification." Hardware design thus begins with a behavioral design, a form of specification that is far easier to develop and debug than is hardware itself.

The behavioral design is transformed by CAD tools into a structural design. The transformation process is called synthesis; its main benefit is the saving of valuable engineering time: The tools provide a reasonable first cut at a low level design, one that the engineer will further optimize by hand. The optimized structural design is run through another CAD tool that replaces the logic-level structures with equivalent physical layouts taken from a library, places those structures on the chip, and routes the signals that connect them. What is produced is a physical design

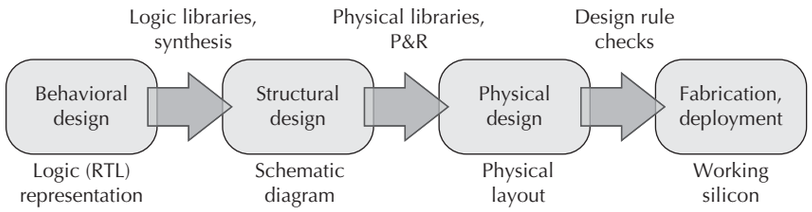

Figure 69.1: The Typical Flow in Digital Semiconductor Design and Development

that looks on the computer screen exactly like the chip that will be fabricated. This design is also optimized by hand.

Before the physical design can be fabricated, it must pass a number of tests (for example, electrical checks, design rule checks, and so forth) that ensure that the part as built will faithfully reproduce the design. These tests ensure that each level of design (behavioral, structural, physical) corresponds exactly to the other, that whatever physical connections are implied at one level of design are implemented in another, and that all electrical connections existing at the physical level are anticipated by the higher-level designs. The end result of all the testing is a reasonable guarantee that, when the part is fabricated, any mistakes found in the implementation will be the result of a faulty specification (that is, behavioral design) and not the fault of fabrication. Each level of the design (behavioral, structural, physical) can be tested thoroughly with CAD tools, and as each successive design approaches more closely the final physical form, so, too, the tests applied to the design mimic more closely the tests one would perform on an actual chip, the more realistic and convincing the results, and the more confident the engineer that the design will work when fabricated.

Note that the reason digital VLSI design is verifiable is because of its limited palette: A digital designer can use wires and transistors, and that is all. This was considered extremely limiting when the concept was introduced in the late 1970s by Carver Mead and Lynn Conway (Mead and Conway 1979), as designers had been comfortable designing digital functions using all manner of devices, processing steps, and process technologies that were incompatible with each other and all essentially analog, not digital, in regard to their analysis. The limited palette, coupled with strict design rules, meant one could relegate design analysis and verification to CAD tools, thereby enabling significantly more sophisticated designs. Notably, it was only after the introduction of VLSI design rules that the semiconductor industry began to develop truly complex designs, ushering in the heyday of exponential growth.

Software Design

The second design flow, that of software development, is shown in Figure 69.2. The application is specified at a high level: a functional level—that is, what the application should do, as opposed to how it should be built. The specification is usually descriptive prose or pseudocode, and the components that make up the application are identified and assigned to the various designers and design teams (compare to Cusumano and Selby 1997).

The components are designed and developed by hand, and they are tested in isolation before being integrated

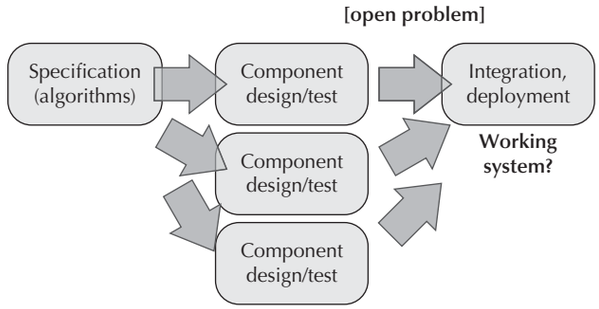

Figure 69.2: The Typical Flow in Software Design and Development

into the main code base. This presents problems for components developed by different people or different organizations, because it heightens the importance of correctly and unambiguously specifying the interfaces between components. Witness the loss of the 1999 Mars probe (Oberg 1999), in which the technical failure was due to a mismatch of measurement units used by different development teams in different countries. Microsoft's solution to the problem is to attempt to catch these problems early by enforcing the frequent integration of components into the main code base while in development, thereby increasing the likelihood of discovering any incompatibilities (Cusumano and Selby 1997).

