

Chapter 1

Goal-Directed Design

Our book has a simple premise: If achieving the user's goals is the basis of our design process, the user will be satisfied and happy. If the user is happy, he will gladly pay us money (and recommend that others do the same), and then we will be successful as a business.

On the surface, this premise sounds quite obvious and straightforward: Make the user 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 either happy or successful — or both?

Digital Products Need Better Design Methods

Most digital products today *emerge* from the development process like a monster emerging from a bubbling tank. Developers, instead of planning and executing with their users in mind, end up creating technological solutions over which they ultimately have little 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*. The authors propose a somewhat more detailed definition:

- ✓ Understanding the user's wants, needs, motivations, and contexts
- ✓ Understanding business, technical, and domain requirements and constraints
- ✓ Translating this knowledge into *plans* for artifacts (or artifacts themselves) whose form, content, and behavior is useful, usable, and desirable, as well as economically viable and technically feasible

This definition applies across all design disciplines, although the precise focus on form versus content versus behavior varies by design discipline.

When performed using the appropriate methods, design can provide the missing human connection in technological products. But clearly, the current approach to the design of digital products either isn't working or isn't happening as advertised.

The design of digital products today

Most digital products are built from an engineering point of view. True, marketing departments are sometimes able to provide requirements, but these often have little to do with what users actually *need* and have more to do with following the competition, providing feature checklists to

6 Part I: Bridging the Gap

IT departments, or making guesses based on user surveys or customer wish lists. None of these approaches take into account the users' *goals* in any systematic fashion. We'll soon see why goals are so important.

Developers have a different set of imperatives than users, centering on technology and engineering methodology. Meanwhile, marketing departments focus on what drives press attention, on feature lists, and on what people *say* they will buy. Users, when asked direct questions about the products they use, tend to focus on low-level tasks—contrary to what you might suspect, few users are consciously aware of or are able to clearly articulate their goals.

The result of these approaches is, unfortunately, software that irritates, reduces productivity, and fails to meet user needs. Figure 1-1 shows the evolution of the software development process, and where, if at all, design has 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 bad interactions—“lipstick on the pig,” as one of the authors' clients referred to it. The design process, as we will soon discuss, needs to *precede* coding and testing to ensure that products truly meet the needs of users.

Evolution of the Software Development Process

1. Originally, programmers did it all:

In the early days of the software industry, smart programmers dreamed up useful software, wrote it, and even tested it on their own. But as their businesses grew, the software business and software products became more complicated.



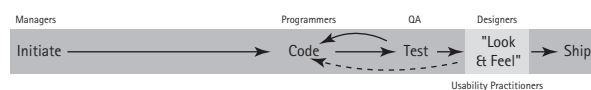
2. Managers brought order:

Inevitably, professional managers were brought in. Good product managers understand the market and competitors. They define software products by creating requirements documents. Often, however, requirements are little more than a list of features, and managers find themselves having to give up features in order to meet schedules.



3. Testing and design became separate steps:

As the industry matured, testing became a separate discipline and a separate step in the process. In the move from command-line to graphical user interface, design and usability also became involved in the process, though often only at the end, and often only affecting visual presentation. Today, common practice includes simultaneous coding and design followed by bug and user testing and then revision.



4. Design must precede the programming effort:

A goal-directed approach to software development means that all decisions proceed from a formal definition of the user and his or her goals. Definition of the user and user goals is the responsibility of the designer—thus design must precede programming.

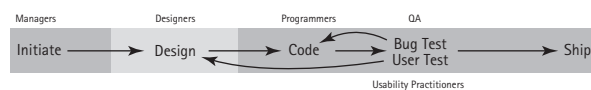


Figure 1-1: The evolution of the software development process. Today, design is often an afterthought. It should, instead, happen before any coding or testing begins.

Here are a few examples of why focusing on technology, markets, and tasks alone (instead of designing for users and their goals) results in the kind of software we've all grown to despise.

SOFTWARE IS RUDE

Software is often rude to the user. It blames the user for making mistakes that are not the user's fault, or should not be. Error message boxes like the one in Figure 1-2 pop up like weeds announcing that the user has failed yet again. These messages also demand that user acknowledge his failure by agreeing: OK.



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? Why do we care? And what are we OKing, anyway? It is not OK that the program failed!

Software frequently interrogates the user, peppering him with a string of terse questions that he is 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.

SOFTWARE MAKES UNWARRANTED ASSUMPTIONS

Software too frequently assumes that its user is computer-literate. For example, when a user is finished editing a document, he closes it, and the program asks if he wants to save it. The technology behind this issue is not trivial. It has to do with the capability of the CPU to directly address information stored on random-access magnetic media — but how is the novice user to know that?

