Harvard Business Review

Six Myths of Product Development

by Stefan Thomke and Donald Reinertsen

Artwork: **Ricky Allman, *Undertable,*** 2011, acrylic on canvas, 72" x 48"

Most product-development managers are always struggling to bring in projects on time and on budget. They never have enough resources to get the job done, and their bosses demand predictable schedules and deliverables. So the managers push their teams to be more parsimonious, to write more-detailed plans, and to minimize schedule variations and waste. But that approach, which may work well in turning around underperforming factories, can actually hurt product-development efforts.

Although many companies treat product development as if it were similar to manufacturing, the two are profoundly different. In the world of manufacturing physical objects, tasks are repetitive, activities are reasonably predictable, and the items being created can be in only one place at a time. In product development many tasks are unique, project requirements constantly change, and the output—thanks, in part, to the widespread use of advanced computer-aided design and simulation and the incorporation of software in physical products—is information, which can reside in multiple places at the same time.

The failure to appreciate those critical differences has given rise to several fallacies that undermine the planning, execution, and evaluation of product development projects. Together, we have spent more than 50 years studying and advising companies on product-development efforts, and we have encountered these misconceptions—as well as others that arise for different reasons—in a wide range of industries, including semiconductors, autos, consumer electronics, medical devices, software, and financial services. In this article we'll expose them and offer ways to overcome the problems they create.

Fallacy 1: High utilization of resources will improve performance.

In both our research and our consulting work, we've seen that the vast majority of companies strive to fully employ their product-development resources. (One of us, Donald, through surveys conducted in executive courses at the California Institute of Technology, has found that the average product-development manager keeps capacity utilization above 98%.) The logic seems obvious: Projects take longer when people are not working 100% of the time—and therefore, a busy development organization will be faster and more efficient than one that is not as good at utilizing its people.

But in practice that logic doesn't hold up. We have seen that projects' speed, efficiency, and output quality inevitably decrease

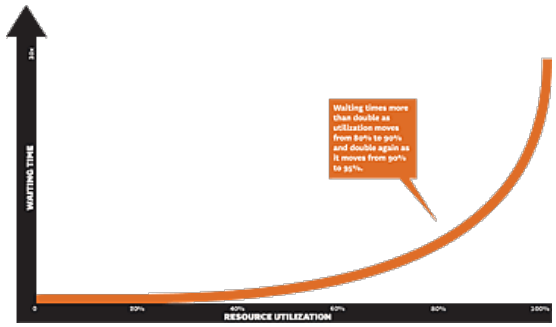
when managers completely fill the plates of their product-development employees—no matter how skilled those managers may be. High utilization has serious negative side effects, which managers underestimate for three reasons:

They don't take into full account the intrinsic variability of development work. Many aspects of product development are unpredictable: when projects will arrive, what individual tasks they'll require, and how long it will take workers who've never tackled such tasks before to do them. Companies, however, are most familiar with repetitive processes like manufacturing and transaction processing, where the work doesn't change much and surprises are few and far between. Such processes behave in an orderly manner as the utilization of resources increases. Add 5% more work, and it will take 5% more time to complete.

Processes with high variability behave very differently. As utilization increases, delays lengthen dramatically. (See the exhibit "High Utilization Leads to Delays.") Add 5% more work, and completing it may take 100% longer. But few people understand this effect. In our experience with hundreds of product-development teams, we have found that most were significantly overcommitted. To complete all projects on time and on budget, some organizations we worked with would have needed at least 50% more resources than they had.

High Utilization Leads to Delays

The curve below is calculated using Queuing Theory, the mathematical study of waiting lines. It shows that with variable processes, the amount of time projects spend on hold, waiting to be worked on, rises steeply as utilization of resources increases. Though the curve changes slightly depending on the project work, it always turns sharply upward as utilization nears 100%.



It is true that some variability is the result of a lack of discipline, and that some product-development tasks (like designing components for an airplane prototype or performing clinical trials) include more-repetitive work. But even if some of the work is predictable, when it's combined with other unpredictable work, you will see queuing problems.