Despite significant attention paid to the open problem of software validation, discovering software errors is still largely done by hand. What makes improving software development challenging is that, unlike hardware development, software is not particularly amenable to CAD-tool support (Brooks 1987), largely due to software's nature, as described before: Truly innovative software significantly pushes the boundaries of what code is considered capable of doing. An innovative application will defy English prose to adequately describe what it does; it will similarly defy existing code to match its functional abilities. By definition, this implies that a truly innovative software application will exceed the abilities of any existing automated testing mechanism. Consequently, software testing is necessarily a human-directed activity today. Improving the reliability of software is still a long-term goal and a heavily researched area.

Embedded Systems Design

To compare with VLSI and software, the design flow for embedded systems is shown in Figure 69.3. This is

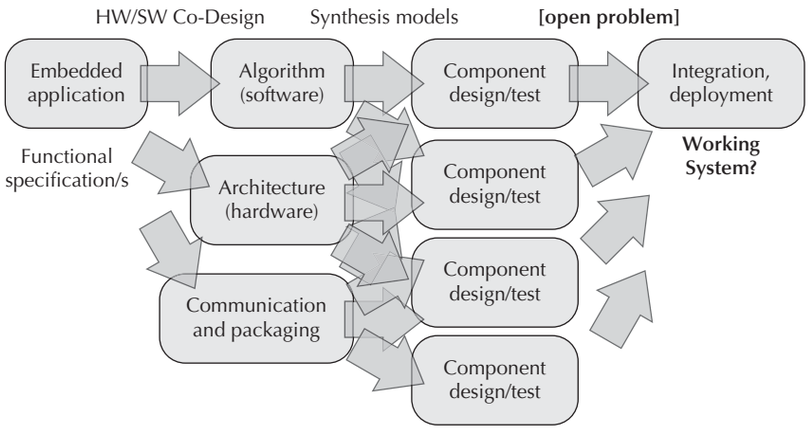

Figure 69.3: The Typical Flow in Embedded Systems Design and Development

described in a bit more detail, because, unlike software design, embedded systems design is amenable to CAD support, largely because embedded systems tend to have far simpler functional specifications. Despite being partly software, embedded systems are more amenable to CAD analysis than general-purpose software because the expected behavior of an embedded system is usually simpler (smaller set of possible inputs, less internal state) and better suited to formal specification than, say, a desktop operating system or word processor.

An embedded application is specified at a functional level using methods that range from the informal to the formal: for example, prose descriptions, block diagrams, pseudocode, state machines, mathematical expressions, MatLab code, UML diagrams, and so forth. The first step partitions the high-level application into constituent pieces such as hardware and software. An entire field of study is devoted to this partitioning problem, called hardware-software co-design, in which the partitioning is automated. An additional component often overlooked is the physical packaging of the components and mechanisms that allow the components to interact (for example, circuit boards, multi-chip modules, 3-D stacking, computer enclosures, switches, wired and wireless networks and their protocols, and so forth). The choice of packaging and communication mechanisms often dictates whether the desired functions can be implemented.

After choosing the components to constitute the application, engineers build and test each non-COTS component—note that the behavior of any given component (and thus its design) may be influenced by any or all of the application's algorithm, architecture, communication networks, and packaging. Example components: software modules must be written, ASICs developed and fabricated (using the VLSI design flow described earlier), actuators and sensors developed, metal/wood/plastic parts machined and assembled, circuit boards manufactured and assembled. Synthesis models for some of these components exist—meaning that, given a high-level specification, a CAD tool can produce a low-level design for the desired part (as in the VLSI steps before)—but these synthesis models are not nearly as well developed as those for semiconductor design, so much of this level of development is done by hand.

An important fact is that, like software, these individual components are designed, built, and tested in isolation from each other. For example, in the design of automobile control networks, it is not unusual for each component (power train, spark-plug ignition, antilock brake system, stability augmentation system) to be designed and built by a separate group, division, or even company whose sole focus is that one component.

When ready, the components are integrated into a complete system, the first testable prototype. In many cases, this is the first time a full system test can be performed, so it is no surprise that it is an open question whether or not the system will indeed work when turned on. Whereas the VLSI design flow is characterized by strict design rules that guarantee a degree of system-level verification, the embedded-systems design flow is characterized by a complete lack of design rules and the use of ad hoc methods (at best) for system-level verification.

