CHAPTER

Scope, Rigor, Complexity, and Project Perspectives

3

Highlights

- The need for rigor in a project.
- Complexity and risk aversion as influences on the need for rigor.
- Scope of delivery.

3.1 INTRODUCTION

We are going to get into UX lifecycle activities soon, but first let's discuss a few parameters that define a design situation and the impact they have on lifecycle choices.

3.1.1 Rigor and Scope: Project Parameters that Determine Process Choices

In any design situation, you need to make choices. Even within an agile UX process, no one UX lifecycle activity or method is one size fits all. Your job is to adopt and adapt UX lifecycle activities, methods, and techniques to specific project conditions (Section 2.5.2). Most of the high-level process choices hinge on two major determiners, rigor and scope.

3.2 RIGOR IN A UX METHOD OR PROCESS

3.2.1 What Is Rigor?

The rigor of a UX design lifecycle activity, method, or technique is determined by the degree of formality, thoroughness, precision, and accuracy with which you perform all the steps. It is also about how meticulously you maintain and document completeness and purity of data—especially usage research and UX evaluation data—collected.

Completeness. Completeness entails thoroughness of methods, which means covering every step in full. Completeness also applies to the usage research and evaluation data. This attention to detail helps designers touch all the bases and fill in all the blanks so that no functions or features are forgotten.

Purity. Purity means being as accurate as possible in your data; in particular, it involves not allowing new spurious "data" to creep in. For example, for high data purity, designer insights or conjectures should be tagged with metadata, identifying and distinguishing them from actual user input.

We refer to the methods for maintaining this completeness and purity of data as rigorous methods.

Example: Building a House: A Possible Need for Rigor

Using an example of house building, imagine the designer (architect) capturing a data item during data elicitation in the predesign Understand Needs activity (Section 2.3.1). In this example, that translates to the architect talking with the potential homeowner, where he learns that the owner wants a whole-house backup generator next to the house.

Later, as the architect in his studio becomes immersed in the design situation to understand the requirements and constraints for the building, he realizes the need to check on possible constraints as to the location of the generator. On a visit to the county building inspector's office, he captures the necessary information regarding the minimum distance between a backup house generator and both the outer edge of the house and the gas main for the house.

In a low-rigor approach, this constraint would just be captured as a simple note or even just kept in the designer's head as "the generator needs to be at least four feet from the gas main and at least four feet from the outer wall of the house." In a rigorous approach, the same constraint would be captured with all the relevant metadata (e.g., identifying the applicable safety code or local building conventions) in a structure that makes it easy to query and trace back, if necessary. It could be something along the lines of:

Type: External constraint
Description: Any external device with an engine, such as a backup generator, shall be placed at least four feet from all structures attached to the foundation of the house, including decks, porches, awnings, and uncovered decks.
Source: Middleburg County building code VT-1872, page 42.
Verified by: Ms. Jane Designer
Date noted: June 13, 2016

This kind of rigor due to specificity and completeness is not always necessary during the building of a small house, but it could be important (or maybe even required by law) in a commercial building project.

3.2.2 Complexity as an Influence on the Need for Rigor

3.2.2.1 The system complexity space

One big reason you can't define one set of methods for designing all systems is that there is a spectrum of system and product types with a broad range of risk versus needs for rigor in lifecycle activities and methods. In this and the next few sections, we look at some possibilities within this spectrum.

In Fig. 3-1, we show a system complexity space defined by the dimensions of interaction complexity and domain complexity (defined in the next two sections). While there undoubtedly are other ways to partition the space, this approach serves our purpose.

System complexity space

A two-dimensional space defined by the dimensions of interaction complexity and domain complexity, depicting a spectrum of system and product types with a broad range of risk versus needs for rigor in lifecycle activities and methods (Section 3.2.2.1).

3.2.2.2 Interaction complexity

Interaction complexity, represented on the vertical axis, is about the intricacy or elaborateness of user actions, including the difficulty of cognitive actions, necessary to accomplish tasks with the system.

Low interaction complexity usually corresponds to systems that support smaller tasks that are generally easy to do, such as ordering flowers from a website. High interaction complexity is usually associated with larger and more difficult tasks, often requiring special skills or training, such as manipulating a color image with Adobe Photoshop (a high-functionality software application for managing and processing large collections of images and photographs).