They don't understand how queues affect economic performance. High utilization of resources inevitably creates queues of projects. When partially completed work sits idle, waiting for capacity to become available, the duration of the overall project will grow. Queues also delay feedback, causing developers to follow unproductive paths longer. They make it hard for companies to adjust to evolving market needs and to detect weaknesses in their product before it's too late. Ironically, these problems are precisely the ones that managers think high utilization will allow their teams to avoid.

Even when managers know that they're creating queues, they rarely realize the economic cost. Although that cost can be quantified, we've found that the vast majority of companies don't calculate it. Managers need to weigh queue costs against the costs of underutilized capacity in order to strike the right balance.

In product development, work-in-process inventory is predominantly invisible. Manufacturing queues consist of physical things, and when inventory in a factory doubles, it's obvious. That's not the case in product development, where inventory largely consists of information, such as design documentation, test procedures and results, and instructions for building prototypes. When inventory doubles in an engineering process, there are no physical signs. Moreover, because accounting standards require most R&D inventory to be carried at zero value, financial statements give no indication of serious inventory excesses in product development.

It is very difficult to fight a problem that you can't see or measure. Consider the situation at a major pharmaceutical firm. Several years ago its newly appointed head of drug discovery faced a managerial dilemma. Like other senior executives who run large R&D organizations, he was trying to find ways to make his scientists more innovative. He wanted them to experiment more with new chemical compounds that could generate promising new drugs and, at the same time, to eliminate unpromising candidates as early as possible. Experiments with living organisms, however, were the responsibility of animal testing, a department that was not under his control and was run as a cost center. It was evaluated by how efficiently it used testing resources, which

naturally led to high utilization. Consequently, the scientists in drug discovery had to wait three to four months for the results of tests that took a little more than a week to perform. The "well-managed" testing organization impeded the discovery unit's progress.

The obvious solution to such problems is to provide a capacity buffer in processes that are highly variable. Some companies have long understood this. For decades, 3M has scheduled product developers at 85% of their capacity. And Google is famous for its "20% time" (allowing engineers to work one day a week on anything they want—a practice that means extra capacity is available if a project falls behind schedule). However, in our experience this kind of solution is quite hard to implement. As we will discuss, few organizations can resist the temptation to use every last bit of available capacity. Managers reflexively start more work whenever they see idle time.

But there are other viable solutions:

Change the management-control systems. For the pharmaceutical company, this might involve taking steps to align the objectives of the animal-testing unit with those of the discovery unit. The company could, for example, reward animal testing for prompt responses (measuring time from request to completion of test) rather than resource utilization.

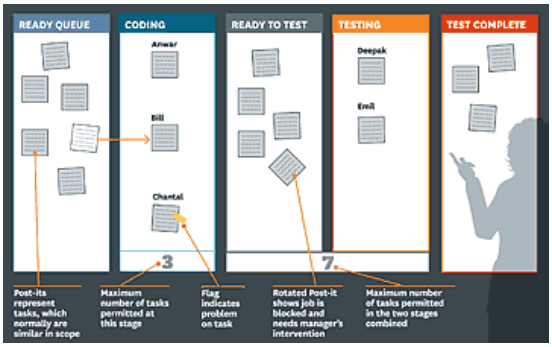
Selectively increase capacity. Adding extra resources to the areas where the utilization rates are 70% or higher can significantly reduce waiting time. If the pharmaceutical company did this in animal testing, it would obtain feedback on new chemical compounds far faster. In settings where testing is conducted with computer modeling and simulation, increasing capacity is often relatively inexpensive, since it just involves buying additional computer equipment and software licenses.

Limit the number of active projects. If the pharmaceutical firm couldn't increase animal testing's capacity, it could still lower the utilization rate by reducing the number of active projects exploring new chemical compounds. The discipline of putting a hard limit on what goes into a product-development pipeline often results in sharper focus and clearer priorities.

Make the work-in-process inventory easier to see. One method is to use visual control boards. These can take a number of forms, but the key is to have some sort of physical token, such as a Post-it note, represent the development work (see the exhibit "Typical Work-in-Process Control Board"). A control board should display all active work and show what state each part of the project is in. It should be at the center of the team's management process. Teams can hold 15-minute daily stand-up meetings around such boards to coordinate efforts and keep work moving.

