CHAPTER

Agile Lifecycle Processes and the Funnel Model of Agile UX

4

Highlights

- Challenges in building systems.
- Change happens.
- Change creates a gap between the reality of requirements and the designer's understanding of the same.
- Success relies on being able to close that gap and respond to change.
- The old waterfall process.
- Embracing an agile lifecycle process.
- Scope and chunking are the most important characteristics.
- Agility is an outcome of chunking.
- Agile UX.
- Agile software engineering (SE).
- Looking at change in terms of divergence between understanding and reality.
- The funnel model of agile UX.
- Late funnel activities: Syncing with the SE sprints.
- Early funnel activities: Upfront analysis and design.

4.1 CHALLENGES IN BUILDING SYSTEMS

4.1.1 Change Happens During a Project

4.1.1.1 Evolution of project requirements and parameters

These days, much of the discussion about lifecycle processes is about how well they can respond to change. Why is the ability to respond to change so important? First of all, perhaps the biggest single lesson learned in the history of software

engineering (and UX design) is: *change is inevitable*. Over the course of a project, most project parameters change, including:

- Requirements (statement of system needs).
- Product concept, vision.
- System architecture.
- Design ideas.
- Available technology.

For simplicity, we will refer to this set of project parameters as the "requirements."

Here are some things we know:

- Technology is constantly changing; technology evolves to be better and faster and there are technological paradigm shifts (new products, new uses of old products).
- Because change occurs over time, the longer the time to deliver a release that can be evaluated, the more change can occur between original requirements and evaluation feedback.
- Because a larger scope means a larger time to delivery, to mitigate the impact of change, the scope needs to be small.

Stating this as a prerequisite for success:

Prerequisite 1: In a successful project, the scope needs to be small, so the time it takes to deliver a release is limited.

4.1.1.2 External changes

Things in the world at large also can change during a project, such as:

- Technology available at the time.
- Client's directions and focus (possibly due to shifting organizational goals or market factors).

Because these changes are external to the process, we have little control over them but we have to be responsive.

4.1.2 Two Views of These Changes

4.1.2.1 Reality

The "reality" view of change reflects true changes and reveals "real requirements" as they evolve within the project.

4.1.2.2 Designer's understanding of these changes

For simplicity in this context, we will use the term "designer" to denote the UX designer and the whole team, including relevant SE roles. The designer's view of change reflects the designer's understanding, awareness, or perception of the changes and is the result of evaluation feedback.

4.1.3 The Gap Between Views

There is a gap between the reality of requirements and the designer's perception of requirements. The designer's view of requirements usually lags reality in time and falls short of reality in content, but a successful project needs this gap to remain small.

Putting it as another prerequisite:

Prerequisite 2: In a successful project, the gap between reality (true requirements) and the designer's understanding of the same needs to remain small.

Prerequisite 2 interacts with Prerequisite 1 in that a larger scope of the delivery results in a greater time it takes to deliver. The greater the time it takes, the more the evolution of needs and the larger the gap.

Also, because initial requirements were based only on perceived needs, there is a gap to begin with, compounding the problem.

4.1.4 Responding to Change

The ability of the designer and the whole team to react to changes during the project depends on closing the gap between reality and the designer's view of reality—that is, how well the designer's understanding tracks the real requirements. This is all about the choice of lifecycle processes, an important subject of this book.

4.1.5 Closing the Gap

Closing the gap between real requirements and the designer's understanding of real requirements entails updating the designer's understanding as change occurs (i.e., as project parameters evolve). Updating the designer's awareness happens through learning afforded by feedback in the process (next section).

4.1.6 True Usage is the Only Ascertainer of Requirements

Perceived requirements are described by envisioning usage. In contrast, the ability to track changes and thereby know real requirements depends on learning from real usage feedback. Framing this as another prerequisite:

Prerequisite 3: Feedback from actual usage is the only way to know real requirements.

We can't know the real needs until after using the system. But of course, there is a dilemma: You can't build the system to try it out without knowing the requirements.

This dilemma must be addressed by a lifecycle process that can handle constant change. No, that isn't strong enough. Your lifecycle process doesn't just have to handle change; it must incorporate change as an operational feature of the process, and it must embrace constant change as a necessary mechanism for learning throughout the process (Dubberly & Evenson, 2011).