To Do It Better

The embedded systems design flow is an obvious candidate for technology innovation. The integration of complex heterogeneous components is a stumbling block for system design: One cannot guarantee the correctness of a system by verifying components in isolation—the entire system must be verified as a whole. Yet component-level design and verification is typical practice in building modern-day embedded systems; system-level testing is frequently done only when the already-built components are assembled into a working system. The practice leaves latent design bugs that are too subtle to be uncovered in a lab setting and that manifest themselves after product deployment. For example, the software loops in an automobile control system are usually designed and tested in isolation, but a recent trend in the industry is to time-share these loops on a single microprocessor. In this scenario, the implicit assumption of independence is no longer valid, and, now that there can be direct interaction between the loops, unintended (and untested) behaviors can arise. Another example: A recently proposed redesign of the Blackhawk helicopter replaced the existing electrical wiring of the controller-area network with a fiber-optic channel. Though

the new optical network increased bandwidth tremendously, the increase in packet latency that was required by the optical network-interface hardware made it impossible to design a stability and control augmentation system that would meet the desired specifications.

Because they tend to be far simpler than general-purpose software systems, embedded systems are relatively low-hanging fruit in the sense of applying CAD tools to their design-time verification. The good news is that some industry leaders sense this and have proven the technology by designing their complex embedded systems entirely in software—for example Fiat's rapid development of its 500 model (*Economist* 26 April, 2008). Such a CAD-oriented method is far from standard in most industries, and so any company that can adopt this technique stands to improve its time to market and reduce design bugs discovered late in the development cycle.

EMERGENT PARADIGMS: MANUFACTURING AS A SERVICE, DESIGN AS AN END PRODUCT

Recent trends would seem a boon for companies that want to focus more attention on technology innovation; they enable a company to spend less capital on infrastructure and less attention on manufacturing. Rather than exploiting these trends merely to cut costs, an organization should instead spend the freed capital and attention on R&D, innovation, and quality assurance.

In every era, manufacturing has been pushed to the fringes of society, away from living and communal spaces. Factories have been moved away from dense urban areas; manufacturing has been exported to the Third World. At the same time, design has never been pushed away; design has never been exported, except to the detriment of the exporter. Design is the core intellectual exercise that can define a company, an industry, a culture, a nation. Assuming you consider yourself an innovator, if you give design over to a third party, you have given away your reason for being—anything else you bring to the table can be bought; all else but design is a commodity.

The interesting result is manufacturing as a service, a phenomenon increasing in both visibility and popularity. Traditionally, capital expenses are required to enter a manufacturing industry: Before you can build widgets, you must first build a factory that can build widgets. One consequence of the Internet and the international competition it has enabled (Friedman 2005) is the number of plants offering custom manufacturing at wholesale prices. For example, circuit-board manufacturers accept customer designs uploaded to the Web; they manufacture the boards and perform assembly as well—that is, given bills of materials they will return a fabbed and completely populated board and can obtain all COTS parts directly from distributors. Some of the world's largest semiconductor companies, such as Taiwan Semiconductor (TSMC), focus on building other people's designs, not their own. Similar services are available for CNC routing, plastics, metalworking, final assembly, and so forth.

Because of this phenomenon, the traditional capital-expense barrier to entry no longer exists: One can now manufacture a new product line that is manufactured entirely by third parties. Certainly the per-unit cost in this scenario is higher, but the capital start-up cost is gone; that means the cost exposure of testing the waters with a new product, even in a new industry, is effectively nil. It is only an issue for experiments that fail so miserably and so publicly that negative PR hurts the company's other products.

This now enables a focus on design as an end product in and of itself: Anything your organization can design, someone out there can and will manufacture for you, and the Net has not only enabled this but also simplified tremendously the search for willing manufacturers.

The obvious conclusion is that this empowers start-up companies to compete in industries that were previously out of reach. Numerous examples can be found in the semiconductor industry now that design firms need not spend the several billion dollars required to build a fab: Silicon Valley is bursting with companies offering intellectual property, rather than hard goods, as their value added.

The less obvious conclusion is that it also enables larger companies to try the same trick: to explore new products or even new industries outside the company's core area that would have been deemed too risky had the start-up costs included capital expenses. With deeper pockets, a large company should be able to cast a wider net than a start-up—in effect, to spread the risk and increase the likelihood of success by emulating the effect of several start-ups in several different areas or by trying several different approaches in the same area (compare to MacMillan et al. 1986).