SOFTWARE IS OBSCURE

Software is frequently obscure, hiding meaning, intentions, and actions from the user. Programs often express themselves in incomprehensible jargon that cannot be fathomed by normal users (“How many stop bits?”) and sometimes even by experts (“Please specify IRQ.”).

Features are hidden behind a veil of menus and dialogs and windows. How can the user know that the answer lies in the help system if he can't find the help system? Even when the user finds the right dialog, he might find it populated with terse abbreviations, obscure commands, and inscrutable icons.

More frequently than you might think, software demands that its users answer tough questions before telling them the effects their answers might have. For example, how can a user possibly decide between a full installation, custom installation, and laptop installation if he isn't told what each of them means in terms of functionality as well as disk space?

SOFTWARE EXHIBITS INAPPROPRIATE BEHAVIOR

If a 10-year-old child behaved like some software programs, he'd be sent to his room without supper. Programs forget to shut doors behind themselves, 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.

8 Part I: Bridging the Gap

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.

So what, then, is the real problem here? Why does the technology industry have such a problem with design of interactive products?

We're ignorant 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.

We have a conflict of interest

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 those designing a product are not the same people building it. It simply isn't possible for a programmer to advocate for the user, the business, and the technology at the same time.

We lack 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, 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 sport in excess of 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.

The current process of determining what software will do and how it will communicate with the user is today closely intertwined with its construction. Programmers, deep in their thoughts of algorithms and code, “design” user interfaces the same way that miners “design” the landscape with their cavernous pits and enormous tailing piles. The software interface design process alternates between the accidental and the non-existent.

Many programmers today embrace the notion that integrating users directly into the programming process on a frequent basis—weekly, or sometimes even daily—can solve 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. Users, 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.

To understand how to create a workable process that brings user-centered design to software, we need 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, and 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: feasibility, viability, and desirability (see Figure 1-3). If any one of these three foundations is significantly weaker than the others 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 prepurchase 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.