Typical Work-in-Process Control Board

Control boards make invisible work visible by showing the precise stage that each work item is in. In designing the boards, most teams limit the number of tasks at each stage to prevent delays. This simple board contains features that might be found on a software project that involves a team composed of six to 10 people.



Fallacy 2: Processing work in large batches improves the economics of the development process.
A second cause of queues in product development is batch size. Let's say a new product is composed of 200 components. You could choose to design and build all 200 parts before you test any of them. If you instead designed and built only 20 components before you began testing, the batch size would be 90% smaller. That would have a profound effect on queue time, because the average queue in a process is directly proportional to batch size.

The reduction of batch sizes is a critical principle of lean manufacturing. Small batches allow manufacturers to slash work in process and accelerate feedback, which, in turn, improves cycle times, quality, and efficiency. Small batches have even greater utility in product development, but few developers realize the power of this method.

One reason is the nature of their work flow. Again, because the information they're producing is mostly invisible to them, the

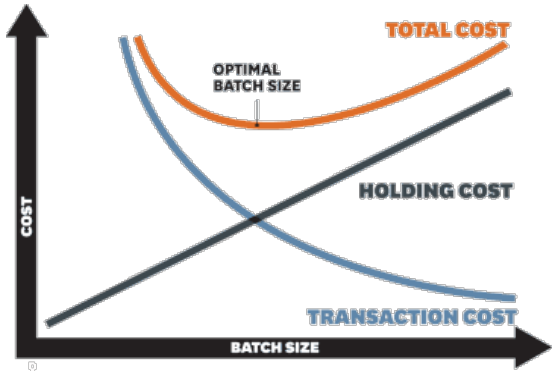
batch sizes are too. Second, developers seem to have an inherent bias to use large batches—possibly because they incorrectly believe that large batches produce economies of scale.

In a well-managed process, the batch size will balance transaction and holding costs (see the exhibit "How to Determine Optimal Batch Size"). It's similar to buying eggs at the grocery store. If you buy a 12-month supply on a single trip, your transaction cost is low, but most of the eggs will spoil, increasing your holding cost. If you buy a one-day supply at a time, your spoilage will be low, but your transaction costs will be high. Intuitively, you try to strike a balance between the two.

How to Determine Optimal Batch Size

Changes in batch size affect two primary costs: the transaction cost and the holding cost. As batch sizes become larger, average inventory levels rise, which raises holding costs. But at the same time, transaction costs decrease, because it takes fewer transactions to service demand.

The optimal batch size is the point where the total cost (combined holding and transaction cost) is lowest. When a company operates near this point, small deviations have little impact. For example, if a company operates at less than 20% above or below the optimal batch size, total costs increase less than 3%. So even rough estimates permit a company to capture large economic benefits.



The companies that understand how this works have exploited IT advances to reduce batch sizes, often with astonishing results. Some software companies that used to test large batches of code every 90 days are now testing much smaller batches several times a day. A manufacturer of computer peripherals that used a similar approach with its software group reduced cycle time in software testing by 95% (from 48 months to 2.5 months), improved efficiency by 220%, and decreased defects by 33%. The cost savings were twice as high as the company had expected. Although those results were exceptional, we have found that reducing batch size improves most development projects significantly. Similarly, computerized modeling and simulation tools have dramatically lowered the optimal batch size of experimentation and testing in companies that develop physical products.

Fallacy 3: Our development plan is great; we just need to stick to it.
In all our consulting work and research, we've never come across a single product-development project whose requirements remained stable throughout the design process. Yet many organizations place inordinate faith in their plans. They attribute any deviations to poor management and execution and, to minimize them, carefully track every step against intermediate targets and milestones. Such thinking is fine for highly repetitive activities in established manufacturing processes. But it can lead to poor results in product innovation, where new insights are generated daily and conditions are constantly changing.