3.2.2.3 Domain complexity

On the horizontal axis, we show work domain complexity, which is about the degree of intricacy and the technical, or possibly esoteric, nature of the corresponding field of work. Convoluted and elaborate mechanisms for how parts of the system work and communicate within the ecology of the system contribute to domain complexity.

Ecology

In the setting of UX design, the ecology is the entire set of surrounding parts of the world, including networks, other users, devices, and information structures, with which a user, product, or system interacts (Section 16.2.1).

User work in domain-complex systems is often mediated and collaborative, with numerous "hand offs" in a complicated workflow containing multiple dependencies and communication channels, along with compliance rules, regulations, and exceptions in the way work cases are handled. Examples of high work-domain complexity include systems for geological fault analysis for earthquake prediction, global weather forecasting, and complete healthcare systems.

Low work-domain complexity means that the way the system works within its ecology is relatively simple. Examples of low domain complexity include that

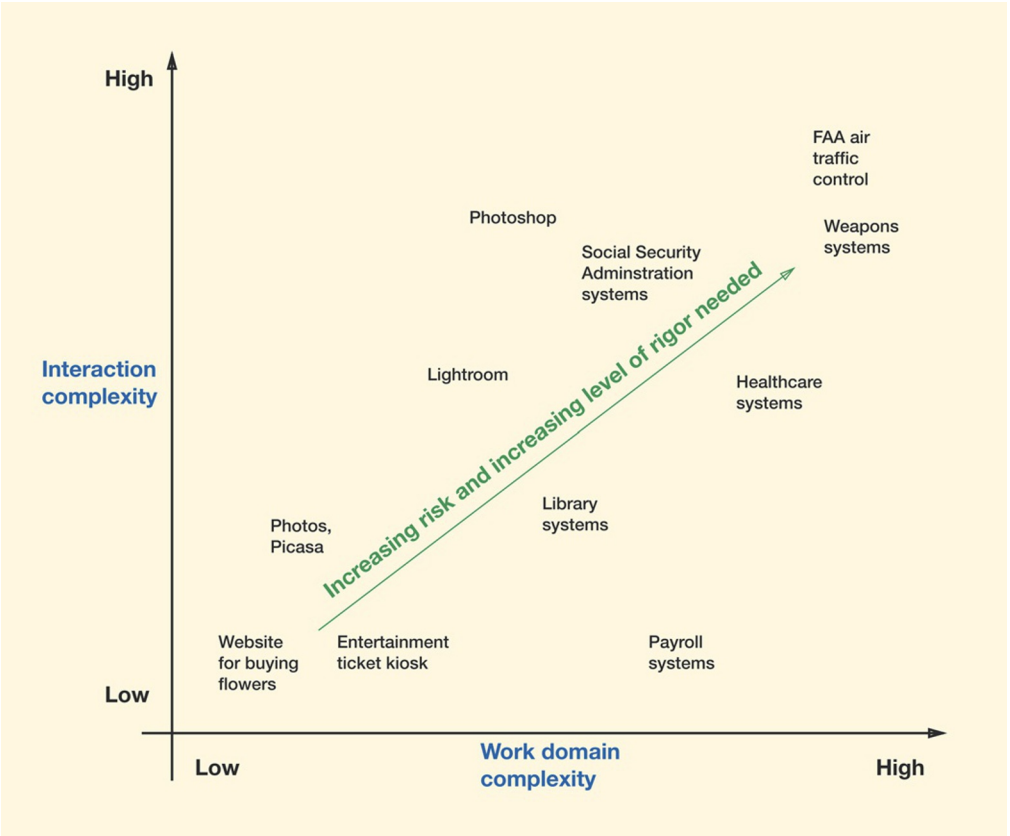

Fig. 3-1

The system complexity space: interaction complexity versus domain complexity.

same website for buying flowers and a simple personal calendar management application.

