1

Goal-Directed Design

This book has a simple premise: If we design and construct products in such a way that the people who use them achieve their goals, these people will be satisfied, effective, and happy and will gladly pay for the products and recommend that others do the same. Assuming that this can be achieved in a cost-effective manner, it will translate into business success.

On the surface, this premise sounds quite obvious and straightforward: Make people happy, and your products will be a success. Why then are so many digital products so difficult and unpleasant to use? Why aren't we all happy and successful?

Digital Products Need Better Design Methods

Most digital products today emerge from the development process like a creature emerging from a bubbling tank. Developers, instead of planning and executing with a mind towards satisfying the needs of the people who purchase and use their products, end up creating technologically focused solutions that are difficult to use and control. Like mad scientists, they fail because they have not imbued their creations with humanity.

Design, according to industrial designer Victor Papanek, is *the conscious and intuitive effort to impose meaningful order*. We propose a somewhat more detailed definition of human-oriented design activities:

- Understanding users' desires, needs, motivations, and contexts
- Understanding business, technical, and domain opportunities, requirements, and constraints
- Using this knowledge as a foundation for plans to create products whose form, content, and behavior is useful, usable, and desirable, as well as economically viable and technically feasible

This definition is useful for many design disciplines, although the precise focus on form, content, and behavior will vary depending on what is being designed. For example, an informational Web site may require particular attention to *content*, whereas the design of a chair is primarily concerned with *form*. As we discussed in the Introduction, interactive digital products are uniquely imbued with complex *behavior*.

When performed using the appropriate methods, design can provide the missing human connection in technological products. But clearly, most current approaches to the design of digital products aren't working as advertised.

The creation of digital products today

Digital products come into this world subject to the push and pull of two, often opposing, forces — developers and marketers. While marketers are adept at understanding and quantifying a marketplace opportunity, and at introducing and positioning a product within that market, their input into the product design process is often limited to lists of requirements. These requirements often have little to do with what users actually *need* or *desire* and have more to do with chasing the competition, managing IT resources with to-do lists, and making guesses based on market surveys — what people *say* they'll buy. (Contrary to what you might suspect, few users are able to clearly articulate their needs. When asked direct questions about the products they use, most tend to focus on low-level tasks or workarounds to product flaws.) Unfortunately, reducing an interactive product to a list of hundreds of features doesn't lend itself to the kind of graceful orchestration that is required to make complex technology useful. Adding "easy to use" to the list of requirements does nothing to improve the situation.

Developers, on the other hand, often have no shortage of input into the product's final form and behavior. Because they are in charge of construction, they decide exactly what gets built. And they, too, have a different set of imperatives than the product's eventual users. Good developers are focused on solving challenging technical problems, following good engineering practices, and meeting deadlines. They are often given incomplete, confusing, and sometimes contradictory instructions and are forced to make significant decisions about the user experience with little time or background.

Thus, the people who are most often responsible for the creation of our digital products rarely take into account the users' *goals*, needs, or motivations, and at the same time tend to be highly reactive to market trends and technical constraints. This can't help but result in products that lack a coherent user experience. We'll soon see why goals are so important in addressing this issue.

The results of poor product vision are, unfortunately, digital products that irritate, reduce productivity, and fail to meet user needs. Figure 1-1 shows the evolution of the development process and where, if at all, design has historically fit in. Most of digital product development is stuck in the first, second, or third step of this evolution, where design either plays no real role or it becomes a surface-level patch on shoddy interactions — "lipstick on the pig," as one of our clients once referred to it. The design process, as we will soon discuss, should *precede* coding and testing to ensure that products truly meet the needs of users.

In the dozen years since the publication of the first edition of this book, software and interactive products have certainly improved. Many companies have begun to focus on serving the needs of people with their products, and are spending the time and money to do upfront design. Many more companies are still failing to do this, and as they maintain their focus on technology and marketing data, they continue to create the kind of digital products we've all grown to despise. Here are a few symptoms of this affliction.

Digital products are rude

Digital products often blame users for making mistakes that are not their fault, or should not be. Error messages like the one in Figure 1-2 pop up like weeds announcing that the user has failed yet again. These messages also demand that the user acknowledge his failure by agreeing: OK.