A classic study of technical problem solving done by Thomas Allen of MIT highlights the fluid nature of development work. He found that engineers who were developing an aerospace subsystem conceived of and evaluated a number of design alternatives before selecting one that they judged to be the best. Along the way their preferences changed frequently, as they tested and refined competing technical solutions. This is typical in innovation projects: Testing and experimentation reveal what does and doesn't work, and initial assumptions about costs and value may be disproved.

Defining customers' needs can also be hard to do at the outset of a product-development project. When you think about it, that's not surprising: It isn't easy for customers to accurately specify their needs for solutions that don't yet exist. In fact, familiarity with existing product attributes can interfere with an individual's ability to express his or her need for a novel product. Customers' preferences can also shift abruptly during the course of a development project, as competitors introduce new offerings and new trends emerge.

For all those reasons, sticking to the original plan—no matter how excellent its conception and how skillful its execution—can

be a recipe for disaster. This is not to suggest that we don't believe in planning. Product development is a set of complex activities that require careful coordination and attention to the smallest detail. However, the plan should be treated as an initial hypothesis that is constantly revised as the evidence unfolds, economic assumptions change, and the opportunity is reassessed. (See **"The Value Captor's Process,"** by Rita Gunther McGrath and Thomas Keil, HBR May 2007.)

Fallacy 4: The sooner the project is started, the sooner it will be finished.
As we discussed earlier, idle time is anathema to managers. They tend to exploit any downtime by starting a new project. Even if the task cannot be completed because people have to return to another project, managers reason that anything accomplished on the new project is work that won't have to be done later. Such thinking leads companies to start more projects than they can vigorously pursue, diluting resources.

This dilution is dangerous. If a company embarks on a project before it has the resources to move ahead, it will stumble slowly through the development process. That's problematic because product-development work is highly perishable: Assumptions about technologies and the market can quickly become obsolete. The slower a project progresses, the greater the likelihood it will have to be redirected. Indeed, one branch of the military discovered that its cost and schedule overruns were exponentially proportional (to the fourth power) to a project's duration. In other words, when the original schedule of a project doubled, the cost and schedule overruns increased by a factor of 16.

The importance of reducing the amount of work in process is evident when we look at one of the classic formulas of queuing theory: Little's Law. It simply states that, on average, cycle time is proportional to the size of the queue divided by the processing rate. Thus, if 20 people are ahead of you in line at Starbucks and the barista is serving five people a minute, you will be served in four minutes. You can shorten the cycle time by raising the processing rate or by reducing the number of jobs under way. In most settings the latter is the only practical choice.

â€©For some product developers the solution has been to rigorously control the rate at which they start work. They match it to the rate at which work is actually completed; carefully manage the number of projects in process; make sure that once a project is launched, it is adequately staffed until it is completed; and resist the temptation to steal resources from an ongoing project to squeeze in new ones.

Fallacy 5: The more features we put into a product, the more customers will like it.
Product-development teams seem to believe that adding features creates value for customers and subtracting them destroys value. This attitude explains why products are so complicated: Remote controls seem impossible to use, computers take hours to set up, cars have so many switches and knobs that they resemble airplane cockpits, and even the humble toaster now comes with a manual and LCD displays.

Companies that challenge the belief that more is better create products that are elegant in their simplicity. Bang & Olufsen, the Danish manufacturer of audio products, televisions, and telephones, understands that customers don't necessarily want to fiddle with the equalizer, balance, and other controls to find the optimum combination of settings for listening to music. Its high-end speakers automatically make the adjustments needed to reproduce a song with as much fidelity to the original as possible. All that's left for users to select is the volume.

Getting companies to buy into and implement the principle that less can be more is hard because it requires extra effort in two areas of product development:

Defining the problem. Articulating the problem that developers will try to solve is the most underrated part of the innovation process. Too many companies devote far too little time to it. But this phase is important because it's where teams develop a clear understanding of what their goals are and generate hypotheses that can be tested and refined through experiments. The quality of a problem statement makes all the difference in a team's ability to focus on the few features that really matter.