3.2.2.4 The system complexity space quadrants

Simple interaction, simple work domain. The simplest quadrant is in the lower left corner of Fig. 3-1, where both interaction and work domain are the least complex. This quadrant contains smaller websites, certain interactive applications, and many commercial products. Just because this is the simple-simple quadrant, however, does not mean that the products are overly simple; the products of this quadrant can still be sophisticated.

There is an abundance of relatively simple systems in the world. Some, but not all, commercial software products are domain-simple and interaction-simple,

at least relative to large systems of other types. Again, the website for ordering flowers is a good example of this quadrant. Interaction is very simple; just one main task involving a few choices and the job is done. Work domain complexity is also relatively simple because it involves only one user at a time and the workflow is almost trivial. Many apps on mobile devices, a significant segment of the commercial product market, are simple-interaction and simple-domain.

Although emotional impact factors do not apply to every system or product in this quadrant, emotional impact can be very important here, especially with respect to factors such as aesthetics or fun or joy of use. The smartphone and the personal mp3 music player are good examples of commercial products in this quadrant that have emotional impact issues, including meaningfulness (emotional impact within long-term usage).

Because systems and commercial products in this quadrant have less complexity, they generally require less rigor.

Complex interaction, complex work domain. As you move all the way across the diagonal in Fig. 3-1, you reach the upper right quadrant where you find interaction-complex and domain-complex systems, which are usually large and complicated.

Serious systems. Serious systems live in the far upper right corner of the system complexity space. An example is an air traffic control system used to decide the landing orders for an incoming airliner. An air traffic control system has enormous domain and interaction complexity, with workflow and collaboration among a large number of work roles and user types. Another defining example for this quadrant is a large system for the Social Security Administration.

For large domain-complex systems, such as military weapons systems, you are most likely to encounter resistance to innovation. Radical designs are not always welcome; conformity can be thought more important. Users and operators, in some cases, commit operations to habit and perform tasks with learned behavior even if there are better ways. Work practice change must be approached with caution.

This sector of serious systems within the system complexity space usually has little, if anything, to do with emotional impact factors such as aesthetics, fun, or joy of use.

Enterprise systems. Among the kinds of systems one sees in the rest of this quadrant are the so-called enterprise systems, large information systems used within organizations that have typically been forgotten in discussions of usability and user experience. Gillham (2014) puts it this way: “Most big businesses globally are locked into some kind of enterprise technology. Unfortunately, such systems

Risk

The danger or possibility of things going wrong, of features or requirements being missed, or the result not meeting the needs of users; possibility of not accommodating legacy needs or not complying with legal or safety constraints (Section 3.2.4).

are not only fiendishly difficult to install and maintain, but often equally challenging for the workforce to use. When the stakes are so high, why is the user experience of enterprise systems so bad?"

The highest need for rigor. The domain-complex and interaction-complex quadrants are where you find the highest risks and, therefore, the highest need for rigor to manage risk. These project conditions include:

- When you have the greatest regulatory compliance requirements.
- When the importance of error avoidance in usage is highest (e.g., in mission-critical systems such as for air traffic control or for military weapons control).
- When you cannot risk getting these things wrong and the cost of failure is unacceptable.
- When you have a contractual obligation for high rigor.

Because of their sheer size and this need for rigorous processes, these systems are typically among the most difficult and expensive to design and develop.

Complex interaction, simple work domain. In the upper left quadrant of Fig. 3-1, one of the "in-between" quadrants, we see interaction-complex and domain-simple systems. It is typical of an interaction-complex system to have a large volume of functionality resulting in a large number and broad range of complex user tasks. The older-style digital watch (not a smart watch) is a lightweight but good example. Its interaction complexity often stems from a large variety of modal settings using overloaded and unlabeled push buttons. The domain, however, is still simple, being mainly about "what time is it?" Workflow is trivial; there is only one work role set within a simple system ecology.

Attention in this quadrant is needed for interaction design—myriad tasks, screen layouts, user actions, and even metaphors. Rigorous formative evaluation may be needed for the consistency and usability of the conceptual design and the detailed interaction design. The focus of modeling will be on tasks—task structure and task interaction models—and perhaps the artifact model, but not much attention will be given to work roles, workflow, or most of the other models of Chapter 9.

Artifact model

A representation showing how users employ, manipulate, and share key tangible objects (physical or electronic work practice artifacts) as part of the flow in their work practice (Section 9.8).