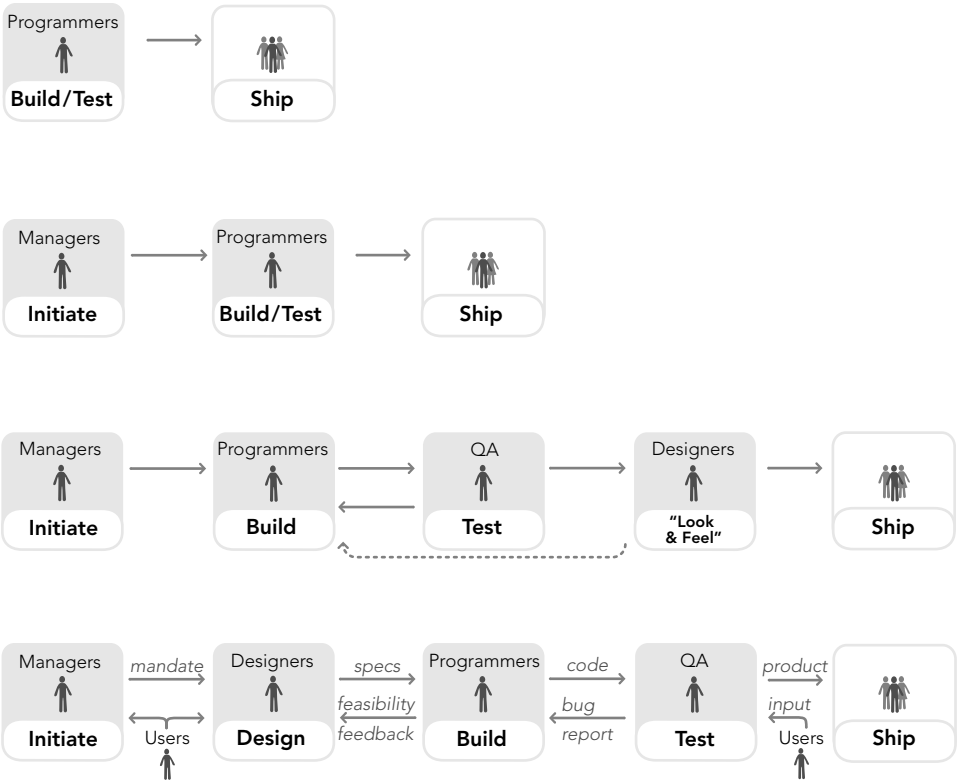

Figure 1-1 The evolution of the software development process. The first diagram depicts the early days of the software industry when smart programmers dreamed up products, and then built and tested them. Inevitably, professional managers were brought in to help facilitate the process by translating market opportunities into product requirements. As depicted in the third diagram, the industry matured, testing became a discipline in its own right, and with the popularization of the graphical user interface (GUI), graphic designers were brought in to create icons and other visual elements. The final diagram shows the Goal-Directed approach to software development where decisions about a product's capabilities, form, and behavior are made before the expensive and challenging construction phase.

Figure 1-2 Thanks for sharing. Why didn't the program notify the library? What did it want to notify the library about? Why is it telling us? And what are we OKing, anyway? It is not OK that the program failed!

Digital products and software frequently interrogate users, peppering them with a string of terse questions that they are neither inclined or prepared to answer: "Where did you hide that file?" Patronizing questions like "Are you sure?" and "Did you really want to delete that file or did you have some other reason for pressing the Delete key?" are equally irritating and demeaning.

Our software-enabled products also fail to act with a basic level of decency. They forget information we tell them and don't do a very good job of anticipating our needs. For example, the feature-rich Palm Treo smartphone doesn't anticipate that a user might want to add the phone number of someone who has just called to an *existing* contact. It doesn't take a lot of research or imagination to deduce that this is something that many users will want to do, but nevertheless one is forced to go through a complicated maze involving copying the phone number, navigating to the contact in question, and pasting into the appropriate field.

Digital products require people to think like computers

Digital products regularly assume that people are technology literate. For example, in Microsoft Word, if a user wants to rename a document she is editing, she must know that she must either close the document, or use the "Save As..." menu command (and remember to delete the file with the old name). These behaviors are not consistent with the way a normal person thinks about renaming something; rather, they require that a person change her thinking to be more like the way a computer works.

Digital products are also often obscure, hiding meaning, intentions, and actions from users. Programs often express themselves in incomprehensible jargon that cannot be fathomed by normal users ("How many stop bits?") and are sometimes incomprehensible even to experts ("Please specify IRQ.").

Digital products exhibit poor behavior

If a 10-year-old child behaved like some software programs or devices, he'd be sent to his room without supper. Programs forget to shut the refrigerator door, leave shoes in the middle of the floor, and can't remember what you told them only five minutes earlier. For example, if you save a Microsoft Word document, print it, and then try to close it, the program once again asks you if you want to save it! Evidently the act of printing caused the program to think the document had changed, even though it did not. Sorry, Mom, I didn't hear you.

Programs often require us to step out of the main flow of tasks to perform functions that should fall immediately to hand. Dangerous commands, however, are often presented right up front where unsuspecting users can accidentally trigger them. The overall appearance of many programs is overly complex and confusing, making navigation and comprehension difficult.

Digital products require humans to do the heavy lifting

Computers and their silicon-enabled brethren are supposed to be labor-saving devices, but every time we go out into the field to watch real people doing their jobs with the assistance of technology, we are struck and horrified by how much work they are forced to do to manage the operation of software. This work can be anything from manually keying values from one window into another, to copying and pasting between applications that don't otherwise speak to each other, to the ubiquitous clicking and pushing and pulling of windows around the screen to access hidden functionality that people use every day to do their job.

Why are these products so bad?

So what, then, is the real problem? Why is the technology industry generally so inept at designing the interactive parts of digital products? There are three primary reasons: ignorance about users, a conflict of interest between serving human needs and construction priorities, and the lack of a process for understanding human needs as an aid to developing appropriate product form and behavior.

Ignorance about users

It's a sad truth that the digital technology industry doesn't have a good understanding of what it takes to make users happy. In fact, most technology products get built without much understanding of the users. We might know what *market segment* our users are in, how much money they make, how much money they like to

spend on weekends, and what sort of cars they buy. Maybe we even have a vague idea what kind of jobs they have and some of the major tasks that they regularly perform. But does any of this tell us how to make them happy? Does it tell us *how* they will actually use the product we're building? Does it tell us *why* they are doing whatever it is they might need our product for, *why* they might want to choose our product over our competitors, or *how* we can make sure they do? Unfortunately, it does not.

We'll soon see how to address the issue of understanding users and their behaviors with products.

Conflicting interests

A second problem affects the ability of vendors and manufacturers to make users happy. There is an important conflict of interest in the world of digital product development: The people who build the products — programmers — are usually also the people who design them. Programmers are often required to choose between ease of coding and ease of use. Because programmers' performance is typically judged by their ability to code efficiently and meet incredibly tight deadlines, it isn't difficult to figure out what direction most software-enabled products take. Just as we would never permit the prosecutor in a legal trial to also adjudicate the case, we should make sure that the people designing a product are not the same people building it. Even with appropriate skills and the best intentions, it simply isn't possible for a programmer to advocate effectively for the user, the business, and the technology all at the same time.