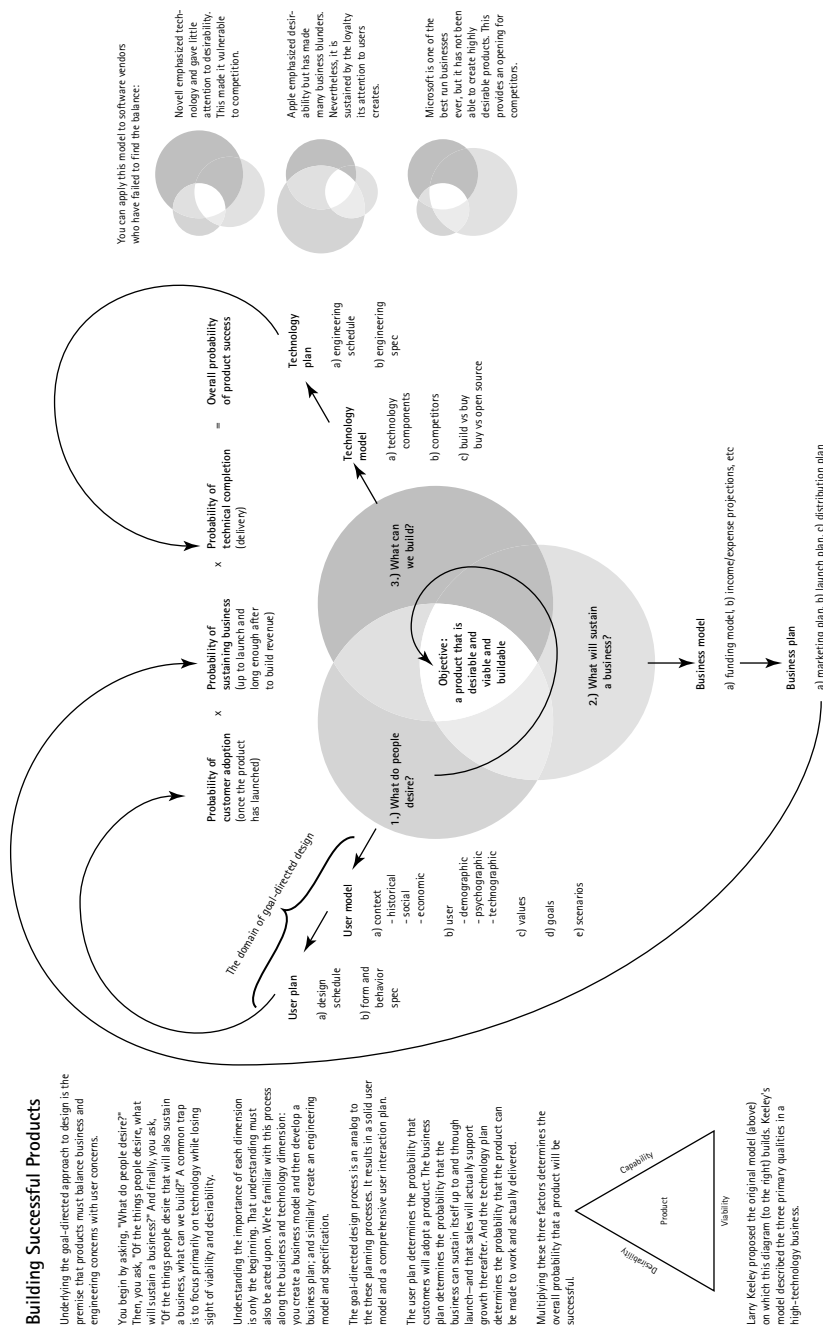


Figure 1-3: Building successful digital products. Expanding on Keeley's triangle (left), the center diagram indicates the three major processes that need to be followed in tandem to create successful technology products. This book addresses the first and foremost issue: how to create a product people will desire.

Planning and Designing Behavior

The planning of complex digital products, especially ones that interact directly with humans, requires a significant up-front effort by professional designers, just as the planning of complex physical structures that interact with humans require a significant up-front 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 a matter of aesthetic choice, but 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 that addresses the *goals* (Rudolf, 1998) of users: **Goal-Directed Design**. To understand Goal-Directed Design, we first need to better understand human 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? 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.

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 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.

12 Part I: Bridging the Gap

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
- ✓ Slow users down so they don't get an adequate amount of work done
- ✓ Prevent fun and/or bore users with navigational tedium

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 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 after coding has begun (and certainly not after it has ended).

Finally, when companies *do* focus on the users, they 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

Goals are not the same as tasks. A goal is an end condition, whereas a task is an intermediate step that helps to reach a goal. Goals motivate people to perform tasks. It is very important not to confuse tasks with goals, but it is easy to mix them up.

Luckily, there is an easy way to tell the difference between tasks and goals. Goals are driven by human motivations, which change very slowly, if at all, over time. Tasks are transient, based almost entirely on the technology at hand. For example, when traveling from St. Louis to San Francisco, a person's *goals* are likely to be speed, comfort, and safety. In 1850, a settler would have made the journey in a covered wagon; and, in the interest of safety, he would have brought along his trusty rifle. Traveling from St. Louis to San Francisco today, a businessman makes the journey in a jet aircraft; and, in the interest of safety, he is required to leave his firearms at home. The goals remain unchanged, but the tasks have so changed with the technology that they are, in some respects, in direct opposition.

Looking through the lens of goals allows you to leverage technology to eliminate irrelevant tasks. For example, if you want to get to work in the morning, your goal is to get there as quickly and safely as possible. With today's technology, that means executing the tasks of getting in your car and braving traffic, or perhaps waiting for a train and walking the rest of the way. In the Star Trek future, you can throw a switch and materialize in your office via transporter beam. Looking at goals can similarly help designers eliminate tasks that technology renders unnecessary for humans to perform.

Many developers and usability professionals approach the design of interfaces by asking, “What are the tasks?” Although this may get the job done, it won’t produce the best solution possible, and often it won’t satisfy the user. When designing an interface, task analysis is useful, but only after user goals have been analyzed. Design based solely on 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. Asking, “What are the user’s goals?” lets us see through the confusion and create more appropriate and satisfactory design.

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 (1990) 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 the 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, ones 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 5 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 logical 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 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 broader roles 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 over-specialized: 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.

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.

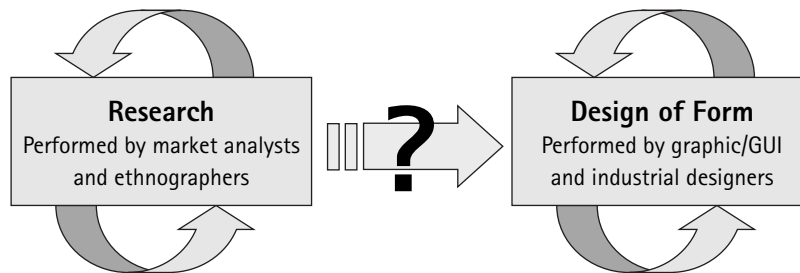


Figure 1-4: A problematic design process. Traditionally, research and design have been separated, with each activity handled by specialists. *Research* has, until recently, referred primarily to market research, and *design* is too often limited to visual design or skin-deep industrial design. More recently, **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.

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 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 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 explicit, systematic processes for defining user models, establishing design requirements, and translating those into a high-level interaction framework (see Figure 1-5).

Knowledge about users must be synthesized into a model that leverages user data *as a design tool*. Other models, such as workflows and environmental contexts, are also important and useful, but appropriate user models are singularly critical. After user models are created, the usage patterns, mental models, and user goals captured by them can be systematically mapped to an interaction framework that also addresses the business and technical imperatives that are captured in other models.

