Chapter 13

INTERACTION DESIGN IN PRACTICE

Objectives

The main goals of the chapter are to accomplish the following:

- Describe some of the key trends in practice related to interaction design.
- Enable you to discuss the place of UX design in agile development projects.
- Enable you to identify and critique interaction design patterns.
- Explain how open source and ready-made components can support interaction design.
- Explain how tools can support interaction design activities.

13.1 Introduction

As our interviewee at the end of Chapter 1, Harry Brignull, remarked, the field of interaction design changes rapidly. He says, "A good interaction designer has skills that work like expanding foam." In other words, the practice of interaction design is quite messy, and keeping up with new techniques and developments is a constant goal. When placed within the wider world of commerce and business, interaction designers face a range of pressures, including restricted time and limited resources, and they need to work with people in a wide range of other roles, as well as stakeholders. In addition, the principles, techniques, and approaches introduced in other chapters of this book need to be translated into practice, that is, into real situations with sets of real users, and this creates its own pressures.

Many different names may be given to a practitioner conducting interaction design activities, including interface designer, information architect, experience designer, usability engineer, and user experience designer. In this chapter, we refer to *user experience designer* and *user experience design* because these are most commonly found in industry to describe someone who performs the range of interaction design tasks such as interface design, user evaluations, information architecture design, visual design, persona development, and prototyping.

Other chapters of this book may have given the impression that designers create their designs from scratch, with little or no help from anyone except users and immediate colleagues, but in practice, user experience (UX) designers draw on a range of support. Four main areas of support that impact the job of UX designers are described in this chapter.

- Working with software and product development teams operating an agile model of development (introduced in Chapter 2, "The Process of Interaction Design") has led to technique and process adaptation, resulting in agileUX approaches.
- Reusing existing designs and concepts is valuable and time-saving. Interaction design and UX design patterns provide the blueprint for successful designs, utilizing previous work and saving time by avoiding "reinventing the wheel."
- Reusable components—from screen widgets and source code libraries to full systems, and from motors and sensors to complete robots—can be modified and integrated to generate prototypes or full products. Design patterns embody an interaction idea, but reusable components provide implemented chunks of code or widgets.
- There is a wide range of tools and development environments available to support designers in developing visual designs, wireframes, interface sketches, interactive prototypes, and more.

This chapter introduces each of these four areas.

In this video, Kara Pernice suggests three challenges for UX in practice. It is available at **https://www.youtube.com/watch?v=qV5lLjmL278**.

Here is a concrete view of what a UX designer does in practice: **https://www.interaction-design.org/literature/article/7-ux-deliverables-what-will-i-be-making-as-a-ux-designer**

BOX 13.1

Technical Debt in UX

Technical debt is a term commonly used in software development, coined originally by Ward Cunningham in 1992, which refers to making technical compromises that are expedient in the short term but that create a technical context that increases complexity and cost in the long term. As with financial debt, technical debt is acceptable as a short-term approach to overcoming an immediate shortfall, provided that the debt will be repaid quickly. Leaving a debt for longer results in significant extra costs. Technical debt can be incurred unintentionally, but pressures associated with time and complexity also lead to design trade-offs that may prove to be expensive in the longer term.

UX debt is created much like technical debt in the sense that trade-offs are made for the needs of the project.

To address technical debt, a discipline of *refactoring* is needed, that is, correcting any pragmatic trade-offs quickly after the immediate pressure has receded. Significant difficulties arise if these trade-offs are not identified, understood, and corrected in a timely manner. Two interrelated situations can lead to significant user experience debt that is then extremely costly to correct.

- *If an organization did not, in the past, understand the value of good user experience design and products or software systems with poor user experiences persist*. This can be particularly prevalent for internal systems and products, where the drive for a good user experience is less acute than for externally marketed products that face more competition from other providers.
- *If an organization has a large portfolio of products, each of which was developed independently*. This can be the result of acquisitions and mergers of companies, each with their own UX brand, leading to a proliferation of designs.

In severe cases, UX debt can lead to the revamping of infrastructure and complete renewal of products. ■

For an interesting take on UX debt, see this article: **https://www.nngroup.com/articles/ux-debt**.

13.2 AgileUX

Since the rise of agile software development during the 2000s, UX designers have been concerned about the impact that it will have on their own work (Sharp et al., 2006), and the debate is ongoing (McInerney, 2017). *AgileUX* is the collective label given to efforts that aim to resolve these concerns by integrating techniques and processes from interaction design and those from agile methods. While agile software development and UX design have some characteristics in common such as iteration, a focus on measurable completion criteria, and user involvement, agileUX requires a reorganization and some rethinking of UX design activities and products. A recent reflection on the evolution of agileUX concluded that integrating agile and UX requires mutual team understanding across three dimensions, and those dimensions are variably understood (Da Silva et al., 2018): the "process and practice" dimension is understood; the "people and social" dimension is nearly understood; but the "technology and artifact" dimension—that is, use of technology to coordinate teams' activities and artifacts to mediate teams' communication—has yet to be properly understood. A key aspect is for agile development teams to understand that user experience design is not a role but is a discipline and mind-set. This account makes it clear that using agileUX in practice is far from straightforward. The key is to find a suitable balance that preserves both the research and reflection

needed for good UX design, as well as rapid iterations that incorporate user feedback and allow technical alternatives to be tested.

In a plan-driven (waterfall) software development process, requirements are specified as completely as possible before any implementation begins. In an agile software development process, requirements are specified only in enough detail for implementation to begin. Requirements are then elaborated as implementation proceeds, according to a set of priorities that change on a regular basis in response to changing business needs.

To integrate UX design into an agile workflow, it also needs to progress in a similar fashion. Reprioritization may happen as frequently as every two weeks, at the beginning of each iterative cycle. The shift from developing complete requirements up front to "just-in-time" or just enough requirements aims to reduce wasted effort, but it means that UX designers (along with their software engineer colleagues) have had to rethink their approach. All of the techniques and principles that UX designers use are just as relevant, but how much of each activity needs to be completed at what point in the iterative cycle and how the results of those activities feed into implementation need to be adjusted in an agile development context. This can be unsettling for designers, as the design artifacts are their main deliverable and hence may be viewed as finished, whereas for agile software engineers, they are consumables and will need to change as implementation progresses and requirements become elaborated.

Consider the group travel organizer example introduced in Chapter 11, and assume that it is being developed using agileUX. Four epics (large user stories) for the product are identified in Chapter 11, as follows:

1. As a <group traveler>, I want <to choose from a range of potential vacations that suit the group's preferences> so that <the whole group can have a good time>.
2. As a <group traveler>, I want <to know the visa restrictions for everyone in the group> so that <visas can be arranged for everyone in the group in plenty of time>.
3. As a <group traveler>, I want <to know the vaccinations required to visit the chosen destination> so that <vaccinations can be arranged for everyone in the group in plenty of time>.
4. As a <travel agent>, I want <up-to-date information displayed> so that <my clients receive accurate information>.