For simple work domains, regardless of interaction complexity, usage research rarely results in learning something new that can make a difference in informing design. So less rigor will suffice leading up to design creation. For design creation, however, complex interaction requires careful and systematic brainstorming, ideation, and sketching, plus iterative evaluation and refinement as well as attention to emotional impact factors.

Simple interaction, complex work domain. In the lower right quadrant of Fig. 3-1, the other "in-between" quadrant, we see interaction-simple and domain-complex systems. In this quadrant, user tasks are relatively simple and easy to understand, so less attention to task descriptions is needed. The key effort for users in this quadrant is understanding the domain and its often esoteric work practice. Designers need rigorous usage research to focus on conceptual design and easy-to-understand user models of how the system works. Once that is understood, the interaction is relatively straightforward for users. Tax preparation software for average households is a good example because the underlying domain is complex but the data entry into forms can be simplified to a step-by-step process.

Sometimes systems for managing libraries, shown in the middle of the work domain complexity scale near the bottom of Fig. 3-1 fall into the simple-interaction, complex-work-domain quadrant. Typical library systems have low interaction complexity because the range of tasks and activities for any one user is fairly circumscribed and straightforward and the complexity of any one user task is low. Therefore, for a library system, for example, you do not need to exercise much rigor in the modeling of tasks.

However, a full library system has considerable domain complexity. The work practice of library systems can be esoteric and most UX designers will not be knowledgeable in this work domain. For example, special training is needed to handle the surprisingly important and highly controlled small details in cataloging procedures. Therefore, a rigorous approach to usage research may be warranted.

Gradations within the system complexity space. Some systems or products clearly fall into one quadrant or the other, but other projects can have fuzzy quadrant boundaries. Websites, for example, can belong to multiple quadrants, depending on whether they are for an intranet system for a large organization, a very large e-commerce site, or just a small site for sharing photographs. Products such as a printer or a camera are low in domain complexity but can have medium interaction complexity.

Healthcare systems typically cross system complexity space quadrants. Some parts internal to a small doctor's office might be relatively simple. But large healthcare systems that integrate medical instrumentation, health record databases, and patient accounting have complex work domains. Similarly, machines in a patient's hospital room can have a fairly broad range of technical tasks and activities, giving them relatively high interaction complexity.

The healthcare domain is also saddled with more than its share of regulation, paperwork, and compliance issues, plus legal and ethical requirements—all of which lead to high work domain complexity and a high need for rigor in UX lifecycle activities and methods.

3.2.3 Domain Familiarity as an Influence on the Need for Rigor

Even if a domain is not complex in absolute terms, if it is unfamiliar to the UX designer, it will seem complex, at least at first. Familiarity with the target domain is part of the designer's cognitive scaffolding for understanding the design problem. The way to achieve this domain familiarity is by initially using UX methods with high rigor.

Our example of house building (Section 3.2.1) is in a domain that is at least somewhat familiar to us all. Now, consider a domain that is more specialized and esoteric, the domain of financial portfolio analytics and management. If the designer is not very familiar with this domain, there is a greater need for capturing as much detail from users and usage as possible so they can go back and refer to it for clarification or education.

In my (Pardha) practice in this domain, I often encountered unfamiliar terminology. Or, sometimes users from different portfolio management firms described different practices and philosophies on investment in different ways, making it more difficult for me to maintain a unified understanding of the domain. During usage research data collection sessions with clients, users, and subject matter experts (SMEs), in order to maintain rigor, it was often important to capture metadata such as who said what and where they worked because that detail was important for later analyses to contextualize what was said.

3.2.4 Risk Aversion Influences the Need for Rigor

Risk is the danger or possibility of things going wrong, of features or requirements being missed, or the result not meeting the needs of users; possibility of not accommodating legacy needs or not complying with legal or safety constraints.