The lack of a process

The third reason the digital technology industry isn't cranking out successful products is that it has no reliable *process* for doing so. Or, to be more accurate, it doesn't have a *complete* process for doing so. Engineering departments follow — or should follow — rigorous engineering methods that ensure the *feasibility* and quality of the technology. Similarly, marketing, sales, and other business units follow their own well-established methods for ensuring the commercial *viability* of new products. What's left out is a repeatable, predictable, and analytical process for *transforming an understanding of users into products that both meet their needs and excite their imaginations*.

When we think about complex mechanical devices, we take for granted that they have been carefully designed for use, in addition to being engineered. Most manufactured objects are quite simple, and even complex mechanical products are quite

simple when compared to most software and software-enabled products that can be compiled from over one million lines of code (compare this to a mechanical artifact of overwhelming complexity such as the space shuttle, which has 250,000 parts, only a small percentage of which are *moving* parts). Yet most software has never undergone a rigorous design process from a user-centered perspective.

In the worst case, decisions about what a digital product will do and how it will communicate with users is simply a byproduct of its construction. Programmers, deep in their thoughts of algorithms and code, end up "designing" product behaviors and user interfaces the same way that miners end up "designing" the landscape with cavernous pits and piles of rubble. In unenlightened development organizations, the digital product interaction design process alternates between the accidental and the nonexistent.

Sometimes organizations do adopt a design process, but it isn't quite up to the task. Many programmers today embrace the notion that integrating customers directly into the development process on a frequent basis can solve human interface design problems. Although this has the salutary effect of sharing the responsibility for design with the user, it ignores a serious methodological flaw: a confusion of domain knowledge with design knowledge. Customers, although they might be able to articulate the problems with an interaction, are not often capable of visualizing the solutions to those problems. Design is a specialized skill, just like programming. Programmers would never ask users to help them *code*; design problems should be treated no differently. In addition, customers who *purchase* a product may not be the same people who *use* it from day to day, a subtle but important distinction.

This doesn't mean that designers shouldn't be interested in getting feedback on their proposed solutions. However, each member of the product team should respect the others' areas of expertise. Imagine a patient who visits his doctor with a horrible stomachache. "Doctor," he says, "it *really* hurts. I think it's my appendix. You've got to take it out as soon as possible." Of course, a responsible physician wouldn't perform the surgery without question. The patient can express the symptoms, but it takes the doctor's professional knowledge to make the correct diagnosis.

To better understand how to create a workable process that brings user-centered design to digital products, it's useful to understand a bit more about the history of design in manufacturing and about how the challenges of interactive products have substantially changed the demands on design.

The Evolution of Design in Manufacturing

In the early days of industrial manufacturing, engineering and marketing processes alone were sufficient to produce *desirable* products: It didn't take much more than good engineering and reasonable pricing to produce a hammer, diesel engine, or tube of toothpaste that people would readily purchase. As time progressed, manufacturers of consumer products realized that they needed to differentiate their products from functionally identical products made by competitors, so design was introduced as a means to increase user desire for a product. Graphic designers were employed to create more effective packaging and advertising, and industrial designers were engaged to create more comfortable, useful, and exciting forms.

The conscious inclusion of design heralded the ascendance of the modern triad of product development concerns identified by Larry Keeley of the Doblin Group: capability, viability, and desirability (see Figure 1-3). If any one of these three foundations is significantly weak in a product, it is unlikely to stand the test of time.

Now enter the computer, the first machine created by humans that is capable of almost limitless *behavior* when properly coded into software. The interesting thing about this complex behavior, or interactivity, is that it completely alters the nature of the products it touches. Interactivity is compelling to humans, so compelling that other aspects of an interactive product become marginal. Who pays attention to the black box that sits under your desk — it is the interactive screen, keyboard, and mouse to which users pay attention. Yet, the interactive behaviors of software and other digital products, which should be receiving the lion's share of design attention, all too frequently receive no attention at all.

The traditions of design that corporations have relied on to provide the critical pillar of desirability for products don't provide much guidance in the world of interactivity. Design of behavior is a different kind of problem that requires greater knowledge of *context*, not just rules of visual composition and brand. Design of behavior requires an understanding of the user's relationship with the product from before purchase to end-of-life. Most important of all is the understanding of how the user wishes to use the product, in what ways, and to what ends.