Again, if this sounds like the approach that venture capitalists take, it is exactly that, and the strategy behind the tactics is exactly the same. Many innovators, after starting up one or more successful high-tech companies, move out of the start-up industry and into the VC industry; this is perfectly logical, because a nest egg (for example, capital earned from the sale of the innovator's start-up) is more likely to grow if it is put into more than one basket.

Capital Freed to Take Multiple Shots

The lesson to take away is simple but powerful. Innovation is like throwing darts, in that each shot may or may not hit the mark, even for an accomplished thrower. The most reliable way to make a bull's-eye is to take multiple attempts.

Similarly, the most reliable way to make money in the start-up industry, given a fixed amount of attention, time, and energy, is not to start up a company but to fund start-up companies.

Given that, an executive seeking technological innovation will maximize his chances of success by becoming a de facto VC—by treating the company's engineering staff as a collection of would-be startup companies, effectively enabling the executive to take multiple shots at the bull's-eye. The paradigms emerging in modern business support such an approach on two distinct levels:

- Exploiting the manufacturing services of third parties reduces the start-up costs associated with a new endeavor and thus allows a large company to afford multiple attempts at innovating in new areas.

- Offloading part of one's manufacturing onto third parties allows the focus of a company to shift toward design, a much higher-margin (higher risk, higher reward) activity. Note well the words etched on the back of Apple products: Designed by Apple in California.

The trick, as always, will be to manage well the innovative processes.

CONCLUSION: INNOVATION AND THE LION'S SHARE OF REWARD

Engineers do not innovate simply because they are told to; often, they will innovate despite being told not to. For a good engineer, innovation is the reason for being; good ideas come about in the same way and for the same reason that paintings or sculptures come from a good artist and words come from a good writer—because they must.

It should be no wonder, then, that start-up companies, when taken as an industry, are more consistently innovative than larger companies: Start-ups provide creative environments relatively unencumbered by bureaucracy, and they tend to reward the innovative engineer at a level equal to or above that of a nontechnical manager. Indeed, in most start-ups, there is little management within engineering, and what little exists is typically technical (engineers). This represents the ideal environment for nurturing creativity, and it tends to attract some of the most innovative designers in the high-tech industry, often away from larger corporations.

Large firms are desperately trying to close the innovation gap, usually by bringing in talent from outside—by hiring designers or design teams, purchasing or partnering with start-up companies, and so forth—but this is clearly unsustainable without relatively high turnover. Numerous academic studies and business articles on innovation and intrapreneurship try to close the gap by enabling management-level creativity or fostering a positive, creative development environment within the organization. Despite this attention, the problem remains.

The heart of the matter would seem to be relatively simple: reward. Many firms talk loud and proud about the importance of innovation but fail to reward those who innovate. The lion's share of corporate reward typically goes to the executive managers who enable innovation and not the engineers who turn ideas into reality. When actions fail to support a company's words, talented engineers will turn to the one industry that puts its money where its mouth is: the high-tech start-up industry—the one industry that rewards individual engineers significantly for their innovations, and, not coincidentally, the one industry that consistently innovates.

GLOSSARY

Brain Drain: The phenomenon of design talent leaving a company en masse, usually precipitated by a specific incident such as a (forthcoming) change in management. The connotation is that, afterward, the company's primary innovators are gone, diminishing the company's capacity for technology innovation.

CAD Tool: Software used to perform design tasks that can be automated. Because the computer can perform tasks much more quickly and accurately than a human (in particular, performing the same mundane task repeatedly on billions of different inputs), anything that can be relegated to a computer in the design process should be. This is not trivial, given that the bulk of design is inherently creative; thus, the majority of CAD-supported activities center around verification.

Design (process): The act of deciding what to build and how to build it. Most products represent one choice out of an infinite space of possibilities: the act of deciding on that one choice out of the entire space, and more importantly, justifying the decision is the design process.

Design (produced item): A high-level description, or specification, of the end product. Each product domain defines its own specification format: for example, a design for a house is a blueprint; a design for a circuit is a schematic diagram; a design for a software application could be a rough sketch of a screen shot and a list of behaviors attached to each button.

Development: The act of producing what is specified in a given design. Analogy: building a house on the basis of its blueprints.