As we hinted in Section 3.2.2.4, an important goal-related factor in choosing the level of rigor in the UX design lifecycle activities and methods is aversion to risk. There are cases where not getting the UX design right creates high risks, often because of requirements for things such as legacy accommodation, legal constraints, or safety concerns. *A legacy system is an old method, technology, computer system, or application program, of, relating to, or being a previous or outdated computer system with maintenance problems that date back possibly many years.*[1]

[1] https://en.wikipedia.org/wiki/Legacy_system

The less the tolerance for risks the more the need for rigor in the lifecycle activities, methods, and techniques. But of course, high rigor throughout the lifecycle process will add cost and time to complete.

3.2.4.1 The risk of data loss

The most significant data loss in the UX design process is a loss of completeness. For reasons of speed and economy, the data get condensed, summarized, and otherwise abbreviated. You can avoid data loss by recording (audio or video) every detail of user interviews and observations. But it is tedious and costly to transcribe the recordings and the result is a ton of text to wade through to separate the important stuff from all the noise and unimportant verbiage; it's almost never worth it.

So usage researchers often just take notes to summarize the main points to themselves. Although this technically introduces a loss of completeness, it doesn't necessarily reduce the quality or usefulness of the data. Raw data usually need a step of abstraction, anyway, to remove irrelevant detail and to highlight what is important. But you can take this abstraction too far. By being lazy or careless or not taking enough time, you start to lose data that could be important later in the overall process.

Abstraction

The process of removing extraneous details of something and focusing on its irreducible constructs to identify what is really going on, ignoring everything else (Section 14.2.8.2).

3.2.4.2 Risks associated with legal, safety, and compliance constraints

Systems such as air traffic control systems, healthcare and medical records systems, and financial systems have legal requirements with respect to public safety risks that must be taken very seriously. Designing a product for the financial industry requires a process that accounts for compliance to complex federal regulations built into the business process involved in using the product. Your lifecycle activities and methods need to incorporate sufficient rigor to ensure (and possibly even prove) that the system meets all those compliance checks.

3.2.5 The Stage of Development within Your Project as an Influence on the Need for Rigor

The stage of development within your project is another determiner of the need for rigor. All projects go through different "stages" over time. Regardless of method choices based on other project parameters, the appropriateness of a level of rigor and corresponding choices of UX methods and techniques for lifecycle activities will change as a project evolves through these stages of development. For example, early stages might demand a strong focus on rigorous usage research to obtain the most user, usage, and domain knowledge upfront.

Inspection (UX)

An analytical evaluation method in which a UX expert evaluates an interaction design by looking at it or trying it out, sometimes in the context of a set of abstracted design guidelines. Expert evaluators are both participant surrogates and observers, asking themselves questions about what would cause users problems and giving an expert opinion predicting UX problems (Section 25.4).

But there might be very little emphasis on rigorous evaluation in early stages. Spending large resources on early evaluation could be a waste because the design is still fluctuating. So for early stages, it might be better to use lightweight, rapid, and frequent evaluation methods. As an example, to evaluate an early conceptual design you might choose a quick design review.

In later stages of evaluation, to refine a now-stable design, you might move to UX inspection of a low-fidelity prototype and eventually to the more rigorous lab-based testing using a higher-fidelity prototype, increasing the amount and quality of data you need to retain in each step. It takes more rigor to keep track of this extra data.

3.2.6 Project Resources: Budgets, Schedules, and/or Personnel Capabilities are Determiners of Rigor

High rigor costs money and takes time. Slim budgets and short schedules are practical realities that can constrain your lifecycle activity and method choices and restrict the level of rigor you are able to provide.

Another important kind of resource is person power. How many people do you have, what project team roles can they fill, and what UX skills, experience, and training do they bring to the project?

UX professionals with extensive experience and maturity are likely to need less of the formal aspects of rigorous methods, such as thorough usage research or detailed UX evaluation goals and targets. For these experienced UX professionals, following the process in detail does not add much to what they can accomplish using their already internalized knowledge and honed intuition.

Product perspective

A design viewpoint in which the design target is a large organizational information system (Section 3.4.1).

Enterprise system perspective

A design viewpoint in which the design target is a personal object (a consumer product), such as a device or software app, that a user buys for private use. The product perspective is a consumer perspective (Section 3.4.1).