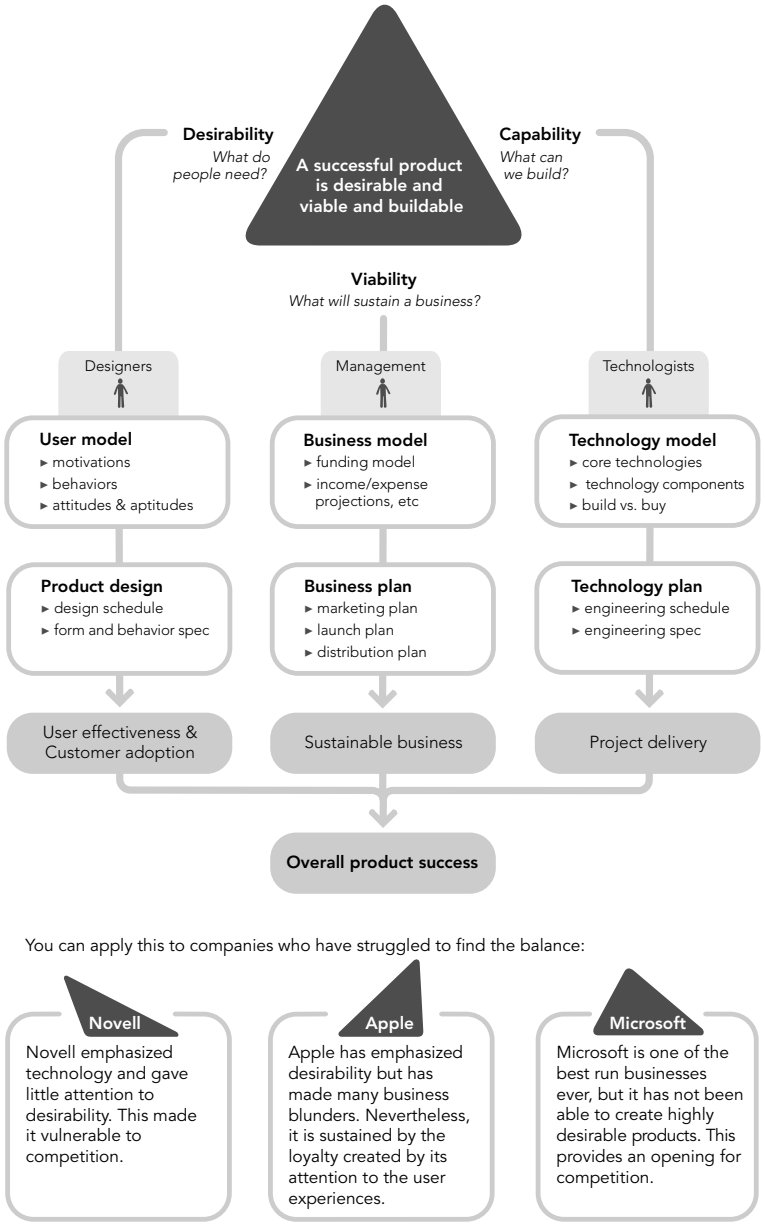

Figure 1-3 Building successful digital products. The diagram indicates the three major processes that need to be followed in tandem to create successful technology products. This book addresses the first and foremost issue: how to create a product people will desire.

Planning and Designing Behavior

The planning of complex digital products, especially ones that interact directly with humans, requires a significant upfront effort by professional designers, just as the planning of complex physical structures that interact with humans requires a significant upfront effort by professional architects. In the case of architects, that planning involves understanding how the humans occupying the structure live and work, and designing spaces to support and facilitate those behaviors. In the case of digital products, the planning involves understanding how the humans using the product live and work, and designing product behavior and form that supports and facilitates the human behaviors. Architecture is an old, well-established field. The design of product and system behavior — **interaction design** — is quite new, and only in recent years has it begun to come of age as a discipline.

Interaction design isn't merely a matter of aesthetic choice; rather, it is based on an understanding of users and cognitive principles. This is good news because it makes the design of behavior quite amenable to a repeatable process of analysis and synthesis. It doesn't mean that the design of behavior can be automated, any more than the design of form or content can be automated, but it *does* mean that a systematic approach is possible. Rules of form and aesthetics mustn't be discarded, of course, but they must work in harmony with the larger concern of achieving user goals via appropriately designed behaviors.

This book presents a set of methods to address the needs of this new kind of behavior-oriented design, which addresses the *goals* (Rudolf, 1998) and motivations of users: **Goal-Directed Design**. To understand Goal-Directed Design, we first need to better understand user goals and how they provide the key to designing appropriate interactive behavior.

Recognizing User Goals

So what are user goals? How can we identify them? How do we know that they are real goals, rather than tasks they are forced to do by poorly designed tools or business processes? Are they the same for all users? Do they change over time? We'll try to answer those questions in the remainder of this chapter.

Users' goals are often quite different from what we might guess them to be. For example, we might think that an accounting clerk's goal is to process invoices efficiently. This is probably not true. Efficient invoice processing is more likely the goal of the clerk's employer. The clerk is more likely concentrating on goals like appearing competent at his job and keeping himself engaged with his work while

performing routine and repetitive tasks, although he may not verbally (or even consciously) acknowledge this.

Regardless of the work we do and the tasks we must accomplish, most of us share these simple, personal goals. Even if we have higher aspirations, they are still more personal than work related: winning a promotion, learning more about our field, or setting a good example for others, for instance.

Products designed and built to achieve business goals alone will eventually fail; personal goals of users need to be addressed. When the user's personal goals are met by the design, business goals are far more effectively achieved, for reasons we'll explore in more detail in later chapters.

If you examine most commercially available software, Web sites, and digital products today, you will find that their user interfaces fail to meet user goals with alarming frequency. They routinely:

- Make users feel stupid
- Cause users to make big mistakes
- Require too much effort to operate effectively
- Don't provide an engaging or enjoyable experience

Most of the same software is equally poor at achieving its business purpose. Invoices don't get processed all that well. Customers don't get serviced on time. Decisions don't get properly supported. This is no coincidence.

The companies that develop these products don't have the right priorities. Most focus their attention far too narrowly on implementation issues, which distract them from the needs of users.

Even when businesses become sensitive to their users, they are often powerless to change their products because the conventional development process assumes that the user interface should be addressed after coding begins — sometimes even after it ends. But just as you cannot effectively design a building after construction begins, you cannot easily make a program serve users' goals once there is a significant and inflexible code base in place.

Finally, when companies *do* focus on the users, they tend to pay too much attention to the *tasks* that users engage in and not enough attention to their *goals* in performing those tasks. Software can be technologically superb and perform each business task with diligence, yet still be a critical and commercial failure. We can't ignore technology or tasks, but they play only a part in a larger schema that includes designing to meet user goals.

Goals versus tasks and activities

Goals are not the same as tasks or activities. A goal is an expectation of an end condition, whereas both activities and tasks are intermediate steps (at different levels of organization) that help someone to reach a goal or set of goals.