When Walt Disney was planning Disneyland, he didn't rush to add more features (rides, kinds of food, amount of parking) than other amusement parks had. Rather, he began by asking a much larger question: How could Disneyland provide visitors with a magical customer experience? Surely, the answer didn't come overnight; it required painstakingly detailed research, constant experimentation, and deep insights into what "magical" meant to Disney and its customers. IDEO and other companies have dedicated phases in which they completely immerse themselves in the context in which the envisioned product or service will be used. Their developers read everything of interest about the markets, observe and interview future users, research offerings that will compete with the new product, and synthesize everything that they have learned into pictures, models, and diagrams. The result is deep insights into customers that are tested, improved, or abandoned throughout the iterative development process.

Determining what to hide or omit. Teams are often tempted to show off by producing brilliant technical solutions that amaze their peers and management. But often customers would prefer a product that just works effortlessly. From a customer's point of view,

the best solutions solve a problem in the simplest way and hide the work that developers are so proud of.

One company that has understood this is Apple. It is known for many things—innovative products, stylish designs, and savvy marketing—but perhaps its greatest strength is its ability to get to the heart of a problem. (See **"The Real Leadership Lessons of Steve Jobs,"** by Walter Isaacson, in our April issue.) As the late Steve Jobs once explained, "When you start looking at a problem and it seems really simple, you don't really understand the complexity of the problem. And your solutions are way too oversimplified. Then you get into the problem, and you see it's really complicated. And you come up with all these convoluted solutions....That's where most people stop." Not Apple. It keeps on plugging away. "The really great person will keep on going," said Jobs, "and find...the key underlying principle of the problem and come up with a beautiful, elegant solution that works."

Determining which features to omit is just as important as—and perhaps more important than—figuring out which ones to include. Unfortunately, many companies, in an effort to be innovative, throw in every possible bell and whistle without fully considering important factors such as the value to customers and ease of use. When such companies do omit some planned functionality, it's typically because they need to cut costs or have fallen behind schedule or because the team has failed in some other way.

Instead, managers should focus on figuring out whether the deletion of any proposed feature might improve a particular product and allow the team to concentrate on things that truly heighten the overall customer experience. This can be determined by treating each alleged requirement as a hypothesis and testing it in small, quick experiments with prospective customers.

Development teams often assume that their products are done when no more features can be added. Perhaps their logic should be the reverse: Products get closer to perfection when no more features can be eliminated. As Leonardo da Vinci once said, "Simplicity is the ultimate sophistication."

Fallacy 6: We will be more successful if we get it right the first time.
Many product-development projects fail to meet their objectives for budgets, schedules, and technical performance. Undoubtedly, poor planning, rigid processes, and weak leadership all play a role. But another cause that's often overlooked is managers' demand that their teams "get it right the first time." Requiring success on the first pass biases teams toward the least-risky solutions, even if customers don't consider them much of an improvement over what's already available. Worse yet, teams have little incentive to pursue innovative solutions to customers' problems.

To avoid making mistakes, teams follow a linear process in which each stage (specify, design, build, test, scale, launch) is carefully monitored at review "gates." Work on the next stage cannot begin until the project passes through the gate. As the project moves down the line, significant commitments are made and the cost of responding to new insights increases by orders of magnitude. Successful tests in late stages are celebrated, and surprises, no matter how valuable they are, are considered setbacks. Unfortunately, such a linear process flow can cause project overruns because test feedback is delayed, teams cling to bad ideas longer than they should, and problems aren't unearthed until it's expensive to solve them.

A tolerance for "getting it wrong the first time" can be the better strategy as long as people iterate rapidly and frequently and learn quickly from their failures. Advances in simulation and rapid-prototyping technologies have made operating in this fashion vastly easier and less expensive.

Practical Guidelines for Overcoming Common Fallacies

A checklist for today's product-development managers

1. Make queues and information flows visible.

2. Quantify the cost of delays and factor it into your decisions.

3. Introduce resource slack where utilization is highest.

4. Shift the focus of control systems from efficiency to response time.

5. Reduce transaction costs to enable smaller batch sizes and faster feedback.

6. Experiment with smaller batches; you can easily revert to large batches if this doesn't work.

7. Treat the development plan as a hypothesis that will evolve as new information becomes available.

8. Start projects only when you are ready to make a full commitment.

9. Aim for simplicity: Ask what features can be deleted, not just what can be added.

10. Experiment early, rapidly, and frequently, with computer models and physical prototypes, in controlled and real-life customer environments.