This means you need a lifecycle process that gives feedback on requirements before the whole system is built as well as feedback at every step from real usage, which is the key to learning about how conditions are changing.

4.1.7 Communicating Feedback About Requirements

Even when all the prerequisites so far are met, there can still be barriers to a successful project due to faulty feedback communication, gaps in the transfer of what is on the user's mind to what is on the designer's mind.

4.1.7.1 Communication problems on the user's side

Depending on users to tell you what is wrong with a system is not always reliable because users:

- Are not necessarily knowledgeable about technology and the overall system.
- Might have trouble formulating problems in their own minds (e.g., inability to abstract from problem instance details).
- Might lack the ability to articulate feedback about requirements.
- Might give feedback based on what they *think* they want.
- Have biases about certain aspects of the system.

And, even if users understand and articulate good requirements, it is possible that the designer misunderstands, a normal problem in written and verbal communication.

We can sum this up as another prerequisite:

Prerequisite 4: In a successful project, feedback about requirements must be communicated effectively.

4.2 THE OLD WATERFALL SE LIFECYCLE PROCESS

The original waterfall process was developed in the preconsumer era when most systems were large enterprise systems and users were trained to use a system for specific business purposes. There really were no system-development considerations of usability or UX.

4.2.1 The Waterfall Process was an Early SE Attempt to Get Organized

The waterfall process (originally called the waterfall model (Royce, 1970)) was among the earliest of formal software engineering lifecycle processes. The waterfall process is one of the simplest (in form, at least) ways to put lifecycle activities together to make an SE lifecycle process. The process was so named because it was described as an ordered, essentially linear sequence of phases (lifecycle activities), each of which flowed into the next like the set of cascading tiers of a waterfall (Fig. 4-1).

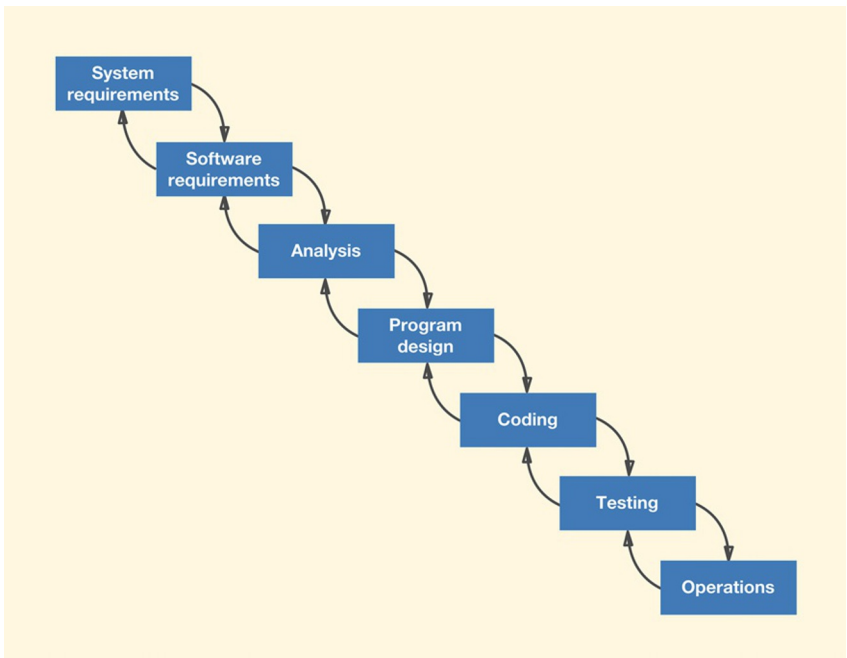

Fig. 4-1
Cascades from phase to phase in the waterfall model (adapted from Royce, 1970).

The objective of the waterfall process was to deliver the full system at once. Because this process was an attempt to overcome the previous "Wild West" approach to software, it was methodical and tended to be rigorous. It also helps to know that the systems being built at that time were mostly large enterprise or government systems. But, because it operated with large scope in the extreme, the waterfall process was also slow, cumbersome, unmanageable, and not very responsive to change.

For more about how the waterfall model got started see Section 6.5.

