CHAPTER

UX Design Requirements: User Stories and Requirements

10

Highlights

- The concept of requirements for UX design.
- User stories as the agile way to express UX design requirements.
- UX design requirements are needed in rare situations.
- Validate user stories and requirements with users and stakeholders.

10.1 INTRODUCTION

10.1.1 You are Here

We begin each process chapter with a "you are here" picture of the chapter topic in the context of The Wheel, the overall UX design lifecycle template (Fig. 10-1). Within the Understand Needs UX design lifecycle activity, this chapter is about the subactivity in which you codify as user stories and UX requirements, the UX design wants and needs you discovered in usage research.

10.1.2 User Stories and Requirements Are About Codifying UX Design Wants

The point of usage research (all of Part 2) is to discover and learn about the UX design wants and needs. The point of user stories and requirements (this chapter) is to codify those wants and needs as inputs to the design of the target product or system. There are several ways to express these design needs, including use cases, user stories, and requirements. In this chapter, we stress the latter two.

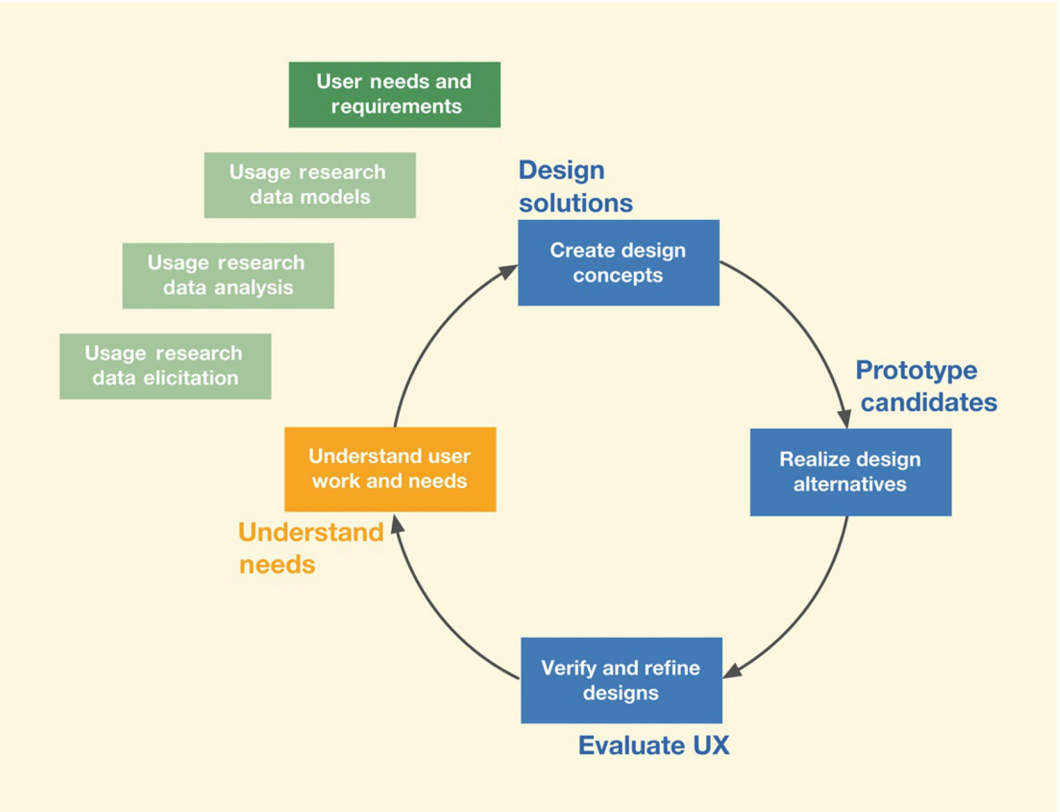

Fig. 10-1

You are here in the chapter on the user stories and UX requirements subactivity within the Understand Needs box, in the context of the overall Wheel UX lifecycle process.

10.1.3 Introduction to User Stories

User stories are an agile way of capturing what is to be designed. User stories:

- Involve user inputs.
- Are stated in the perspective of usage.
- Are stated from a customer perspective.
- Were made popular by agile proponents.
- Are now the de facto standard in industry.
- Are focused on delivering chunks of meaningful capabilities to users.

There are other means for stating design needs from these perspectives, but user stories are the popular way.

10.1.4 Introduction to Requirements

As we said in Section 4.2, back in the old days, when the Waterfall process was widespread, it was traditional to codify what is needed in the system in formal statements called requirements. Those requirements were targeted as features the system should support. There was a flavor of system-centeredness.

Today formal requirements are rarely used and, when they are, it is usually because of a focus on the software engineering (SE) side, where there is a huge subdiscipline dedicated to requirements.

Waterfall Lifecycle Process

One of the earliest formal software engineering lifecycle processes; an ordered linear sequence of lifecycle activities, each of which flowed into the next like a set of cascading tiers of a waterfall (Section 4.2).

10.1.5 Choose the Approach You Need

We have two major sections in this chapter: one on user stories and one on requirements. Choose one depending on your design situation. If you need to use a rigorous full-scope approach because you are working in the complex-interaction and complex-work-domain quadrant of the system complexity space or if you are contractually obligated to do so, adopt formal requirements. Otherwise, choose user stories, especially for an agile process.