3.2.7 Being Rapid in Lifecycle Activities, Methods, and Techniques

While some methods are inherently more rigorous than others (e.g., lab-based evaluation compared to inspection methods), it is important to note that any method can be performed across a range of rigor. Applying an evaluation method (or any method) with more rigor can achieve more complete and more accurate results, but will take more time and is more costly. Similarly, by taking shortcuts you can usually increase speed and reduce costs of almost any UX evaluation method, but at the price of reduced rigor.

3.2.7.1 Not every project needs rigorous UX methods

For many projects, certainly in the commercial product perspective and often in the enterprise system perspective, high rigor isn't necessary, isn't worth the cost, or simply isn't possible given limited project resources.

In response, less rigorous UX methods and techniques have evolved in the literature and practice that are faster and less expensive but still allow you to get good results from your effort and resources.

3.2.7.2 Rapid methods are a natural result

When higher rigor is not required, the point of working at a lower level of rigor is to reduce lifecycle cost and time. Most of the time you can be less rigorous by abbreviating the "standard" rigorous methods and techniques, which means taking shortcuts, skipping unnecessary steps, and maintaining only the most important data. Or you can use rapid alternative methods that are inherently less rigorous.

3.2.7.3 Over time our need for rigor has diminished

As another factor, UX practice has matured and we have gotten better at our craft. As a result, we don't need to be as rigorous and thorough in most common UX situations. The plodding and burdensome fully rigorous process has been abandoned as an outlier. A rare project that requires full rigor will come with specifications about what to do and how to achieve it. Now the "standard" method for each lifecycle activity is just a "regular" process, which is a combination of a practical middle-of-the road rigor and some obvious ways to be efficient.

3.2.7.4 Rapidness principle: Work as rapidly as you can

Regardless of other factors, you should always seek the fastest way to do things. The principle underlying all choices of lifecycle activities, methods, and techniques is: *Go as fast as you can, subject to constraints imposed by project goals and the need for rigor.* Ries (Adler, 2011) says it is a misconception that if you work slowly and deliberately, you will produce a better product and you'll be able to fix problems as you go. The agile approach to both UX and SE (software engineering) has proven this misconception wrong.

Being economical and lightweight is now a way of life for a designer and is part of the foundation of agile processes (see next section). Even when you need a rigorous approach, you should be just doing what comes naturally in making smart choices and not wasting time or resources needed to do things that won't really contribute to your final design. For example, an otherwise rigorous approach could be done without one or more of the types of models (Chapter 9) if they are not essential to the project.

3.3 SCOPE OF DELIVERY

Our use of the term "scope" refers to how the target system or product is "chunked" in each iteration or sprint for delivery for agile implementation. UX roles deliver their designs to the SE people in chunks of a given size and the SE

role consumes the chunks of UX design as specifications for code, which it then implements and delivers as chunks of possibly some other size of functional software to clients and users.

In a large scope, chunks are composed of multiple features or even large portions of the system. In a small scope, synonymous with agility, chunks are usually comprised of one feature at a time. Even large and complex enterprise systems are being developed with small-scope approaches in agile software development these days. We, the UX designers, will always end up delivering our designs in chunks of a small scope within an agile UX process.

However, some early UX design work (for example, to establish the conceptual design) may require a larger scope. And, of course, a large scope does offer a nice structure for describing the UX lifecycle activities in this book.

Example: Large and Small Scope in Building a House

Here we return to that familiar setting, that of designing and building a house, to illustrate the trade offs between large and small scope in simple terms. By the time the full house is completed and delivered, a custom house designer should have addressed a range of user needs and issues, including:

- User lifestyle to be supported.
- User preferences, specifications, and requirements.
- Work flows of occupants.
- Zoning laws and other external constraints.
- Ecology of the house (the setting, the neighborhood).
- User/owner values (efficiency, greenness, footprint).
- Style (modern, colonial, Spanish).

Here's how scope plays out in terms of how you approach designing and building a house:

- Large scope: Design the whole house first and build it all before delivering it to the client.
- Small scope: Design one room (for example, the kitchen) first, build it, and deliver it to the client, and follow up with another room, say the living room, and so on, in a series of "increments" until the whole house is completed.