Donald Norman describes a hierarchy in which activities are composed of tasks, which are in turn composed of actions, which are then themselves composed of operations. Using this scheme, Norman advocates "Activity-Centered Design" (ACD), which focuses first and foremost on understanding activities. His claim is that humans adapt to the tools at hand, and understanding the activities that people perform with a set of tools can more favorably influence the design of those tools. The foundation of Norman's thinking comes from Activity Theory, a Soviet-era Russian theory of psychology that emphasizes understanding who people are by understanding how they interact with the world, and which has in recent years been adapted to the study of human-computer interaction, most notably by Bonnie Nardi.

Norman concludes, correctly, that the traditional task-based focus of digital product design has yielded inadequate results. Many developers and usability professionals still approach the design of interfaces by asking, "What are the tasks?" Although this may get the job done, it won't produce much more than an incremental improvement: It won't provide a solution that differentiates your product in the market, and very often won't really satisfy the user.

While Norman's ACD takes some important steps in the right direction by highlighting the importance of the user's context, we do not believe that it goes quite far enough. While a method like ACD can be very useful in properly breaking down the "what" of user behaviors, it really doesn't address what should be the first question asked by any designer: *Why* is a user performing an activity, task, action, or operation in the first place? Goals motivate people to perform activities; understanding goals allows you to understand the expectations and aspirations of your users, which can in turn help you decide which activities are truly relevant to your design. Task and activity analysis is useful at the detail level, but only after user goals have been analyzed. Asking, "What are the user's goals?" lets you understand the *meaning* of activities to your users, and thus create more appropriate and satisfactory designs.

If you're still unsure about the difference between goals and activities or tasks, there is an easy way to tell the difference between them. Since goals are driven by human motivations, they change very slowly — if at all — over time. Activities and tasks are much more transient, since they are based almost entirely on whatever technology is

at hand. For example, when traveling from St. Louis to San Francisco, a person's *goals* are likely to include traveling quickly, comfortably, and safely. In 1850, a settler wishing to travel quickly and comfortably would have made the journey in a covered wagon; in the interest of safety, he would have brought along his trusty rifle. Today, a businessman traveling from St. Louis to San Francisco makes the journey in a jet aircraft and, in the interest of safety, he is required to leave his firearms at home. The *goals* of the settler and businessman remain unchanged, but their activities and tasks have changed so completely with the changes in technology that they are, in some respects, in direct opposition.

Design based solely on understanding activities or tasks runs the risk of trapping the design in a model imposed by an outmoded technology, or using a model that meets the goals of a corporation without meeting the goals of their users. Looking through the lens of goals allows you to leverage available technology to eliminate irrelevant tasks and to dramatically streamline activities. Understanding users' goals can help designers eliminate the tasks and activities that better technology renders unnecessary for humans to perform.

Designing to meet goals in context

Many designers assume that making interfaces easier to learn should always be a design target. Ease of learning is an important guideline, but in reality, as Brenda Laurel notes, the design target really depends on the context — who the users are, what they are doing, and what goals they have. You simply can't create good design by following rules disconnected from the goals and needs of the users of your product.

Let us illustrate: Take an automated call-distribution system. The people who use this product are paid based on how many calls they handle. Their most important concern is not ease of learning, but the efficiency with which users can route calls, and the rapidity with which those calls can be completed. Ease of learning is also important because it affects the happiness and, ultimately, the turnover rate of employees, so both ease and throughput should be considered in the design. But there is no doubt that throughput is the dominant demand placed on the system by the users and, if necessary, ease of learning should take a back seat. A program that walks the user through the call-routing process step by step each time merely frustrates him after he's learned the ropes.

On the other hand, if the product in question is a kiosk in a corporate lobby helping visitors find their way around, ease of use for first-time users is clearly a major goal.

A general guideline of interaction design that seems to apply particularly well to productivity tools is that *good design makes users more effective*. This guideline takes

into account the universal human goal of not looking stupid, along with more particular goals of business throughput and ease of use that are relevant in most business situations.

It is up to you as a designer to determine how you can make the users of your product more effective. Software that enables users to perform their tasks *without addressing their goals* rarely helps them be truly effective. If the task is to enter 5000 names and addresses into a database, a smoothly functioning data-entry application won't satisfy the user nearly as much as an automated system that extracts the names from the invoicing system.

Although it is the user's job to focus on her tasks, the designer's job is to look beyond the task to identify *who* the most important users are, and then to determine *what* their goals might be and *why*. The design process, which we describe in the remainder of this chapter and detail in the remaining chapters of Part I, provides a structure for determining the answers to these questions, a structure by which solutions based on this information can be systematically achieved.

The Goal-Directed Design Process

Most technology-focused companies don't have an adequate process for user-centered design, if they have a process at all. But even the more enlightened organizations, those that can boast of an established process, come up against some critical issues that result from traditional ways of approaching the problems of research and design.

In recent years, the business community has come to recognize that user research is necessary to create good products, but the proper nature of that research is still in question in many organizations. Quantitative market research and market segmentation is quite useful for *selling* products but falls short of providing critical information about *how people actually use products* — especially products with complex behaviors (see Chapter 4 for a more in-depth discussion of this topic). A second problem occurs after the results have been analyzed: Most traditional methods don't provide a means of *translating research results into design solutions*. A hundred pages of user survey data don't easily translate into a set of product requirements, and they say even less about how those requirements should be expressed in terms of a meaningful and appropriate interface structure. Design remains a black box: "A miracle happens here." This gap between research results and the ultimate design solution is the result of a process that doesn't connect the dots from user to final product. We'll soon see how to address this problem with Goal-Directed methods.

Bridging the gap

As we have briefly discussed, the role of design in the development process needs to change. We need to start thinking about design in new ways, and start thinking differently about how product decisions are made.