At the beginning of the project, these epics will be prioritized, and the central goal of the product (to identify potential vacations) will be the top priority. This will then initially be the focus of development activities. To allow users to choose a vacation, epic 4, supporting the travel agent to update travel details, will also need to be implemented (otherwise travel details will be out of date), so this is also likely to be prioritized. Establishing the detailed requirements and the design of the other two areas will be postponed until after a product that allows users to choose a vacation has been delivered. Indeed, once this product is delivered, the customer may decide that offering help for vaccinations and visas does not result in sufficient business value for it to be included at all. In this case, referring users to other, more authoritative sources of information may be preferable.

Conducting UX activities within an agile framework requires a flexible point of view that focuses more on the end product as the deliverable than on the design artifacts as deliverables. It also requires cross-functional teams where specialists from a range of disciplines, including UX design and engineering, work closely together to evolve an understanding of both the users and their context, as well as the technical capabilities and practicalities of the technology. In particular, agileUX requires attention to three practices, each of which is elaborated in the following sections.

- What user research to conduct, how much, and when
- How to align UX design and agile working practices
- What documentation to produce, how much, and when

"Enough storyboarding. Let's shoot something."

Source: Leo Cullum / Cartoon Stock

13.2.1 User Research

The term *user research* refers to the data collection and analysis activities necessary to characterize the users, their tasks, and the context of use before product development begins. Field studies and ethnography are often used in these investigations, but agile development works on short "timeboxes" of activity (up to four weeks in length, but often only two weeks in length) and hence does not support long periods of user research. (Different names are given by different agile methods to the iteration, or timeframe, the most common being *sprint*, *timebox*, and *cycle*.) Even a month to develop a set of personas or to conduct a detailed investigation into online purchasing habits (for example) is too long for some agile development cycles. User-focused activities evaluating elements of the design, or interviews to clarify requirements or task context, can be done alongside technical development (see the parallel tracks approach discussed in a moment), but planning to conduct extensive user research once iterative development starts will result in shallow user research, which is impossible to react to, as there just isn't enough time.

One way to address this is for user research to be conducted before the project begins, or indeed before it is announced, as suggested by Don Norman (2006), who argues that it is better to be on the team that decides which project will be done at all, hence avoiding the constraints caused by limited timeboxes. This period is often called *iteration zero* (or Cycle 0, as you'll see later in Figure 13.2), and it is used to achieve a range of up-front activities including software architecture design as well as user research.

Another approach to conducting user research for each project is to have an ongoing program of user research that revises and refines a company's knowledge of their users over a longer time span. For example, Microsoft actively recruits users of their software to sign up and take part in user research that is used to inform future developments. In this case, the specific data gathering and analysis needed for one project would be conducted during iteration zero, but done in the context of a wider understanding of users and their goals.

ACTIVITY 13.1

Consider the "one-stop car shop" introduced in Activity 11.4. What kind of user research would be helpful to conduct before iterative development begins? Of these areas, which would be useful to conduct in an ongoing program?

Comment

Characterizing car drivers and the hybrid driving experience would be appropriate user research before iterative development begins. Although many people drive, the driving experience is different depending on the car itself and according to the individual's capabilities and experiences. Collecting and analyzing suitable data to inform the product's development is likely to take longer than the timebox constraints would allow. Such user research could develop a set of personas (maybe one set for each class of car) and a deeper understanding of the hybrid driving experience.

Car performance and handling is constantly evolving, however, and so an understanding of the driving experience would benefit from ongoing user research. ■

Lean UX (see Box 13.2) takes a different approach to user research by focusing on getting products into the market and capturing user feedback on products that are in the marketplace. It specifically focuses on designing and developing innovative products.

BOX 13.2

Lean UX (Adapted from Gothelf and Seiden (2016))

Lean UX is designed to create and deploy innovative products quickly. It is linked to agileUX because agile software development is one of its underlying philosophies and it champions the importance of providing a good user experience. Lean UX builds upon UX design, design thinking, agile software development, and the Lean Startup ideas (Ries, 2011). All four perspectives emphasize iterative development, collaboration between all stakeholders, and cross-functional teams.

Lean UX is based on tight iterations of build-measure-learn, a concept central to the lean startup idea, which in turn was inspired by the lean manufacturing process from Japan. The lean UX process is illustrated in Figure 13.1. It emphasizes waste reduction, the importance of experimentation to learn, and the need to articulate outcomes, assumptions, and hypotheses about a planned product. Moving the focus from outputs (for example, a new smartphone app) to outcomes (for example, more commercial activity through mobile channels) clarifies the aims of the project and provides metrics for defining success. The importance of identifying assumptions was discussed in Chapter 3, "Conceptualizing Interaction." An example assumption might be that young people would rather use a smartphone app to access local event information than any other media. Assumptions can be expressed as hypotheses that can be put to the test more easily through research or by building a *minimum viable product* (MVP) that can be released to the user group.

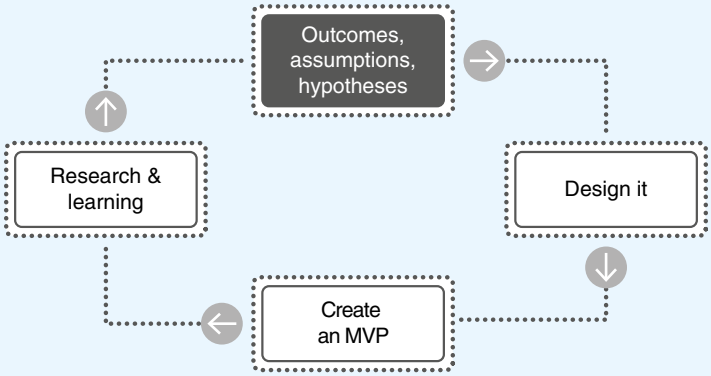


Figure 13.1 The Lean UX process
Source: Gothelf and Seiden (2016). Used courtesy of O'Reilly Media

Testing hypotheses, and hence assumptions, is done through experimentation, but before undertaking an experiment, the evidence required to confirm or refute each assumption needs to be characterized. An MVP is the smallest product that can be built that allows assumptions to be tested by giving it to a user group and seeing what happens. Experimentation and the evidence collected are therefore based on actual use of the product, and this allows the team to learn something.