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.

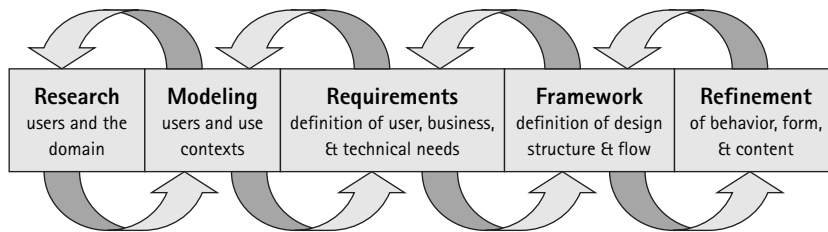


Figure 1-5: Bridging the gap between research and design. Three primary activities close the gap. A process of modeling that synthesizes research results into design tools, a process of synthesizing and defining requirements from these models, and a process of translating the knowledge captured in the models and requirements into a design framework that reflects the goals and needs of users, while also addressing business and technical imperatives.

A process overview

Goal-Directed Design combines techniques of ethnography, stakeholder interviews, market research, product/literature reviews, 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 (see Figure 1-6). This process can be roughly divided into five phases: *Research*, *Modeling*, *Requirements Definition*, *Framework Definition*, and *Refinement*. These phases follow the five component activities of interaction design identified by Gillian Crampton Smith and Philip Tabor (1996)—understanding, abstracting, structuring, representing, and detailing—with a greater emphasis on modeling user behaviors and defining system behaviors.

The remainder of this chapter provides a high-level view of the five phases of Goal-Directed Design. Chapters 4, 5, and 6 provide a more process-oriented overview of the methods involved in each of these phases.

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, 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 **usage 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 to professional roles; for consumer products, they tend to correspond to lifestyle choices. Usage patterns and the goals associated with them drive the creation of **personas** in the Modeling phase. Market research helps select and filter for valid personas that fit corporate business models. Stakeholder interviews, literature reviews, and product audits deepen the designers' understanding of the domain and elucidate business goals 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, usage 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 behavior patterns, 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.

Possible user persona type designations include:

- ✓ **Primary:** the persona's needs are sufficiently unique to require a distinct interface form and behavior
- ✓ **Secondary:** a primary interface serves the needs of the persona with a minor modification or addition
- ✓ **Supplemental:** the persona's needs are fully satisfied by a primary interface
- ✓ **Served:** the persona is not an actual user of the product, but is indirectly affected by it and its use
- ✓ **Negative:** the persona is created as an explicit, rhetorical example of whom *not* to design for

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 provides the much-needed connection between user and other models and the framework of the design. This phase employs scenario-based design methods (Carroll, 1995), 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 term of objects, actions, and contexts), prioritized and informed by persona goals, behaviors, and interactions with other personas in various contexts.

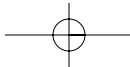


Figure 1-6: The Goal-Directed Design process

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. As this iteration occurs, both business goals 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, 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, like their visual design counterparts (see Chapter 19), provide guidance in determining appropriate system behavior in a variety of contexts. Chapters 2 and 3 and the whole of Section 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 developed by Christopher Alexander (1977). 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.

REFINEMENT

The **Refinement** phase proceeds similarly to the Framework Definition phase, but with greater focus on task coherence, using **key path** (walkthrough) and **validation scenarios** focused on storyboarding paths through the interface in high detail. 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.

Goals, not features, are the key to product success

Programmers and engineers—people who are intrigued by technology—share a strong tendency to think about products in terms of functions and features. This is only natural, as this is how developers build software: function by function. The problem is that this isn’t how users want to use it.

The decision about whether a feature should be included in a product shouldn't rest on its technological underpinnings. The driving force behind the decision should never be simply that we have the technical capability to do it. The deciding factor should be whether that feature directly, or indirectly, helps to achieve the goals of the user while still meeting the needs of the business.

The successful interaction designer must be sensitive to 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 is practiced.

The Goal-Directed process, with its clear rationale for design decisions, makes persuading engineers easier, keeps marketing and management stakeholders in the loop, and ensures that the design in question isn't just guesswork, or a reflection of the team members' personal preferences.



Interaction design is not guesswork.

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

- ✓ How do I find out who my users are?
- ✓ How do I learn what my users are trying to accomplish?
- ✓ 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 and controllable face on technology?
- ✓ How can my product deal with user problems?
- ✓ How will my product help infrequent users become more expert?
- ✓ How can my product provide sufficient depth for expert users?

We will help you to answer these questions in the remainder of this book. You will get tools you need to identify key users of your products, understand them and their goals, and translate this understanding into winning design solutions using Goal-Directed Design.