Design as product definition

Design has, unfortunately, become a limiting term in the technology industry. For many developers and managers, the word has come to mean what happens in the third process diagram shown in Figure 1-1: a visual facelift on the **implementation model** (see Chapter 2). But design, when properly deployed (as in the fourth process diagram shown in Figure 1-1), both identifies user requirements and defines a detailed plan for the behavior and appearance of products. In other words, design provides true **product definition**, based on goals of users, needs of business, and constraints of technology.

Designers as researchers

If design is to become product definition, designers need to take on a broader role than that assumed in traditional design, particularly when the object of this design is complex, interactive systems.

One of the problems with the current development process is that roles in the process are overspecialized: Researchers perform research, and designers perform design (see Figure 1-4). The results of user and market research are analyzed by the usability and market researchers and then thrown over the transom to designers or programmers. What is missing in this model is a systematic means of translating and synthesizing the research into design solutions. One of the ways to address this problem is for designers to learn to be researchers.

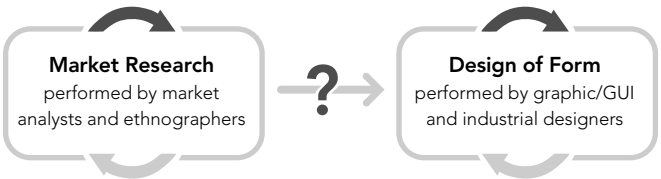

Figure 1-4 A problematic design process. Traditionally, research and design have been separated, with each activity handled by specialists. *Research* has, until recently, referred primarily to market research, and *design* is too often limited to visual design or skin-deep industrial design. More recently, **user research** has expanded to include qualitative, ethnographic data. Yet, without including designers in the research process, the connection between research data and design solutions remains tenuous at best.

There is a compelling reason for involving designers in the research process. One of the most powerful tools designers bring to the table is empathy: the ability to feel what others are feeling. The direct and extensive exposure to users that proper user research entails immerses designers in the users' world, and gets them thinking about users long before they propose solutions. One of the most dangerous practices in product development is isolating designers from the users because doing so eliminates empathic knowledge.

Additionally, it is often difficult for pure researchers to know what user information is really important from a design perspective. Involving designers directly in research addresses both issues.

In the authors' practice, designers are trained in the research techniques described in Chapter 4 and perform their research without further support or collaboration. This is a satisfactory solution, provided that your team has the time and resources to train your designers fully in these techniques. If not, a cross-disciplinary team of designers and dedicated user researchers is appropriate.

Although research practiced by designers takes us part of the way to Goal-Directed Design solutions, there is still a translation gap between research results and design details. The puzzle is missing several pieces, as we will discuss next.

Between research and design: Models, requirements, and frameworks

Few design methods in common use today incorporate a means of effectively and systematically translating the knowledge gathered during research into a detailed design specification. Part of the reason for this has already been identified: Designers have historically been out of the research loop and have had to rely on third-person accounts of user behaviors and desires.

The other reason, however, is that few methods capture user behaviors in a manner that appropriately directs the definition of a product. Rather than providing information about user goals, most methods provide information at the task level. This type of information is useful for defining layout, workflow, and translation of functions into interface controls, but is less useful for defining the basic framework of what a product *is*, what it *does*, and how it should meet the broad needs of the user.

Instead we need an explicit, systematic process to bridge the gap between research and design for defining user models, establishing design requirements, and translating those into a high-level interaction framework (see Figure 1-5). Goal-Directed Design seeks to bridge the gap that currently exists in the digital product development process, the gap between user research and design, through a combination of new techniques and known methods brought together in more effective ways.

A process overview

Goal-Directed Design combines techniques of ethnography, stakeholder interviews, market research, detailed user models, scenario-based design, and a core set of interaction principles and patterns. It provides solutions that meet the needs and goals of users, while also addressing business/organizational and technical imperatives. This process can be roughly divided into six phases: Research, Modeling, Requirements Definition, Framework Definition, Refinement, and Support (see Figure 1-5). These phases follow the five component activities of interaction design identified by Gillian Crampton Smith and Philip Tabor — understanding, abstracting, structuring, representing, and detailing — with a greater emphasis on modeling user behaviors and defining system behaviors.

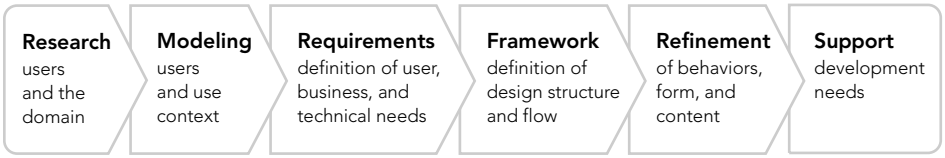

Figure 1-5 The Goal-Directed Design process.

The remainder of this chapter provides a high-level view of the five phases of Goal-Directed Design, and Chapters 4–7 provide more detailed discussion of the methods involved in each of these phases. See Figure 1-6 for a more detailed diagram of the process, including key collaboration points and design concerns.

Research

The Research phase employs ethnographic field study techniques (observation and contextual interviews) to provide qualitative data about potential and/or actual users of the product. It also includes competitive product audits, reviews of market research and technology white papers and brand strategy, as well as one-on-one interviews with stakeholders, developers, subject matter experts (SMEs), and technology experts as suits the particular domain.

One of the principal outcomes of field observation and user interviews is an emergent set of **behavior patterns** — identifiable behaviors that help categorize modes of use of a potential or existing product. These patterns suggest goals and motivations (specific and general desired outcomes of using the product). In business and technical domains, these behavior patterns tend to map into professional roles; for consumer products, they tend to correspond to lifestyle choices. Behavior patterns and the goals associated with them drive the creation of **personas** in the Modeling phase. Market research helps select and filter valid personas that fit business

models. Stakeholder interviews, literature reviews, and product audits deepen the designers' understanding of the domain and elucidate business goals, brand attributes, and technical constraints that the design must support.