As an example, Gothelf and Seiden (2016, pp. 76-77) describes an example of a company that wanted to launch a monthly newsletter. Their assumption was that a monthly newsletter would be attractive to their customers. To test this assumption, they spent half a day designing and coding a sign-up form on their website and collected evidence in the form of the number of sign-ups received. This form was an MVP that allowed them to collect evidence to support or refute their assumption, that is, that a monthly newsletter would be attractive to their customers. Having collected enough data, they planned to continue their experiments with further MVPs that experimented with formats and content for the newsletter. ■

In this video, Laura Klein explains Lean UX, at **http://youtu.be/7NkMm5WefBA**.

13.2.2 Aligning Work Practices

If requirements are specified before implementation begins, there is a tendency for designers to develop complete UX designs at the beginning of a project to ensure a coherent design throughout. In agile terms, this is referred to as *big design up front* (BDUF), and this is an anathema to agile working. Agile development emphasizes regular delivery of working software through evolutionary development and the elaboration of requirements as implementation proceeds. In this context, BDUF leads to practical problems since the reprioritization

of requirements means that interaction elements (features, workflows, and options) may no longer be needed or may require redesigning. To avoid unnecessary work on detailed design, UX design activities need to be conducted alongside and around agile iterations. The challenge is how to organize this so that a good user experience is achieved and the product vision is maintained (Kollman et al., 2009).

In response to this challenge, Miller (2006) and Sy (2007) proposed that UX design work is done one iteration ahead of development work in parallel tracks (see Figure 13.2). The parallel tracks approach to integrating UX design and agile processes originated at Alias—now part of Autodesk. Note that in this diagram, iteration is referred to as *Cycle*. The principle of parallel tracks development is quite simple: that design activity and user data collection for Cycle *n*+1 is performed during Cycle *n*. This enables the design work to be completed just ahead of development work, yet to be tightly coupled to it as the product evolves. Completing it much sooner than this can result in wasted effort, as the product and understanding about its use evolves.

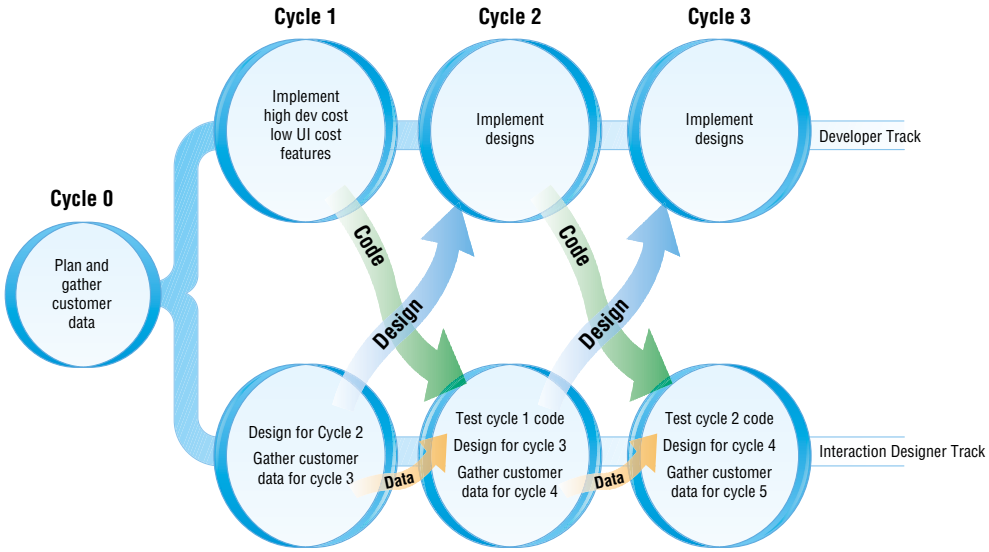


Figure 13.2 Cycle 0 and its relationship to later cycles
Source: Sy (2017)

Cycle 0 and Cycle 1 are different from subsequent cycles because, before evolutionary development can begin, the product vision needs to be created. This is handled in different ways in different agile methods, but all agree that there needs to be some kind of work up front to understand the product, its scope, and its overall design (both technical and UX). Some general data about customers and their behavior may have been collected before Cycle 0, but the vision and overall design is completed for the current project by the end of Cycle 0. The work required will depend on the nature of the product: whether it is a new version of an existing product, a new product, or a completely new experience. Cycle 0 can also be longer than other cycles to accommodate differing needs, but producing pixel-perfect designs of the product before evolutionary development starts is not the aim for Cycle 0.

One of the originators of the parallel tracks development idea, Desiree Sy (2007), explained this in the context of two different products. The first product is SketchBook Pro v2.0, a sophisticated sketching, annotating, and presentation tool to support digital artists. The second is Autodesk's Showcase which, though no longer available, was a real-time automotive 3D visualization product. For SketchBook Pro v2.0, the team conducted a survey of users who had downloaded v1.0 (a free trial version) but had not purchased v2.0. The results of the survey helped the team to refine 100 features into five major work streams, and this information informed development and prioritization throughout the development process. For Showcase, during Cycle 0, the team interviewed potential purchasers who performed work that the tool was going to be designed to support. This data formed the foundation for the design principles of the product as well as prioritization and design decisions as development progressed.

Cycle 1 usually involves technical setup activities in the developer track, which allows the UX designers to get started on the design and user activities for Cycle 2. For subsequent cycles, the team gets into a rhythm of design and user activities in Cycle n–1 and corresponding technical activity in Cycle n.

For example, imagine that development of a smartphone app to support attendance at a music festival is in Cycle n, and that Cycle n is scheduled to work on capturing reviews of the acts performing at the festival. During Cycle n–1, UX designers will have produced initial designs for capturing reviews of the acts by designing detailed icons, buttons, or other graphics, and prototyping different interaction types. During Cycle n, they will answer specific queries about these concrete designs, and they will revise them if necessary based on implementation feedback. Cycle n design work will be to develop concrete designs for the next cycle, which might be focusing on identifying and displaying reviews on demand. Also during Cycle n, UX designers will evaluate the implementation coming out of Cycle n–1. So, in any one cycle, UX designers are handling three different types of activity: evaluating implementations from the previous cycle, producing concrete designs for the next cycle, and answering queries on the designs being implemented in the current cycle.

The team at Alias found that the UX designers worked closely with the developers during design and implementation to make sure that they designed something that could be implemented and also what was indeed implemented was what had actually been designed. The interaction designers felt that there were three big advantages to this process. First, no design time was wasted on features that would not be implemented. Second, usability testing (for one set of features) and contextual inquiry (for the next set) could be done on the same customer visit, thus saving time. Third, the interaction designers received timely feedback from all sides—both users and developers. More importantly, they had time to react to that feedback because of the agile way of working. For example, the schedule could be changed if something was going to take longer to develop than first thought, or a feature could be dropped if it became apparent from the users that something else had higher priority. In summary, "Agile user-centered design resulted in better-designed software than waterfall user-centered design" (Sy, 2007, p. 130).