11. Emphasize overlapping and iterative—not linear—process designs.

12. Focus on quick feedback instead of first-pass success.

Consider what we found in a **study of 391 teams that designed custom integrated circuits.** Teams that followed an iterative approach and conducted early and frequent tests made more errors along the way. But because they used low-cost prototyping technologies, they outperformed (in terms of the time and effort required) teams that tried to get their design right the first time. The teams that faced high prototyping costs invested more effort on specification, development, and verification. And because they did their iterations later in the process—and did far fewer of them—they delayed the discovery of critical problems.

Experimenting with many diverse ideas is crucial to innovation projects. When people experiment rapidly and frequently, many novel concepts will fail, of course. But such early failures can be desirable because they allow teams to eliminate poor options quickly and focus on more-promising alternatives. A crash test that shows that a car design is unsafe, a drug candidate that proves to be toxic, or a software user interface that confuses customers can all be desirable outcomes—provided that they occur early in a process, when few resources have been committed, designs are still very flexible, and other solutions can be tried.

A classic example of the advantages of the "fail early, fail often" approach is Team New Zealand's surprising victory in the 1995 America's Cup. To test ideas for improving the keel design, the team used two nearly identical boats: one boat that was modified during the course of the project and a "control" boat that was not. On a daily basis, the team simulated hypotheses on a local graphics workstation, applied those that looked promising to the one boat, raced it against the control, and analyzed the results. In contrast, its competitor, Team Dennis Conner, which had access to more-powerful computers (supercomputers at Boeing), ran large batches of simulations every few weeks and then tested possible solutions on one boat. The result: Team New Zealand completed many more learning cycles, eliminated unpromising ideas more rapidly, and ended up beating Team Dennis Conner's boat *Young America.*

What we hope is becoming clear by now is that experiments resulting in failures are not necessarily failed experiments. They generate new information that an innovator was unable to foresee. The faster the experimentation cycle, the more feedback can be gathered and incorporated into new rounds of experiments with novel and potentially risky ideas. In such an environment employees tend to persevere when times get tough, engage in more-challenging work, and outperform their risk-averse peers.

But creating this kind of environment isn't easy—a topic that Amy C. Edmondson of Harvard Business School explored in **"Strategies for Learning from Failure"** (HBR April 2011). Failure can lead to embarrassment and expose gaps in knowledge, which can undermine individuals' self-esteem and standing in an organization. After all, how often are managers promoted and teams rewarded for the early exposure of failures that lead a project to be killed—even though the early redeployment of precious resources benefits the company? This is especially true in organizations that have built a "zero tolerance for failure" or "error-free" (Six Sigma) environment.

Thomas Alva Edison understood all this. He organized his famous laboratories around the concept of rapid experimentation, locating machine shops for building models close to the rooms where experiments were conducted so that machinists could cooperate closely with researchers. The labs had libraries containing a vast number of volumes so that information could be found quickly; nearby storerooms with ample quantities of supplies; and a diverse workforce of craftsmen, scientists, and engineers. Edison wanted to make sure that when he or his people had an idea, it could be immediately turned into a working model or prototype. "The real measure of success is the number of experiments that can be crowded into 24 hours," **he said.**

Advances in **information technology,** such as computer-aided design, modeling, and simulation, have already allowed companies to make great strides in developing better products in less time and at a lower cost. Many companies, however, have not tapped the full potential of these tools, because their management thinking has not evolved as quickly as the technology: They still approach the highly variable information-generating work of product development as if it were like manufacturing. As advances in IT continue, the opportunity to improve the product-development process will become even greater. But so will the risks for companies that fail to recognize that product development is profoundly different from manufacturing.

Stefan Thomke is the William Barclay Harding Professor of Business Administration at Harvard Business School. **Donald Reinertsen** is president of Reinertsen & Associates, a consulting firm in Redondo Beach, California. His most recent book is *The Principles of Product Development Flow* (Celeritas Publishing, 2009).