4.2.2 The Waterfall Process Did Have Some Feedback, But Not the Right Kind

4.2.2.1 Verification and validation of phase work products

The waterfall process did not entirely lack feedback before the whole system was delivered. To introduce a formal evaluation component, software people added evaluation in the form of verification and validation (V and V) (Lewis, 1992) at the end of each phase. Verification was to show that the software met specifications and validation was to ensure that it fulfilled its purpose. V and V helped a bit by periodically checking with users to see if the needs were still "valid" and on track.

4.2.2.2 But this wasn't enough

While V and V was a way to incorporate some client feedback after each phase, the ability to track the reality of changes was limited because feedback:

- Was at a large level of scope.
- Occurred only at the end of each large phase.
- Was based on analytic reality checking, not real empirical usage data (because there was no system yet to use until the entire system was delivered at the end of the lifecycle).

These bullet items imply that the waterfall process failed to meet Prerequisite 3 (need feedback based on real usage) because it was a whole scope process, and could not get to real usage until the very end.

To make it worse, the waterfall process also failed to meet Prerequisite 1 (need to limit the time to delivery) and Prerequisite 2 (the gap between reality and the designer's understanding must be small) because whole scope systems take a long time to build and, so, a significant amount of change happens before any feedback can occur, causing the gap from reality to grow very large.

4.2.2.3 Change discovered was too expensive to address

When problems were inevitably discovered at the end of each phase, it was expensive to go back and fix the whole system at that point. New requirements discovered during later stages kicked off expensive rework because they invalidated the previous phases' deliverables. Studies done at the time showed how a problem detected during implementation was many times more expensive to fix than if it was detected during requirements. Although the phase deliverables were not the real system, a heavy commitment of resources was made to correcting those because that was the only thing they could adjust. Even if everything was corrected with the new insights, the waterfall process still failed to meet Prerequisite 3 (feedback must be based on real usage).

4.2.2.4 Feedback was not communicated well with respect to user needs

And, with respect to the feedback they did get in the waterfall process, it often failed to meet Prerequisite 4 (communicating user feedback accurately) because at that time the emphasis was on the system and not the user. V and V addressed communication issues to an extent by giving users the opportunity to review the emerging system design, but SE people took a system perspective and focused on "under the hood" issues. And often these emerging system design artifacts were too technical and abstract and not relevant for users to understand.

4.2.2.5 The bottom line

The outcome was that, because the waterfall lifecycle process represented the ultimate in batching with very large scope rather than incremental results, it didn't produce enough feedback along the way to handle change very well. Because of the large scope, users and clients saw little of the actual product or system until the project had passed through all the lifecycle stages. By that time, many things had changed but there had been no opportunities within the waterfall model lifecycle to learn about these changes and respond along the way. That meant UX and SE people had to work very hard and rigorously in each lifecycle activity to minimize the problems and errors that crept in, making it a slow and laborious process.

The culture of this era was that of a very long product or system shelf life because change was too difficult. All this was no problem for the team, though, because they were getting ready to disband and go off to other assignments. But of course, for the client and users it was a different (and sadder) story: goals were missed, some requirements were wrong, other requirements were not met, the system had severe usability problems, and the final deliverable didn't satisfy its intended purpose.

For more about silos, walls, and boundaries and the disadvantages of the waterfall model lifecycle process, see Section 6.6.

4.3 EMBRACING AN AGILE LIFECYCLE PROCESS

Agile Software Engineering

A lifecycle process approach to software implementation based on delivering frequent small-scope operational and usable releases that can be evaluated to yield feedback (Section 29.2).

An agile lifecycle process (UX or SE) is small-scope approach in which all lifecycle activities are performed for one feature of the product or system and then the lifecycle is repeated for the next feature. An agile process is driven by needs formulated as user stories of capabilities instead of abstract system requirements and is characterized by small and fast deliveries of releases to get early usage-based feedback.

Because of the problems with the waterfall process discussed in the previous section, a search for alternative SE processes led to the idea of agile SE. In the waterfall process, you do each lifecycle activity for the entire product or system. In an agile lifecycle process, you do all the lifecycle activities for one feature of the product or system and then repeat the lifecycle for the next feature.

Agile processes are generally fast, very iterative, and responsive to change. This makes sense because the word "agile" means nimble or responsive to change.