These advantages have been realized by others too, and this parallel tracks way of working has become a popular way to implement agileUX. Sometimes, the UX designers work two iterations ahead, depending on the work to be done, the length of the iteration, and external factors such as time required to obtain appropriate user input. Working in this way does not diminish the need for UX designers and other team members to collaborate closely together, and although the tracks are parallel, they should not be seen as separate processes. This does, however, raise a dilemma, as discussed in the Dilemma box.

DILEMMA

To Co-locate or Not to Co-locate, That Is the Question

UX designers in most large organizations are not numerous enough to have one UX designer for every team, so where should the UX designer be located? Agile development emphasizes regular communication and the importance of being informed about the project as it evolves. Hence, it would be good for the UX designer to be located with the rest of the team. But which team? Maybe a different agile team every day? Or each team for one iteration? Some organizations, however, believe that it is better for UX designers to sit together in order to provide discipline coherence: "UX designers work best when they are separated from the issues of software construction because these issues hamper creativity" (Ferreira et al., 2011). Indeed, this view is shared by some UX designers. If you, as a UX designer, were part of several agile teams, needing to engage with each of them, where would you prefer to be located? What might be the advantages and disadvantages of each, or maybe using a social awareness tool such those introduced in Chapter 5, "Social Interaction," would be more appropriate? ■

This video describes some case studies on the UX techniques used by Android within agile iterations: **http://youtu.be/6MOeVNbh9cY**.

ACTIVITY 13.2

Compare Lean UX, agileUX, and evolutionary prototyping (introduced in Chapter 12, "Design, Prototyping, and Construction"). In what ways are they similar and how do they differ?

Comment

Lean UX produces a MVP to test assumptions by releasing it to users as a finished product and collecting evidence of users' reactions. This evidence is then used to evolve subsequent (larger) products based on the results of this experimentation. In this sense, Lean UX is a form of evolutionary development, and it has similarities with evolutionary prototyping. However, not all the MVPs developed to test assumptions may be incorporated into the final product, just the results of the experiment.

AgileUX is an umbrella term for all efforts that focus on integrating UX design with agile development. Agile software development is an evolutionary approach to development, and hence agileUX is also evolutionary. Additionally, agileUX projects can employ prototyping to answer questions and test ideas as with any other approach, as described in Chapter 12. ■

13.2.3 Documentation

The most common way for UX designers to capture and communicate their design has been through documentation, for instance, user research results and resulting personas, detailed interface sketches, and wireframes. Because UX designers view the design as their main deliverable, a key indicator that their work is ready for sign-off is the availability of comprehensive documentation to show that their goals have been achieved. This may include other forms of design capture, such as prototypes and simulations, but documentation is still common. Agile development encourages only minimal documentation so that more time can be spent on design, thus producing value to the user via a working product.

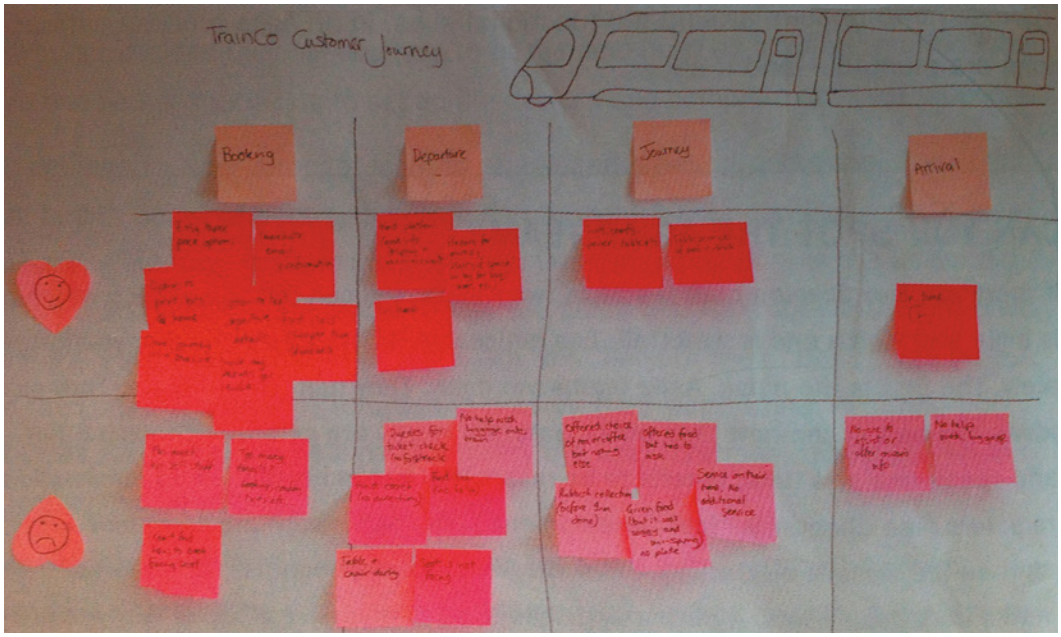
Minimal documentation does not mean "no documentation," and some documentation is desirable in most projects. However, a key principle in agileUX is that documentation should not replace communication and collaboration. To help identify the right level of documentation, Lindsay Ratcliffe and Marc McNeill (2012, p. 29) suggest asking a set of questions of any documentation process.

1. How much time do you spend on documentation? Try to decrease the amount of time spent on documentation and increase design time.
2. Who uses the documentation?
3. What is the minimum that customers need from the documentation?
4. How efficient is your sign-off process? How much time is spent waiting for documentation to be approved? What impact does this have on the project?
5. What evidence is there of document duplication? Are different parts of the business documenting the same things?
6. If documentation is only for the purpose of communication or development, how polished does it need to be?