Chapter 4 provides a more detailed discussion of Goal-Directed research techniques.

Modeling

During the Modeling phase, behavior and workflow patterns discovered through analysis of the field research and interviews are synthesized into domain and user models. Domain models can include information flow and workflow diagrams. User models, or **personas**, are detailed, composite **user archetypes** that represent distinct groupings of behaviors, attitudes, aptitudes, goals, and motivations observed and identified during the Research phase.

Personas serve as the main characters in a narrative, scenario-based approach to design that iteratively generates design concepts in the Framework Definition phase, provides feedback that enforces design coherence and appropriateness in the Refinement phase, and represents a powerful communication tool that helps developers and managers to understand design rationale and to prioritize features based on user needs. In the Modeling phase, designers employ a variety of methodological tools to synthesize, differentiate, and prioritize personas, exploring different *types* of goals and mapping personas across ranges of behavior to ensure there are no gaps or duplications.

Specific design targets are chosen from the cast of personas through a process of comparing goals and assigning a hierarchy of priority based on how broadly each persona's goals encompass the goals of other personas. A process of designating persona types determines the amount of influence each persona has on the eventual form and behavior of the design.

A detailed discussion of persona and goal development can be found in Chapter 5.

Requirements Definition

Design methods employed by teams during the Requirements Definition phase provide the much-needed connection between user and other models and the framework of the design. This phase employs scenario-based design methods with the important innovation of focusing the scenarios not on user tasks in the abstract, but first and foremost on meeting the goals and needs of specific user personas. Personas provide an understanding of which tasks are truly important and why, leading to an interface that minimizes necessary tasks (effort) while maximizing return. Personas become the main characters of these scenarios, and the designers explore the design space via a form of role-playing.

For each interface/primary persona, the process of design in the Requirements Definition phase involves an analysis of persona data and functional needs (expressed in terms of objects, actions, and contexts), prioritized and informed by persona goals, behaviors, and interactions with other personas in various contexts.

This analysis is accomplished through an iteratively refined **context scenario** that starts with a "day in the life" of the persona using the product, describing high-level product touch points, and thereafter successively defining detail at ever-deepening levels. In addition to these scenario-driven requirements, designers consider the personas' skills and physical capabilities as well as issues related to the usage environment. Business goals, desired brand attributes, and technical constraints are also considered and balanced with persona goals and needs. The output of this process is a **requirements definition** that balances user, business, and technical requirements of the design to follow.

Framework Definition

In the Framework Definition phase, designers create the overall product concept, defining the basic frameworks for the product's behavior, visual design, and — if applicable — physical form. Interaction design teams synthesize an **interaction framework** by employing two other critical methodological tools in conjunction with context scenarios. The first is a set of general **interaction design principles** that provide guidance in determining appropriate system behavior in a variety of contexts. Chapters 2 and 3 and the whole of Part II are devoted to high-level interaction design principles appropriate to the Framework Definition phase.

The second critical methodological tool is a set of **interaction design patterns** that encode general solutions (with variations dependent on context) to classes of previously analyzed problems. These patterns bear close resemblance to the concept of architectural design patterns first developed by Christopher Alexander, and more recently brought to the programming field by Erich Gamma, et al. Interaction design patterns are hierarchically organized and continuously evolve as new contexts arise. Rather than stifling designer creativity, they often provide needed leverage to approach difficult problems with proven design knowledge.

After data and functional needs are described at this high level, they are translated into design elements according to interaction principles and then organized, using patterns and principles, into design sketches and behavior descriptions. The output of this process is an **interaction framework definition**, a stable design concept that provides the logical and gross formal structure for the detail to come. Successive iterations of more narrowly focused scenarios provide this detail in the Refinement phase. The approach is often a balance of top-down (pattern-oriented) design and bottom-up (principle-oriented) design.

When the product takes physical form, interaction designers and industrial designers begin by collaborating closely on various **input vectors** and approximate **form factors** the product might take, using scenarios to consider the pros and cons of each. As this is narrowed to a couple of options that seem promising, industrial designers begin producing early physical prototypes to ensure that the overall interaction concept will work. It's critical at this early stage that industrial designers not go off and create concepts independent of the product's behavior.

As soon as an interaction framework begins to emerge, visual interface designers produce several options for a **visual framework**, which is sometimes also referred to as a **visual language strategy**. They use brand attributes as well as an understanding of the overall interface structure to develop options for typography, color palettes, and visual style.

Refinement

The **Refinement phase** proceeds similarly to the Framework Definition phase, but with increasing focus on detail and implementation. Interaction designers focus on task coherence, using **key path** (walkthrough) and **validation scenarios** focused on storyboarding paths through the interface in high detail. Visual designers define a system of type styles and sizes, icons, and other visual elements that provide a compelling experience with clear affordances and visual hierarchy. Industrial designers, when appropriate, finalize materials and work closely with engineers on assembly schemes and other technical issues. The culmination of the Refinement phase is the detailed documentation of the design, a **form and behavior specification**, delivered in either paper or interactive media as context dictates. Chapter 6 discusses in more detail the use of personas, scenarios, principles, and patterns in the Requirements Definition, Framework Definition, and Refinement phases.

Development Support

Even a very well-conceived and validated design solution can't possibly anticipate every development challenge and technical question. In our practice, we've learned that it's important to be available to answer developers' questions as they arise during the construction process. It is often the case that as the development team prioritizes their work and makes trade-offs to meet deadlines, the design must be adjusted, requiring scaled-down design solutions. If the interaction design team is not available to create these solutions, developers are forced to do this under time pressure, which has the potential to gravely compromise the integrity of the product's design.