Agile processes address:

Chunking

Breaking up of requirements for a product or system into small groups, each corresponding to a release, usually based on features, entailing a set of tasks related to a feature (Section 4.3.1).

- Prerequisite 1 (*In a successful project, the scope needs to be small so the time it takes to deliver a release is limited*) by delivering the first chunk relatively quickly because it takes less time to implement.
- Prerequisite 2 (*In a successful project, the gap between reality (true requirements) and the designer's understanding of the same needs to remain small*) and Prerequisite 3 (*Feedback from actual usage is the only way to know real requirements*) by delivering a small chunk that customers can use, thereby bridging the gap between perceived needs and real needs.
- Prerequisite 4 (*In a successful project, feedback about requirements must be communicated effectively*) by:
  - Formulating the needs as user stories of capabilities instead of abstract system requirements.
  - Making the stories about small manageable features instead of the whole system.

Also, by the time agile processes came along, the world of system development was embracing smaller and less complex systems. While there is some development of the large enterprise or government systems of the waterfall days, development is leaning toward smaller and less complex consumer applications. This is a trend that worked in favor of agile processes.

4.3.1 Scope and Chunking are Key to Real Usage Feedback

Chunking is the breaking up of requirements for a product or system into small groups, each corresponding to a release, usually based on features, entailing a set of tasks related to a feature.

In the software engineering transition from the waterfall process, the chunking of features into a small scope for each iteration of the process was the key to getting the feedback necessary to track changes in understanding that occurred with real usage.

Fig. 4-2, adapted from Kent Beck's Extreme Programming book (Beck, 1999), gives us a visual idea of how truly small the scope is in an agile approach. As the figure shows, the waterfall process makes one big pass through the lifecycle activities for the whole system at a time. An iterative approach takes a step toward smaller, but still fairly large, chunks and results in multiple passes. But it isn't until we get to large numbers of frequent iterations for very small chunks that we see the true essence of an agile approach—agility.

Eric Ries, in his book on lean startups (Ries, 2011), declares that *agility is perhaps the single most important thing in successful product development.* It's not so much about brilliant ideas, technology, far-seeing vision, good timing, or even luck. Instead, *you should have a "process for adapting to situations as they reveal themselves."* Lean UX is one variation of agile UX that focuses on producing a minimum viable product (MVP) in each sprint.

In an agile software engineering process, a sprint is a relatively short (not more than a month and usually less) period within an agile SE schedule in which "a usable and potentially releasable product increment is created."[1] "Each sprint has a goal of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product increment." A sprint is the basic unit of work being done in an agile SE environment. In short, it is an iteration associated with a release (to the client and/or users).

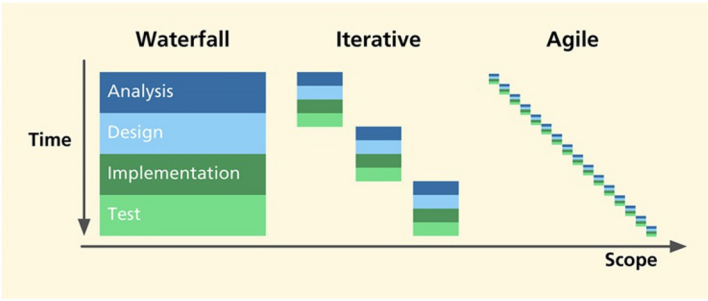

Fig. 4-2
Scope and size of deliverables in the waterfall, iterative, and agile process (adapted with permission from Beck (1999)).

[1] https://www.scrum.org/resources/what-is-a-sprint-in-scrum.

Agile SE processes do this by delivering to the client and/or users meaningful small-scope product or system chunks of capabilities and working software features with every iteration. That way, we never go too far down an unproductive path before we can make a course correction. This constant minimizing of the gap between perceived needs and real needs is how agile processes meet Prerequisite 2.

4.3.2 On the UX Side, We've Always Had a Measure of Agility Without Chunking

Frequent iteration, feedback, learning, and responsiveness to change have always been important goals in UX, too. In fact, the discipline of UX design was built on these as foundational principles. In UX, we didn't need to do chunking to achieve them.

To meet Prerequisite 1, we use fast prototypes to reduce the time it takes from conception to feedback time.