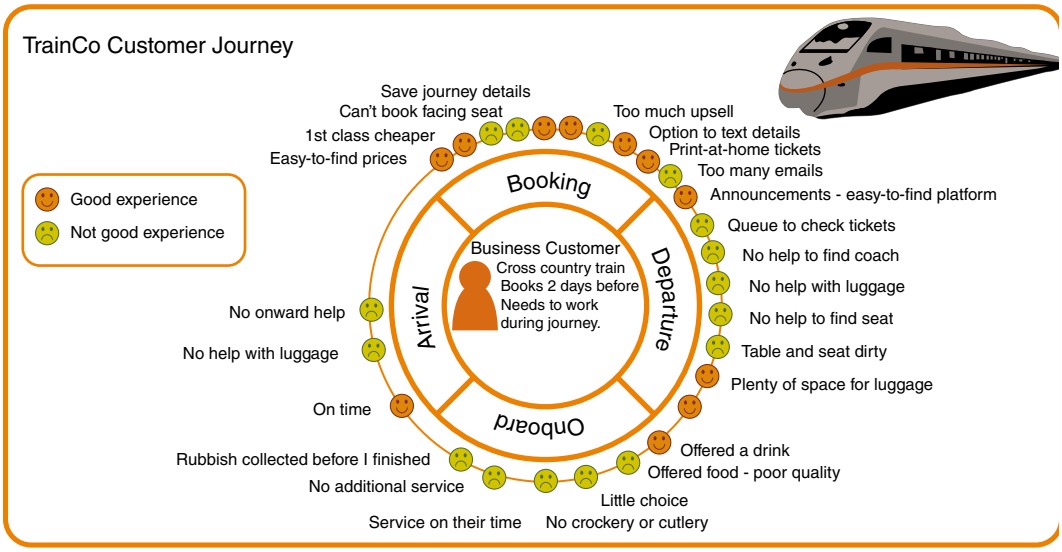
They also use the example in Figure 13.3 to illustrate these points. Both images capture a user journey, that is, one path a user might take through the product. The sketch in Figure 13.3(a) is constructed with sticky notes and string, and it was generated by all of the team members during a discussion. The sketch in Figure 13.3(b) took hours of designer time to draw and polish. It looks good, but that time could have been used to design the product rather than the user journey sketch.

The question of how much documentation is needed in an agile project is not limited to agileUX. Scott Ambler (Ambler, 2002) provides a detailed description of best practices for agile documentation. These support the production of "good enough" documentation in an efficient way, and they are intended to determine what documentation is needed. He proposes questions such as these:

- What is the purpose of the documentation?
- Who is the customer of the documentation?
- When should documents be updated?"



(a)



(b)

Figure 13.3 (a) A low-fidelity user journey, (b) a high-fidelity user journey
Source: Ratcliffe and McNeill (2012)

DILEMMA

Quick, Quick, Slow?

One of the challenges for UX practice is how best to integrate with software and product development conducted using an agile approach. Taking an agile approach is seen as beneficial for a range of reasons, including an emphasis on producing something of use, customer (and user) collaboration, rapid feedback, and minimal documentation—only areas of the product that are definitely going to be implemented are designed in detail. However, focusing on short timeboxes can lead to an impression that everything is being rushed. Creating an appropriate balance between short timeboxes and a reflective design process requires careful planning so that important aspects of UX design are not rushed.

Slow design is part of the *slow movement*, which advocates a cultural shift toward slowing down the pace of life (Grosse-Hering et al., 2013). The main intent of *slow design* is to focus on promoting well-being for individuals, society, and the natural environment by promoting the design of products that are long-lived and sustainable. Working more slowly does not, *per se*, address the impression of rushing, but slow design also emphasizes the importance of providing time to reflect and think, for the user to engage and create their own products, and for products and their use to evolve over time.

The agile movement is here to stay, but the importance of taking time to reflect and think, when necessary, and not rushing to make decisions remains. The dilemma here is finding the right balance between rapid feedback to identify solutions that work and providing the time to stop and reflect. ■

CASE STUDY 13.1

Integrating UX Design into a Dynamic Systems Development Method Project

Challenges, Working Practices, and Lessons Learned

This case study presents a portion of one organization's journey to integrate UX design into one agile software development approach: the Dynamic Systems Development Method (DSDM) framework (see https://www.agilebusiness.org/what-is-dsdm for more details). It describes the difficulties they faced, the working practices adopted, and the lessons learned from their experiences of integrating UX designers into their DSDM agile process.

LShift is a high-tech software development company that works across a broad range of industries, languages, and platforms. The company faced four main challenges while integrating UX design into the DSDM framework.

- *Communication between developers and UX designers*: What is the relevant information that needs to be communicated, how best to communicate it, how to keep communication channels open, and how to keep the emerging design implementation visible for feedback. Difficulties in these areas can cause frustration, problems with the technical feasibility of design solutions, and mistaken expectations by the client.

- *Level of precision in up-front design*: Developers suggested five main reasons why "less is more" when it comes to design documentation ready for the start of developer involvement.
  - Prioritization and de-scoping can lead to a waste of pixel-perfect designs.
  - Some design issues will be found only once you start implementing.
  - Pixel-perfect designs may increase resistance to making design changes.
  - It is better to focus on functionality first and design as you go along.
  - Quality of designs can benefit from early input by developers.
- *Design documentation*: The amount and detail of documentation needs to be discussed early on so that it meets both developers' and designers' requirements.
- *User testing*: User testing can be a challenge in a product development setting if the business does not yet have any customers. This can be addressed at least partially using personas and user representatives.

This case study describes the background to these challenges, provides more detail about these challenges, and introduces some practices that the company used to address them. The case study is available in full at http://tinyurl.com/neehnbk. ■

13.3 Design Patterns

Design patterns capture design experience, but they have a different structure and a different philosophy from other forms of guidance or specific methods. One of the intentions of the patterns community is to create a vocabulary based on the names of the patterns, which designers can use to communicate with one another and with users. Another is to produce literature in the field that documents experience in a compelling form.

The idea of patterns was first proposed by the architect Christopher Alexander, who described patterns in architecture (Alexander, 1979). His hope was to capture the "quality without a name" that is recognizable in something when you know it is good.

But what is a design pattern? One simple definition is that it is a solution to a problem in a context; that is, a pattern describes a problem, a solution, and where this solution has been found to work. Users of the pattern can therefore not only see the problem and solution but can also understand the circumstances under which the idea has worked before and access a rationale for why it worked. A key characteristic of design patterns is that they are generative; that is, they can be instantiated or implemented in many different ways. The application of patterns to interaction design has grown steadily since the late 1990s (for instance, Borchers, 2001; Tidwell, 2006; Crumlish and Malone, 2009) and have continued to be actively developed (for example, Josh et al., 2017).

Patterns on their own are interesting, but they are not as powerful as a pattern language. A *pattern language* is a network of patterns that reference one another and work together to create a complete structure. Pattern languages are not common in interaction design, but there are several pattern collections, that is, sets of patterns that are independent of each other. Patterns are attractive to designers because they are tried and tested solutions to common problems. It is common (although not obligatory) for pattern collections to be associated with software components that can be used with little modification, and as they are common solutions, many users are already familiar with them, which is a great advantage for a new app or product on the market. See Box 13.3 for an example pattern: Swiss Army Knife Navigation.