Large scope. Normally, in a large-scope approach, much of the construction is done in an "assembly line" manner for reasons of cost. The foundation people start and then the framing people do their thing. Then you bring in the electrician and he wires the whole house. Then you bring in the plumber and she installs plumbing in the whole house. Then you bring in the drywall person and he puts up all the interior walls, and so forth.

Small scope. For practical reasons, a small-scope approach is not used in house construction, but let's explore that possibility to see what the trade offs might be.

Suppose this house is being designed and built for an eccentric (and wealthy) software engineer and he wants to do an experiment to see if a small scope can

work for house building, too. As in the case of software, there would have to be some infrastructure built first to support the incremental features to come. In software, this might be essential parts of the operating system and other software services that each feature would call on. In house building, this would be laying the foundation, framing in the skeleton, and establishment of infrastructure shared among rooms (e.g., main breaker box for electrical service). That's always done first, anyway.

Then the client (the adventuresome software engineer) would ask for "delivery" of a feature (for example, the kitchen) in two weeks. The electrical work, plumbing, drywall installation, woodwork, painting, and so on would be done for just the kitchen. The client would show up and possibly even use the kitchen to cook something and then give feedback, and you would certainly learn things that could be used on subsequent rooms.

Then you have to get all these people back to do their thing for the next room, increasing the cost and time delays substantially. It would also drive the contractor crazy. So, if you wanted to do this as an experiment, you would have to pay significantly more. Therefore, while a small-scope approach is effective and efficient in UX and software engineering, it is not an efficient way to build a house.

3.4 THE COMMERCIAL PRODUCT PERSPECTIVE AND THE ENTERPRISE SYSTEM PERSPECTIVE

Most of this chapter has been about process parameters such as rigor, scope, and complexity. Another factor that influences your UX design approach is whether it comes from a commercial product perspective or an enterprise system perspective. Obviously, it isn't appropriate to use the same method to design a smartphone as you would, say, an enormous corporate resource management system.

3.4.1 The Commercial Product Perspective

We use the term "commercial product perspective" (or "product perspective" for short) to describe situations in which the target of our designs is a personal object (a consumer product), such as a device or software app, that a user buys for private use. The product perspective is a consumer perspective.

A product still can involve multiple users (e.g., people sharing music) and multiple activities (e.g., buying music, organizing music, etc.). These cases can be thought of as small systems within the product perspective; for example, a network of educational and entertainment devices connected to the cloud.

Activity-based interaction

Interaction in the context of one or more task thread(s), a set of, or possibly sequences of, multiple, overlapping, and related tasks, often involving more than one device in an ecology (Sections 1.6.2 and 14.2.6.4).

3.4.1.1 Single-user products

The usage context of a single-user product design perspective is usually quite narrow and simple. Examples include games designed to be played by one person, personal calendar applications, portable music players, and smartphones.

However, a single-user product does not necessarily have to be a single application used in isolation, but it can be set in a network of communicating applications supporting activity-based usage. An example is the coordinated use of a calendar, a contacts list, and email.

3.4.1.2 Multiuser collaborative products

Multiuser products are somewhat similar to systems in the enterprise system perspective in that usage involves multiple users and actors spanning multiple activities and there could be a flow of information or even usage artifacts. But the setting is different. Unlike the enterprise system perspective setting of an organization and the organizational goals, the multiuser product perspective setting is more like a user community with its own goals, which could be cooperative or competitive. Examples include families sharing music on a set of smartphones and family interactions with living room devices such as Amazon Echo.

3.4.2 The Enterprise System Perspective

We use the term "enterprise system perspective" to refer to situations where the work practice involves multiple users and actors spanning multiple activities, usually in an organizational setting. The goal of work practice in this perspective is to further the business goals of that organization. Organizations contain numerous and often widely different system usage roles, each contributing to a part of the overall work that gets accomplished. These types of products are typically not owned by a single user and are not as tangible or self-contained in their manifestation to the user. They can be somewhat abstract and disembodied from the people who use them.

Of course, some projects will have design targets that fall somewhere in between the two perspectives.

Now that we have discussed the issues of scope, rigor, complexity, and their impacts on process choices, we are now ready to talk about what it takes for the UX lifecycle to work in an agile context. For this we introduce the funnel model of UX design in the next chapter.