There have been many ways of keeping the gap between user and designer small to meet Prerequisite 2. We run all our models and work products by the client. We have users participate in ideation and sketching for design. We do early and frequent evaluations using low-fidelity prototypes. In other words, we can simulate the experience for the user at a large-scope level and learn from it without building the real system. And, in UX, user feedback is not usually sought a feature at a time. A quality user experience is better achieved and evaluated more holistically, using a top-down approach with a large scope.

UX processes met Prerequisite 3 by simulating real usage via prototypes.

Finally we met Prerequisite 4 (communication of requirements feedback) by using participatory design. In addition, all our artifacts are about usage and user concerns, not system ones.

Participatory Design

A democratic process for UX design actively involving all stakeholders (e.g., employees, partners, customers, citizens, users) to help ensure that the result meets their needs and is usable. Based on the argument that users should be involved in designs they will be using, and that all stakeholders, including and especially users, have equal input into UX design (Section 11.3.4).

4.3.3 But SE Hasn't Had the Luxury of Making User-Facing Prototypes

In a way, this has been easier for the UX people. On the SE side of things, most of the concerns are about system aspects. Therefore, their artifacts are about representations of the inner workings of the system and they tend to be abstract and technical.

On the UX side of things, we are dealing with what the user sees and feels. Therefore, it is easy for us to give the user a flavor of a particular design concept by mocking up these user-facing aspects using prototypes. The SE people couldn't make meaningful low-fidelity mockups. To show something to users in preagile days, they had to build the whole working system.

To be fair, the SE side did experiment with rapid prototyping (Wasserman & Shewmake, 1982a, 1982b), but it was never really part of their classical process models.

4.3.4 And SE Wasn't That Interested in Users, Anyway

Before agile processes, the SE people were not known for talking with users about envisioned behavior with just a sketch or mockup. They didn't have touchpoints with users that focused on the product side of things to situate the users in the design solution domain.

Rather, the SE people historically focused on the process side of things and in technical issues such as code structure, code understanding, and code reusability. If anything, they looked for ways to make it better for the programmer, not the end user.

Their "aha" moment arrived when they figured out that they could flip their approach to delivering chunks that people can use and give product-oriented feedback. In the software engineering (SE) development world agile[2] approaches have quickly become the de facto standard.

4.3.5 So Why Have we in UX Followed SE into an Agile Approach?

There is a practical reason why UX designs still might have to be chunked for delivery to the SE people: to keep pace with sprints on the agile SE side. The SE people are the system builders. Even if we provide designs of the full system at once, they will build it in chunks. So, as a discipline, agile UX has evolved to fit that model of development. We now deliver our UX design chunks to the client, users, product owners, and other stakeholders for direct feedback and we deliver our UX designs in chunks to the SE people for implementation.

See Chapter 29 on connecting agile UX with agile SE for more about the characteristics of agile processes.

4.4 THE FUNNEL MODEL OF AGILE UX

Because of the confusion about how the UX designer needs to work in collaboration with an agile SE process, we created what we call *the funnel model of agile UX, a way of envisioning UX design activities before syncing with agile SE sprints (for*

[2]In the SE world, the term "Agile" has been granted a capital "A," giving it special meaning. Because we don't want to make fine distinctions that don't matter in our context, we won't deify the concept but will stick with the lower case "a."

overall conceptual design in the early funnel) and after syncing with SE (for individual feature design in the late funnel). When we say "agile UX" in this book, we mean agile UX as set within this model, the basis throughout all the process chapters.

4.4.1 Why a New Model Was Needed

Agile UX became the way to bring design for the user experience to agile SE. But there were problems with the way agile UX was approached initially:

- Being agile was interpreted as going fast.
- Following agile SE flow in sprints suffers from a fundamental mismatch with UX concerns.

4.4.1.1 Speed kills: Rapidness and agility are not the same

Sometimes people confuse being agile with being fast. *While an agile process will often be rapid, agility isn't defined by rapidity.* Being rapid just means working faster; being agile is about chunking for design and delivery so we can react to new lessons learned through usage.