BOX 13.3

Swiss Army Knife Navigation: An Example Design Pattern for Mobile Devices (Nudelman (2013). Used courtesy of John Wiley & Sons, Inc.)

The principle behind the Swiss Army Knife Navigation design pattern is to maximize productive use of the screen space and keep users engaged in the content of what they are doing. For example, in a game design, the user does not want to be side-tracked by navigation bars and menu pop-ups. Having a mechanism that allows the controls to fade in and out of view is a much more engaging design.

This design pattern is commonly instantiated as "off-canvas" or "side drawer" navigation (Neil, 2014), where the control bar slides in, overlaying the main screen contents. It is useful because it is a "transient" navigation bar and takes up screen space only temporarily; that is, it can be swiped in over the top of the main app screen and then swiped back once the user has finished the action. It is good from an interaction design point of view because it supports the use of both text and icons to represent actions. It is also good from the point of view of screen layout because it takes up space only when the menu is needed. It can also be used for interactions other than navigation (Peatt, 2014). This is an example of a common design pattern that is also evolving in a range of different directions. Exactly how this navigation bar is implemented varies with different platforms.

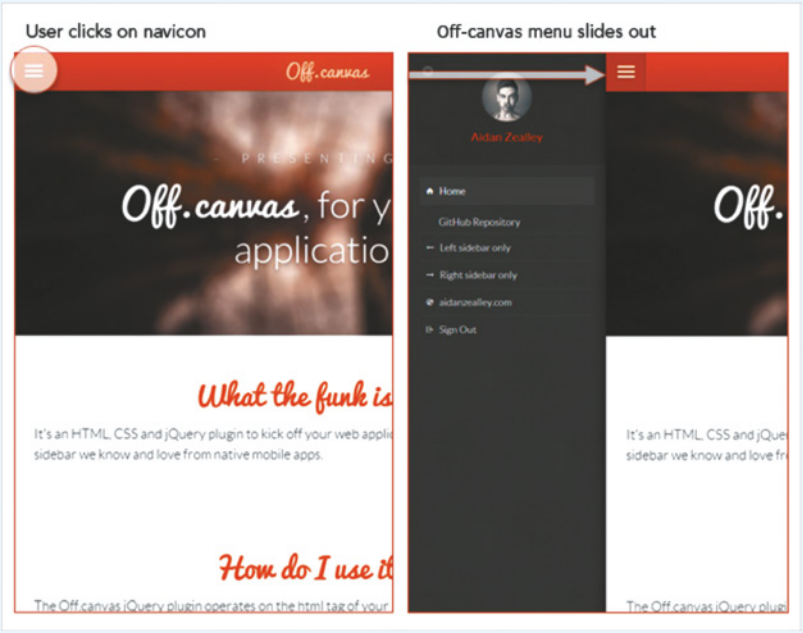


Figure 13.4 Example of Swiss Army Knife Navigation pattern, instantiated as off-canvas navigation
Source: Used courtesy of Aidan Zealley

In figure 13.4, the left image, the menu is represented as a list of lines at the top left of the screen, while in the right image, the menu items have pushed the user view to the right. ■

Examples of interaction design guidelines and pattern libraries plus downloadable collections of screen elements are available at:
Windows: **https://developer.microsoft.com/en-us/windows/apps/design**
Mac: **https://developer.apple.com/design/human-interface-guidelines/**
General UI Design Patterns: **https://www.interaction-design.org/literature/article/10-great-sites-for-ui-design-patterns**
Google Material Design: **https://design.google/**

Pattern collections, libraries, and galleries relevant to interaction design are commonly used in practice (for instance, Nudelman, 2013) and are often accompanied by code snippets available through open source repositories such as GitHub (https://github.com/) or through platform websites such as https://developer.apple.com/library/iOS/documentation/userexperience/conceptual/mobilehig/ for iOS on an iPhone.

Patterns are a "work in progress," because they continue to evolve as more people use them, experience increases, and users' preferences change. Patterns can continue to evolve for some time, but they can also be deprecated, that is, become outdated and no longer considered good interaction design. Reusing ideas that have proved to be successful in the past is a good strategy in general, particularly as a starting point, but it should not be used blindly. In custom applications, the design team may also create their own libraries. As with many areas of design, there is disagreement about which patterns are current and which are outdated.

For a humorous discussion of the hamburger icon, see:
https://icons8.com/articles/most-hated-ui-ux-design-pattern/
For a discussion of the tab bar, see:
https://uxplanet.org/tab-bars-are-the-new-hamburger-menus-9138891e98f4

ACTIVITY 13.3

One design pattern for mobile devices that is deprecated by some and not others is the Carousel navigation pattern, in which the user is presented with several images (of products, for example) horizontally across the screen, or one at a time in the same screen location. Swiping (or clicking) left or right displays other images, just like a carousel.

This design pattern has provoked different reactions by different designers. Search for information on this design pattern using your favorite browser and read at least two articles or blog posts about it: one arguing that it should be deprecated and one that explains how it can be used successfully. Decide for yourself whether the Carousel pattern should be labeled outdated or kept alive.

(a)

(b)

Figure 13.5 Two example carousel navigation styles (a) shows pictures of a house for sale. Note the arrows to the left and right of the row of photos at the bottom. (b) shows a weather application for a mobile phone that can be swiped left and right for other locations. Note the line of dashes in the bottom middle of the screen that indicate there are other screens.

Source: Helen Sharp

Comment

The Nielsen Norman Group has two articles about the carousel on its website (see the URLs at the end of this paragraph).

One presents evidence from a usability trial with one user that shows carousels can fail, and the other presents a balanced view of how to design a good carousel. This second article focuses on the version of the carousel where several images are displayed at the same location on the screen, one at a time. They identify the greatest advantages as being the good use of screen space (because several elements occupy the same space) and that having information at the top of the screen means that visitors are more likely to see it. Disadvantages include that users often navigate past the carousel and that even if users do see the image, it is usually only the first one. The article does suggest using an alternative design, and it goes on to provide some useful examples and guidelines for good carousels.

There is a thread of posts and articles arguing that the carousel should not be used. These also point to evidence that users rarely use the carousel, but if they do they focus only on the first image. Nevertheless, there seems to be no solid set of data to support or refute the usability of the carousel in all of its various forms.