Hired Gun: Somewhat derogatory term for contractor (temporary employee) hired for design or development. The connotation is that the contractor is being hired to do interesting work that existing staff cannot do (either because of existing commitments or technical expertise). If the work to be done is not interesting (that is, grunt work), the contractor would be called something other than "hired gun," such as "warm body" (connotation: all that is required is a pulse).

Skunkworks: A subgroup within a design or engineering organization empowered to design the next-generation product, usually in secret or separately from the rest of the organization. Sometimes, multiple teams can be working on the same goal, each using a different approach and unaware of the de facto competition. The term comes from the alias for Lockheed Martin's advanced development programs.

Specification: Sometimes used as synonym for *design* (produced item). When a distinction is drawn between the two, the design is a concept, and the specification is the formal description of that concept.

Synthesis: The automation of (a portion of) the development process. See *CAD tool:* If anything can be relegated to a computer, it should be. Since the development process is often the act of producing ever-more-specific design specifications, each one based upon the previous spec and replacing descriptions with low-level implementations, it is sometimes possible to automate the translation from description to implementation.

Verification: The act of testing an implementation to ensure that it corresponds to its high-level design specification. This is an extremely difficult task, as any complex system can have many trillions of possible inputs and many trillions of states, and the testing of each input with each state could take anywhere from nanoseconds to milliseconds, or longer if human input

is required. Even at nanosecond speeds, full verification of such a product would take years, and so, significant effort has gone into reducing the number of inputs and states that must be checked to ensure that a product works as advertised.

Wealth: A discussion from Graham (2004):

> If you want to create wealth, it will help to understand what it is. Wealth is not the same thing as money. Wealth is as old as human history. Far older, in fact; ants have wealth. Money is a comparatively recent invention.
>
> Wealth is the fundamental thing. Wealth is stuff we want: food, clothes, houses, cars, gadgets, travel to interesting places, and so on. You can have wealth without having money. If you had a magic machine that could on command make you a car or cook you dinner or do your laundry, or do anything else you wanted, you wouldn't need money. Whereas if you were in the middle of Antarctica, where there is nothing to buy, it wouldn't matter how much money you had.
>
> Wealth is what you want, not money. But if wealth is the important thing, why does everyone talk about making money? It is a kind of shorthand: money is a way of moving wealth, and in practice they are usually interchangeable. But they are not the same thing, and unless you plan to get rich by counterfeiting, talking about *making money* can make it harder to understand how to make money.

CROSS REFERENCES

Competitive Strategies in Technology Management; Economics of Technological Innovation; New Product Development; Innovation Management in Large Corporations; Global Manufacturing: An Overview of Modern Collaborative; Technologies and Product Architecture; Outsourcing and Offshoring in a Globalizing World: A Review of Trends and Perspectives

REFERENCES AND SUGGESTED READINGS

Antoncic, B., and R. D. Hisrich. 2001. Intrapreneurship: Construct refinement and cross-cultural validation. *Journal of Business Venturing* 16(5):495–527.

Avitzur, R. 2004. The graphing calculator story. www.pacifict.com/Story.

Brooks, F. 1987. No silver bullet—essence and accidents of software engineering. *IEEE Computer* 20(4):10–19.

Brooks, F. 1995. *The mythical man-month.* Reading MA: Addison-Wesley.

Burgelman, R. A. 1984. Designs for corporate entrepreneurship in established firms. *California Management Review* 26(3):155–166.

Burrows, P. 2004. The seed of Apple's innovation. *BusinessWeek,* voices of the innovators, October 12. www.businessweek.com/bwdaily/dnflash/oct2004/nf20041012_4018_db083.htm.

Candi, M., and R. Saemundsson. 2008. Oil in water? Explaining differences in aesthetic design emphasis in new technology-based firms. *Technovation* 28(7): 464–471.

Chapman, G. 2000. Cool counts in luring hot techies. *Los Angeles Times,* July 31. http://articles.latimes.com/2000/jul/31/business/fi-61927.

Cusumano, M. A., and R. W. Selby. 1997. How Microsoft builds software. *Communications of the ACM* 40(6): 53–61.

du Preez. J. 2002. Personal communication.

Economist. 2007. Lessons from Apple: What other companies can learn from California's master of innovation. *The Economist,* June 7. www.economist.com/opinion/displaystory.cfm?story_id=9302662.