Arnowitz (2013, p. 76) cautions us that, while agile processes are almost always associated with speed, a single-minded focus on speed is almost guaranteed to be detrimental to the quality of the resulting user experience. Putting speed above everything else is a reckless response to pressure for rapid turnaround and is "proven to create Frankenstein UIs within a mere two to three iterations. That's speed" (Arnowitz, 2013, p. 76). As an analogy, when you're driving a car and starting to get lost, driving faster won't help.

4.4.1.2 The single biggest problem: UX was expected to follow the agile SE flow completely

The SE world has turned essentially all agile and the UX world has struggled to follow suit. Many people thought agile UX workflow should mirror agile SE flow exactly in order to keep in sync with agile SE development. So the UX teams started churning out chunks of UX design. But a good UX design is holistic, cohesive, and self-consistent and these new agile UX practitioners hadn't done the upfront work necessary to establish a coherent overview. And, by the time they got into sync with the SE sprints, the "design" became fragmented.

Arnowitz (2013, p. 78) says that the way agile processes are usually practiced, they are design-hostile environments. And design is what we do in UX. Embodying the opposite of a holistic view, agile practice can easily promote fragmentation.

What was missing was a way for UX to do some kind of initial large-scope usage research and conceptual design before getting into a small-scope rhythm with SE. There was some literature about this problem, but it was mostly discussed as an exception or special case. In the "funnel model of agile UX" of the next section, we show how to include some upfront usage research and design in a mainstream view of agile UX.

4.4.2 Introducing the Funnel Model of Agile UX

The funnel model of agile UX, shown in Fig. 4-3, has two major parts: the early funnel on the left and the late funnel on the right.

4.4.2.1 Scope in the funnel model

The vertical dimension of the diagram is scope. A larger funnel diameter (taller in the vertical dimension of Fig. 4-3) at any point on the funnel represents a larger scope there. And a small diameter means smaller scope at that point. Fig. 4-3 shows a typical case where the scope of the early funnel is larger than the scope of the late funnel.

4.4.2.2 Speed and rigor in the funnel model

The horizontal dimension of the diagram is time, representing how long activities in the funnel take to play out. The stripes or segments depicted on the funnel visually represent iterations or sprints and the length of a segment represents the duration in time of that sprint and, by implication, the speed of methods and

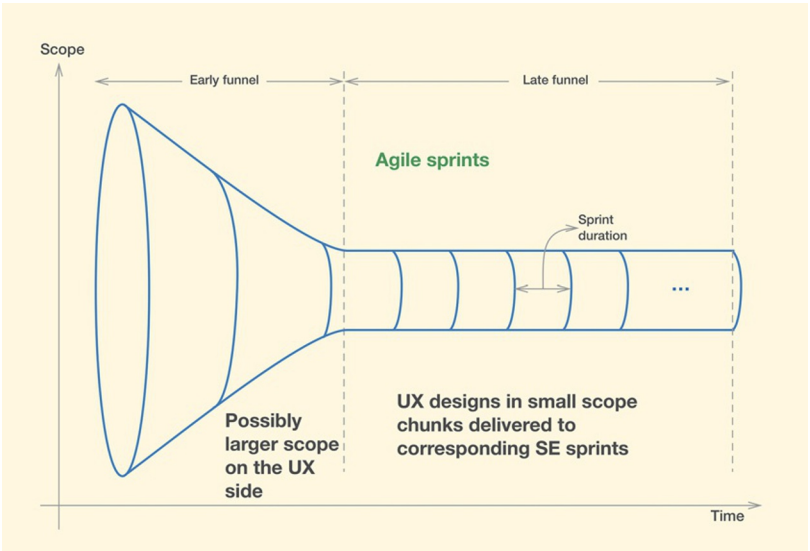

Fig. 4-3
The funnel model of agile UX.