On balance, it seems that some forms of carousel meet the product's goals more readily than others, for example, because only the first image in a series is viewed by most users. Assuming appropriate design and, maybe more importantly, an evaluation with potential users and your content, it seems plausible that the carousel navigation pattern is not yet ready to be deprecated. ■

www.nngroup.com/articles/designing-effective-carousels/
www.nngroup.com/articles/auto-forwarding/

Design patterns are a distillation of previous common practice, but one of the problems with common practice is that it is not necessarily good practice. Design approaches that represent poor practice are referred to as *anti-patterns*. The quality of interaction design and user experience in general has improved immensely since the first edition of this book in 2002, so why are anti-patterns still a problem? Basically, the technology is changing and design solutions that work on one platform don't necessarily work on another. A common source of anti-patterns for mobile devices is where websites or other software have been migrated from a large screen, such as a laptop, to a smartphone. One example of this is the untappable phone number that displays on a smartphone pop-up (see Figure 13.6).

Another kind of pattern that was introduced in Chapter 1 (see Figure 1.10) is the *dark pattern*. Dark patterns are not necessarily poor design, but they have been designed carefully to trick people, championing stakeholder value over user value, for instance. Some apparent dark patterns are just mistakes, in which case they will be corrected relatively quickly once identified. However, when a UX designer's knowledge of human behavior is deliberately used to implement deceptive functionality that is not in the user's best interests, that is a dark pattern. Colin Gray et al. (2018) collated and analyzed a set of 118 dark pattern examples identified by practitioners and identified five strategies: nagging, obstruction, sneaking, interface interference, and forced action.

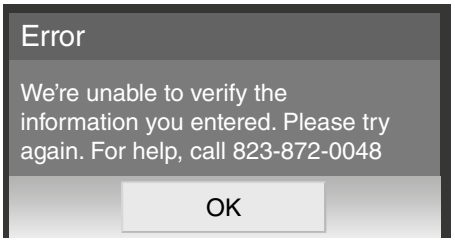


Figure 13.6 An untappable phone number for help when smartphone installation goes wrong

13.4 Open Source Resources

Open source software refers to source code for components, frameworks, or whole systems that is available for reuse or modification free of charge. Open source development is a community-driven endeavor in which individuals produce, maintain, and enhance code, which is then given back to the community through an open source repository for further development and use. The community of open source committers (that is, those who write and maintain this software) are mostly software developers who give their time for free. The components are available for (re)use under software licenses that allow anyone to use and modify the software for their own requirements without the standard copyright restrictions.

Many large pieces of software underlying our global digital infrastructure are powered by open source projects. For example, the operating system Linux, the development environment Eclipse, and the NetBeans development tools are all examples of open source software.

Perhaps more interesting for interaction designers is that there is a growing proportion of open source software available for designing good user experiences. The design pattern implementation libraries introduced in section 13.3 are but one example of how open source software is affecting user experience design. Another example is the Bootstrap framework for front-end web development, released as open source in August 2011 and actively updated on a regular basis; see Figure 13.7 for an example of its use. This framework contains reusable code snippets, a screen layout grid that supports multiple screen sizes, and pattern libraries that include predefined sets of navigational patterns, typefaces, buttons, tabs, and so on. The framework and documentation are available through the GitHub open source repository (https://github.com/twbs/bootstrap#community).

Open source resources require a suitable repository, that is, somewhere for the source code to be stored and made accessible to others. More than this, the repository needs to serve a huge number of users (GitHub was reported to have 31 million users in 2018) who will want to build, review, modify, and extend software products. Managing this level of activity also requires version control, such as a mechanism that retains and can reinstate previous versions of the software. For example, GitHub is based on the version control system called Git. Communities form around these repositories, and submitting code to a repository requires an account. For example, each developer on GitHub can set up a profile that will keep track of their activity for others to see and comment upon.

Figure 13.7 An example website built using the Bootstrap framework
Source: Didier Garcia/Larson Associates.

Most repositories support both public and private spaces. Submitting code to a public space means that anyone in the community can see and download the code, but in a private space the source will be "closed." One of the advantages of putting code on an open source repository is that many eyes can see, use, and modify your work—spotting security vulnerabilities or inefficient coding practices as well as contributing to, extending, or improving its functionality. Other popular open source repositories are BitBucket, Team Foundation Server, and GitLab.

Some advantages of using GitHub over other source repositories are discussed here:
https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946

The GitHub repository itself may look a little daunting for those who first come across it, but there is a community of developers behind it who are happy to help and support newcomers.

An introduction to using GitHub is available here:
https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

13.5 Tools for Interaction Design

Many types of digital tools are used in practice by UX designers, and the tooling landscape changes all the time (Putnam et al., 2018). These tools support creative thinking, design sketching, simulation, video capture, automatic consistency checking, brainstorming, library search, and mind mapping. In fact, any aspect of the design process will have at least one associated support tool. For example, Microsoft Visio and OmniGraffle support the creation of a wide range of drawings and screen layouts, while FreeMind is an open source, mind-mapping tool. In and of themselves, these tools provide significant support for UX design, but they can also work together to speed up the process of creating prototypes of various levels of fidelity.

Elsewhere in this book, we have emphasized the value of low-fidelity prototyping and its use in getting user feedback. As with any prototype, however, paper-based prototypes have their limitations, and they do not support user-driven interaction (see, for example, Lim et al., 2006). In recognition of this, developing interactive, low-fidelity prototypes has been investigated through research for many years (see Lin et al., 2000, or Segura et al., 2012). In recent years, tools to support the creation of interactive prototypes from static graphical elements have become available commercially. For example, overflow.io supports the production of playable user flow diagrams.

Commercial packages that support the quick and easy development of interactive wireframes, or mock-ups, are widely used in practice for demonstration and evaluation. Some commonly used tools are Balsamiq® (https://balsamiq.com/), Axure RP (https://www.axure.com/), and Sketch (https://sketchapp.com/). Activity 13.4 invites you to try one or more of the tools available to create a simple prototype.

Tools available for UX designers, many of which have free trial versions and tutorials, are available at the following links:
https://support.balsamiq.com/tutorials/
www.axure.com/learn/core/getting-started
https://www.digitalartsonline.co.uk/features/interactive-design/16-best-ux-tools-for-designers/
https://blog.prototypr.io/meet-overflow-9b2d926b6093

Having created an interactive wireframe using one of these tools, it is then possible to generate a higher-fidelity prototype by implementing the next prototype using a ready-made pattern library or framework, like those introduced in section 13.3 and section 13.4, to provide a coherent look and feel. This means going from a low-fidelity mockup to a working, styled prototype in one step. Other open source resources can also be used to provide a wider choice of interface elements or design components with which to create the product.