Economist. 2008. Disney: Magic restored. *The Economist,* April 19.

Economist. 2008. Fiat: Rebirth of a carmaker. *The Economist,* April 26.

Elenkov, D. S., and I. M. Manev. 2005. Top management leadership and influence on innovation: The role of sociocultural context. *Journal of Management* 31(3):381–402.

Farber, D. 2007. Facebook investors start $10 million fund for developers. ZDNet blogs: Between the lines, September 17. http://blogs.zdnet.com/BTL/?p=6286.

Florida, R. 2000. Competing in the age of talent: Environment, amenities, and the new economy. Carnegie Mellon University. A report prepared for the R. K. Mellon Foundation, Heinz Endowments, and sustainable Pittsburgh. www.faulkner.edu/admin/websites/cemerson/documents/AgeofTalent1.pdf.

Ford, C. M., and D. A. Gioia. 2000. Factors influencing creativity in the domain of managerial decision making. *Journal of Management* 26(4):705–732.

Friedman, T. 2005. *The world is flat: A brief history of the twenty-first century.* New York: Farrar, Straus and Giroux.

Google. 2008. Android developer challenge. http://code.google.com/android/adc.html.

Graham, P. 2004. *Hackers and painters: Big ideas from the computer age.* Sebastopol, CA: O'Reilly Media.

Grossman, L. 2005. How Apple does it. Time.com, October 16. www.time.com/time/magazine/article/0,9171,1118384,00.html.

Hoffman, A. 2006. Why big high-tech companies are losing the talent war. VentureBeat, November 22. http://venturebeat.com/2006/11/22/why-big-high-tech-companies-are-losing-the-talent-war/.

Hornsby, J. S., D. F. Kuratko, and S. A. Zahra. 2002. Middle managers' perception of the internal environment for corporate entrepreneurship: Assessing a measurement scale. *Journal of Business Venturing* 17(3):253–273.

Kickul, J., and L. K. Gundry. 2001. Breaking through boundaries for organizational innovation: New managerial roles and practices in e-commerce firms. *Journal of Management* 27(3):347–361.

King, D. R., J. G. Covin, and W. H. Hegarty. 2003. Complementary resources and the exploitation of technological innovations. *Journal of Management* 29(4):589–606.

Kostoff, R. N. 1999. Science and technology innovation. *Technovation* 19(10):593–604.

Kriegesmann, B., T. Kley, and M. G. Schwering. 2005. Creative errors and heroic failures: Capturing their innovative potential. *Journal of Business Strategy* 26(3): 57–64.

Kuratko, D. F., R. V. Montagno, and J. S. Hornsby. 1990. Developing an intrapreneurial assessment instrument for an effective corporate entrepreneurial environment. *Strategic Management Journal* 11:49–58.

MacMillan, I., C. Z. Block, and P. N. S. Narashima. 1986. Corporate venturing: Alternatives, obstacles encountered, and experience effects. *Journal of Business Venturing* 1(2):177–191.

Mead, C., and L. Conway. 1979. *Introduction to VLSI systems*. Reading MA: Addison-Wesley.

Menzel, H., C. I. Aaltio, and J. M. Ulijn. 2007. On the way to creativity: Engineers as intrapreneurs in organizations. *Technovation* 27(12):732–743.

Mills, E. 2008. Microsoft eyes health care app developers with $3 million fund. C/NET News.com, February 24. www.news.com/8301-10784_3-9877656-7.html.

Murphy, C. 2008. America's most admired companies 2008: 10 most admired for innovation. CNNMoney.com. http://money.cnn.com/galleries/2008/fortune/0803/gallery.innovation.fortune/index.html.

Oberg, J. 1999. Why the Mars probe went off course. *IEEE Spectrum* 36(12):34–39.

Pinchot, G. 1985. *Intrapreneuring: Why you do not have to leave the corporation to become an entrepreneur.* New York: Harper and Row.

Salary.com. 2008. Salary wizard (for example, compare *chief engineering executive* to *chief marketing executive*).

Tidd, J., J. Bessant, and K. Pavitt, eds. 2005. *Managing innovation: Integrating technological, market and organizational change* 3d ed. Chichester, UK: John Wiley & Sons.

Zahra, S. A., D. O. Neubaum, and M. Huse. 2000. Entrepreneurship in medium-size companies: Exploring the effects of ownership and governance systems. *Journal of Management* 26(5):947–976.