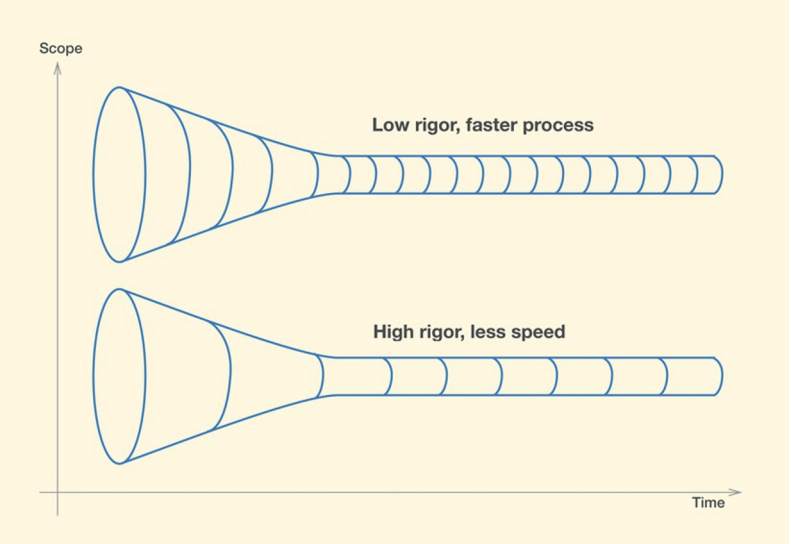

Fig. 4-4
Low rigor and high speed versus high rigor and slow speed in the funnel model.

techniques that have to be used in a given iteration. Longer sprints usually correspond with higher rigor, which will need methods and techniques that are more thorough and meticulous for that iteration (Fig. 4-4).

4.4.3 Late Funnel Activities

The late funnel, or the "spout" on the right side of Fig. 4-3, is where the agile UX and agile SE processes are working in synchronism. Here, the goal of both the UX and SE sides is typically described in terms of small chunks within a small scope (represented by the small diameter of the funnel spout) delivered within a relatively small time increment (narrow sprint duration stripe).

In theory at least, each delivered software chunk is tested until it works and then integrated with the rest. The result of that is regression (looking back) tested until it all works. The idea is to be able to start and end each iteration with something that works (Constantine, 2002, p. 3). Constant and close (but informal) communication produces continuous feedback.

4.4.3.1 Syncing agile UX with agile SE sprints

The UX team provides a design chunk, which the SE team implements along with its design of the corresponding functionality in a sequence of sprints (Fig. 4-5).

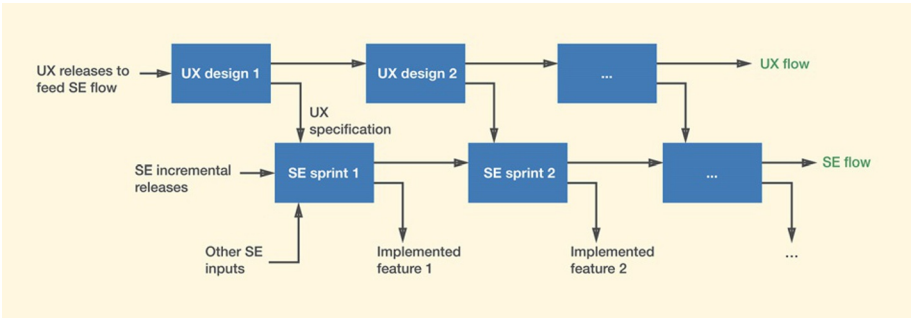

Fig. 4-5
Syncing agile UX with agile SE sprints in the late funnel flow.

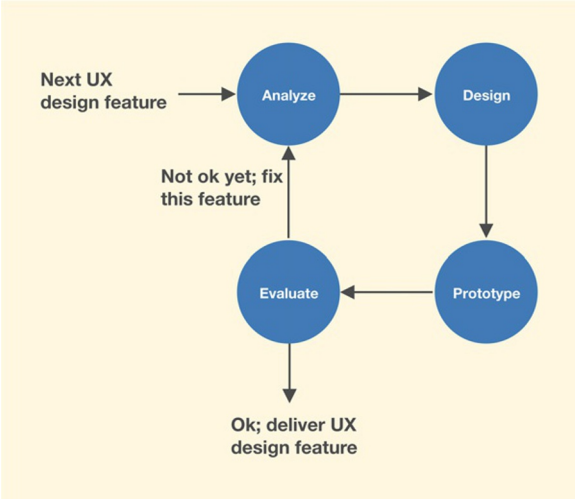

Fig. 4-6
Mini agile UX lifecycle process within a sprint.

Once agile SE gets into the rhythm of going through sprints to produce implementations of chunk as features, the idea of agile UX is to sync with agile SE by providing UX design for each feature in turn, as shown in Fig. 4-5.