Paper-based prototypes are also not very good if technical performance issues such as component interfaces need to be prototyped—software-based prototypes are better. For example, Gosper et al. (2011) describes how, at SAP, employees often use a drawing or graphics package to mock up key use cases and their interfaces, interactions, and task flows and then output that to PowerPoint. This creates a set of slides that can be viewed to give an

overall sense of a user session. However, when they developed a business intelligence tool with key performance and "backend" implications, this form of prototyping was not sufficient for them to assess their product goals. Instead, the UX designer worked with a developer who prototyped some of the elements in Java.

ACTIVITY 13.4

Choose one of the commercially available tools that supports the generation of interactive wireframes or low-fidelity prototypes and generate a wireframe for a simple app, for instance, one that allows visitors to a local music festival to review the acts. Explore the different features offered by the tool, and note those that were particularly useful. Unless your employer or university has a license for these tools, you may not have access to all of their features.

Comment

We used the moqups web demo at https://app.moqups.com/edit/page/ad64222d5 to create the design in Figure 13.8. Two features of note for this tool are (1) that it appears to be similar to other generic graphics packages, and hence it was easy to get started and produce an initial wireframe, and (2) the ruler settings automatically allowed precise positioning of interface elements in relation to each other. ■

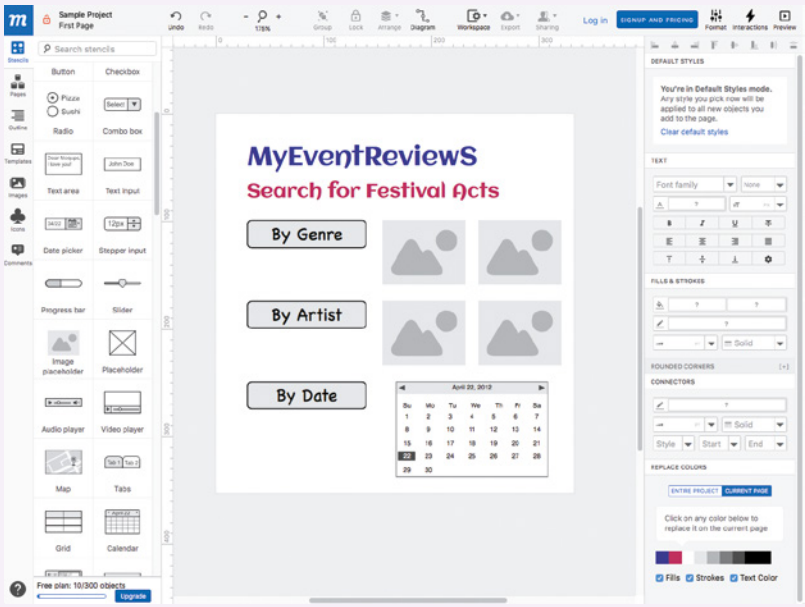


Figure 13.8 A screenshot of our interactive wireframe for the event review app, generated using moqups

Source: https://moqups.com

In-Depth Activity

This in-depth activity continues the work begun on the booking facility introduced at the end of Chapter 11.

1. Assume that you will produce the online booking facility using an agile approach.
   a. Suggest the type of user research to conduct before iteration cycles begin.
   b. Prioritize requirements for the product according to business value, in particular, which requirements are likely to provide the greatest business benefit, and sketch out the UX design work you would expect to undertake during the first four iteration cycles, that is, Cycle 0 and Cycles 1 to 3.
2. Using one of the mock-up tools introduced, generate a mock-up of the product's initial interface, as developed in the assignment for Chapter 12.
3. Using one of the patterns websites listed previously, identify suitable interaction patterns for elements of the product and develop a software-based prototype that incorporates all of the feedback and the results of the user experience mapping achieved at the end of Chapter 12. If you do not have experience in using any of these, create a few HTML web pages to represent the basic structure of the product.

Summary

This chapter explored some of the issues faced when interaction design is carried out in practice. The move toward agile development has led to a rethinking of how UX design techniques and methods may be integrated into and around agile's tight iterations. The existence of pattern and code libraries, together with open source components and automated tools, means that interactive prototypes with a coherent and consistent design can be generated quickly and easily, ready for demonstration and evaluation.

Key Points

- AgileUX refers to approaches that integrate UX design activities with an agile approach to product development.
- A move to agileUX requires a change in mind-set because of repeated reprioritization of requirements and short timeboxed implementation, which seeks to avoid wasted effort.
- AgileUX requires a rethinking of UX design activities: when to perform them, how much detail to undertake and when, and how to feed back results into implementation cycles.
- Design patterns present a solution to a problem in a context, and there are many UX design pattern libraries available.
- Dark patterns are designed to trick users into making choices that have unexpected consequences, for instance, by automatically signing them up for marketing newsletters.
- Open source resources, such as those on GitHub, make the development of standard applications and libraries with consistent interfaces easier and quicker.
- A variety of digital tools to support interaction design in practice are available.

Further Reading

GOTHELF, J., and SEIDEN, J. (2016) *Lean UX: Designing Great Products with Agile Teams* (2nd ed.), O'Reilly. This book focuses on the lean UX approach to development (see Box 13.2), but it also includes a wide range of practical examples and experiences from readers of the first edition of the book as to how agile development and UX design can work well together.

KRUCHTEN, P., NORD, R.L. and OZKAYA, I. (2012) "Technical Debt: From Metaphor to Theory and Practice," IEEE Software, November/December (2nd ed.). This paper is the editors' introduction to a special issue on technical debt. This topic has been largely discussed and written about in the context of software development, with very little mention of interaction design or UX debt. However, these issues are relevant to interaction design practice today, and this paper provides an accessible starting point for anyone wanting to investigate this area further.

PUTNAM, C., BUNGUM, M., SPINNER, D., PARELKAR, A.N., VIPPARTI, S. and CASS, P. (2018), "How User Experience Is Practiced: Two Case Studies from the Field," Proceedings of CHI 2018. This short paper provides some useful insights into UX practice based on two case studies from consumer-facing companies in Illinois.

RAYMOND, E.S. (2001) *The Cathedral and the Bazaar*. O'Reilly. This seminal book is a set of essays introducing the open source movement.

SANDERS, L. and STAPPERS, P. J. (2014) "From Designing to Co-Designing to Collective Dreaming: Three Slices in Time," *interactions*, Nov–Dec, p. 25–33. This provides a fascinating account of the changes in design practice over the last 30 years, a reflection on what design practice is like in 2014, and then a projection into the future to see what design practice may be like 30 years from now. It considers the role of the customer and the designer and how the object being designed emerges from the design process.

SY, D. (2007) "Adapting Usability Investigations for Development," *Journal of Usability Studies* 2(3), May, 112–130. This short paper is a good introduction to some of the key issues faced when trying to perform UX design alongside an agile project. It describes the well-established dual-track process model for agileUX, shown in Figure 13.2.