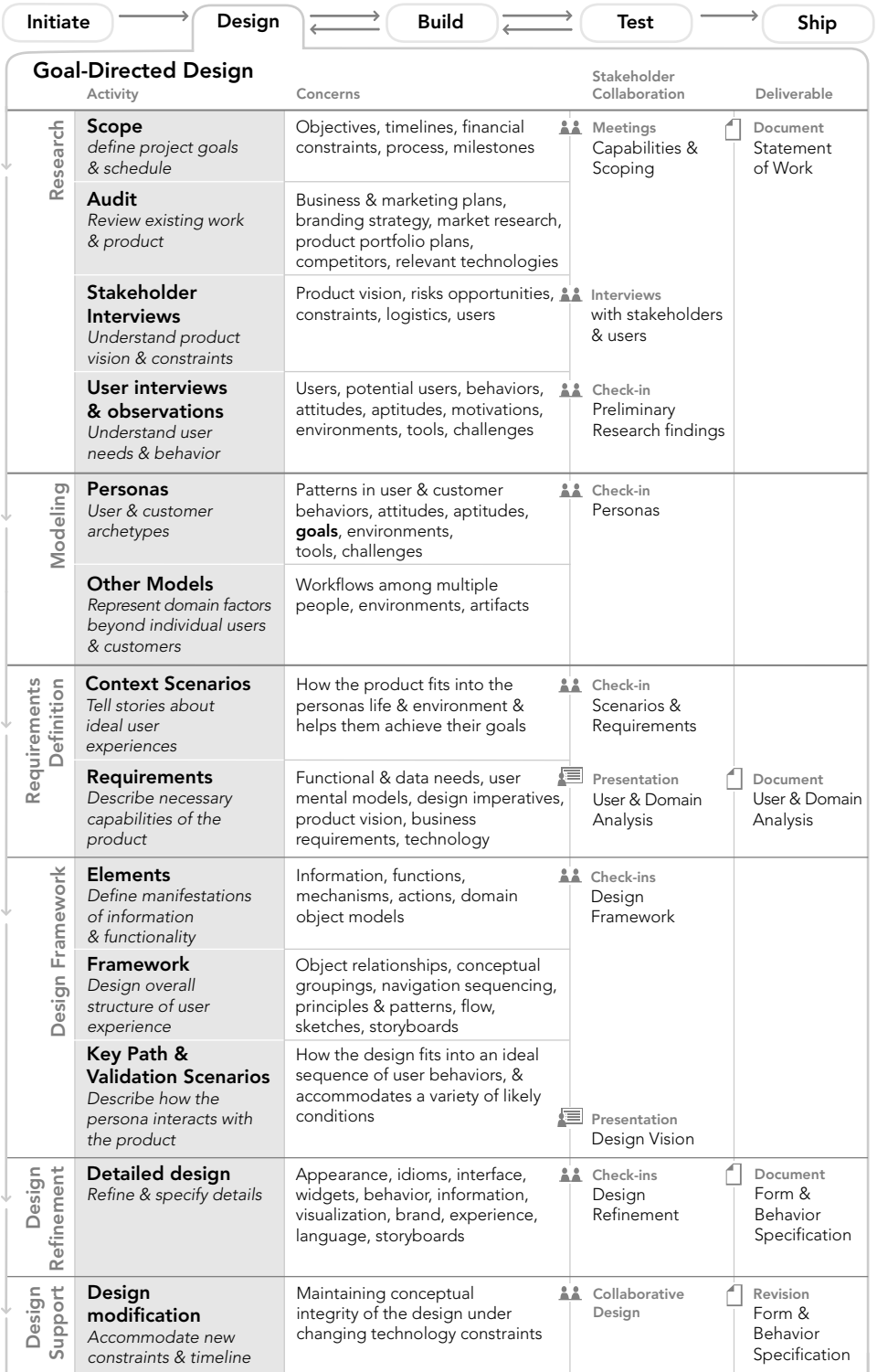

Figure 1-6 A more detailed look at the Goal-Directed Design process.

Goals, not features, are the key to product success

Developers and marketers often use the language of features and functions to discuss products. This is only natural. Developers build software function by function, and a list of features is certainly *one way* to express a product's value to potential customers (though this is clearly limiting, as well). The problem is that these are abstract concepts that only provide limited insight into how human beings can be effective and happy while using technology.

Reducing a product's definition to a list of features and functions ignores the real opportunity — orchestrating technological capability to serve human needs and goals. Too often the features of our products are a patchwork of nifty technological innovations structured around a marketing requirements document or organization of the development team with too little attention paid to the overall user experience.

The successful interaction designer must maintain her focus on users' goals amid the pressures and chaos of the product-development cycle. Although we discuss many other techniques and tools of interaction in this book, we always return to users' goals. They are the bedrock upon which interaction design should be practiced.

The Goal-Directed process, with its clear rationale for design decisions, makes collaboration with developers and businesspeople easier, and ensures that the design in question isn't guesswork, the whim of a creative mind, or just a reflection of the team members' personal preferences.

Interaction design is not guesswork.

Goal-Directed Design is a powerful tool for answering the most important questions that crop up during the definition and design of a digital product:

- Who are my users?
- What are my users trying to accomplish?
- How do my users think about what they're trying to accomplish?
- What kind of experiences do my users find appealing and rewarding?
- How should my product behave?
- What form should my product take?
- How will users interact with my product?
- How can my product's functions be most effectively organized?
- How will my product introduce itself to first-time users?

- How can my product put an understandable, appealing, and controllable face on technology?
- How can my product deal with problems that users encounter?
- How will my product help infrequent and inexperienced users understand how to accomplish their goals?
- How can my product provide sufficient depth and power for expert users?

The remainder of this book is dedicated to answering these questions. We share tools tested by years of experience with hundreds of products that can help you identify key users of your products, understand them and their goals, and translate this understanding into effective and appealing design solutions.