Each UX design box contains a mini UX lifecycle such as the one shown in Fig. 4-6.

We'll talk much more about how the agile UX and agile SE processes are coordinated in Chapter 29.

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).

Conceptual Design

A theme, notion, or idea with the purpose of communicating a design vision about a system or product. It is the part of the system design that brings the designer's mental model to life and conveys it to users (Section 15.3).

Top-Down Design

A UX design approach starting with abstract descriptions of work activities, stripped of information about existing work practice and working toward a best design solution independent of current perspectives and biases (Section 13.4).

4.4.4 Early Funnel Activities

Before we can do the small-scope incremental releases in the synchronized late-funnel flow of Fig. 4-3, we have to start with a full-scope analysis and design up front in the early funnel. UX must start the design for a new system with a top-down view to understand the ecology and needs and to establish a conceptual design.

This is a requirement for UX because of the nature of UX design. Unlike code that is invisible to the user, UX design is not. Code is malleable in that it can be structured and refactored in every release. Redoing the design of the UI like that will drive the users crazy.

This upfront UX activity, the solution to the second problem described in Section 4.4.1.2, which is sometimes called "sprint 0" because it precedes the first sprint in the late funnel, is to establish:

- An overview, a skeleton on which to put the features.
- A solid coherent conceptual design to guide the design for the features.
- An initial top-down design.

This will entail as much of these upfront activities as necessary to understand needs and workflow:

- Usage research.
- Analysis.
- Modeling.

This early funnel work is what puts "UX strategic design decisions up front, where they belong" (Arnowitz, 2013, p. 78).

In the next section, we look at some examples of large-scope early funnel cases.

4.4.4.1 The need to establish a conceptual design

A conceptual design is a high-level model or theme of how the system works. It acts as a framework to aid users in acquiring their own mental model of how the system behaves. In any project charged with designing and building a brand new system or a new version of a confusing existing system, you will need to establish a clear conceptual design upfront as scaffolding for a

coherent overall UX design. Conceptual design is, by its nature, a large-scope notion. Small scope at this point is likely to lead to a fragmented conceptual design.

After the conceptual design is established, the UX team can move to a small scope in the late funnel to deliver the detailed UX designs of individual features to the SE side in increments.

Example: Developing a New Smartphone Design From Scratch

Suppose your project is to create a brand new smartphone to compete with the current market leaders. It has been decided that this entails an entirely new and innovative conceptual design—a design that is better, and more exciting to consumers, than existing options offered in the market.

This case might require a significant large-scope effort in the early funnel to create a full conceptual design, to set the overall ecology of the smartphone, and to create a cohesive design upfront. It's just not possible to design a phone operating system or a brand new consumer-facing application without beginning with a large-scope design on the UX side. Beyond the point where you have established a consistent conceptual design, the project can then adopt the usual small scope in the late funnel to deliver the UX design.

As an interesting note in this case, even after the UX and SE roles end up in lockstep in the late funnel as they release chunks of the smartphone operating system, end users may not see those chunks. This is a case where the learn-through-frequent-customer-releases method doesn't necessarily work because you cannot release a new smartphone to end users in chunks.

4.4.4.2 Small systems with low complexity

Small systems with low complexity are an example of using a large scope, but for a different reason—because the system has low complexity and can be handled in a single shot.

Small systems with low complexity don't usually benefit much from a small-scope approach on the UX side. So, in fact, you can use a large scope through the whole funnel (picture the whole funnel being almost as wide as a wide mouth). If the system isn't large, there may not be enough of a "pipeline" or breadth of system features or complexity for the UX team to chunk the designs into small-scope increments. In other words, the size and complexity of the system can easily be handled with a large scope in the whole funnel and delivered

to the SE side in large scope, leaving them to decompose it into features for their own small-scope implementation, if necessary.

4.4.4.3 SE needs a funnel model, too

The people on the SE side also need to do some upfront analysis and design, at least to establish the system architecture. So, maybe there are really two overlapping funnels.

4.4.4.4 The nexus of early and late parts of the funnel

The transition to the late funnel is a crucial point in a project; it's the beginning of "crunch time," where you get in gear and sync with the agile SE people. Now is the time that user stories drive your design iterations, but the deep understanding (in total) that you developed in usage research and modeling informs the design.