System Complexity Space

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

10.1.6 Requirements in the UX World

10.1.6.1 Requirements as design goals, not constraints

In the old days, requirements were immutable and absolute. "You *shall* provide these features and functions and you *shall* do it in the way we say." This dictatorial view of requirements left no room for creativity in design solutions and no flexibility to adapt to change as it inevitably occurred.

But today's UX mentality is all about design (Kolko, 2015b); we value design over requirements. Rather than even thinking about requirements, we should think about how great products "require vision, perseverance, and the ability to do amazing things in the face of perceived impossibility" (Kolko, 2015b, p. 22).

In today's agile environments, we like to view requirements, especially UX design requirements, not as constraints but as design goals with the freedom to interpret *how* the design should achieve these goals (Kolko, 2015b).

Scope (of Delivery)

Describes how the target system or product is "chunked" (broken into what size pieces) in each iteration or sprint for delivery to the client and users for feedback and to the software engineering team for agile implementation (Section 3.3).

10.1.6.2 UX requirements versus UX design prototypes as SE requirements

To some extent, UX and SE share requirements because both UX and SE participate in the ultimate outcome: the delivered system. But UX people also have their own design requirements (far left side of Fig. 10-2). The SE people, on the right side of the figure, are responsible for implementing everything, so they will have requirements for:

1. The UX parts.
2. The functional parts.

Wireframe Prototype

A prototype composed of wireframes, which are line-drawing representations of UX designs, especially the interaction design of screens (Section 20.4).

The SE people have evolved their own ways of understanding the functional or backend requirements from a user story about a given feature. These functional requirements are to support the tasks, navigation, etc., being designed on the UX side. As a practical matter, UX prototypes—usually wireframe prototypes—are the vehicle for conveying requirements for the UX parts to the implementers (center of Fig. 10-2).

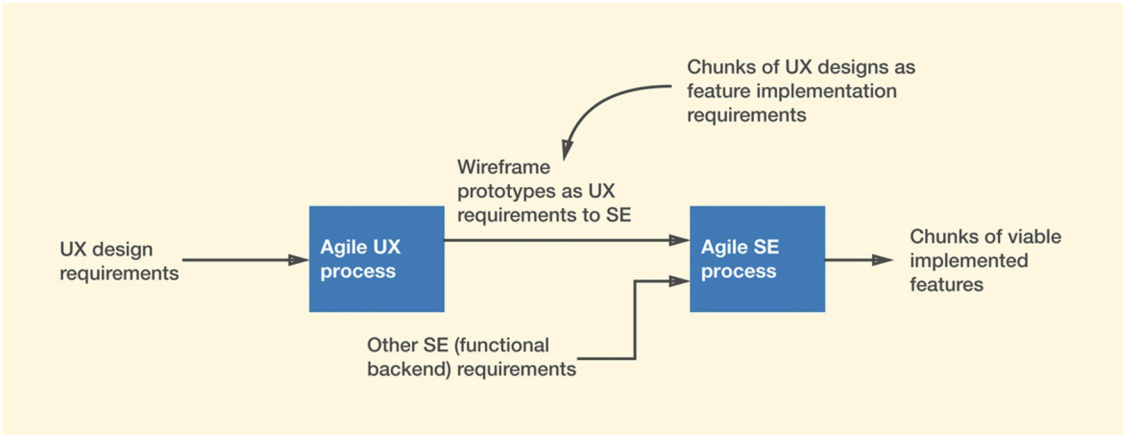

Fig. 10-2

The relationship between UX design requirements and UX prototypes as UX design representations for SE requirements.

10.1.6.3 Software and functional implications of UX design requirements

Because of the close relationship of requirements in UX and SE, this activity is a good time to coordinate among UX, SE, and your client. In fact, it is a good time to be proactive and make sure the SE team is aware of functional requirements on their side that are reflected by task requirements on your UX side.

10.1.7 Formal Requirements are Waning in Popularity

For decades, formal requirements and requirements engineering (Young, 2001) have been a staple component of software engineering and a formal written requirements document was *de rigueur*.

But now, even in the SE world, the importance of requirement specifications is fading, primarily because they don't really work (Beck & Andres, 2004). There is an increasing recognition that:

- Detailed formal requirements cannot ever be complete.
- Detailed formal requirements cannot ever be 100% correct.
- Detailed formal requirements cannot be prevented from changing throughout the lifecycle.

But in both UX and SE, we still need some kind of design requirements, and that is where user stories come in.

10.2 USER STORIES

In today's world, user stories are the de facto approach to both UX design and SE implementation requirements in late-funnel agile development. In usage research analysis, you gathered up a collection of all elemental data notes related to user stories and requirements (Section 8.4). Now is the time to convert them into real user stories and/or UX design requirements. Codifying UX design requirements can be considered the final step of the Understand Needs lifecycle activity.

10.2.1 The Truth About User Stories

10.2.1.1 Asking users what they wanted was originally discouraged

One of the more serious problems with user stories starts with the definition, which is billed as a statement by a user about something (a feature or capability) a user wants. But the original theory of contextual inquiry (usage research) (Beyer & Holtzblatt, 1998) is based on UX analysts determining what users *need* to

support their work. The original pure contextual inquiry was not about users saying what they want because they are not trained UX designers and it's our job to determine user needs.

10.2.1.2 How can user stories make for complete requirements?

Even if we can get good UX requirements as user stories from users, it's unlikely that users would present us with a *complete* set of requirements. So we have to conclude that not all user stories will be coming from users. We get what we can from users, but the goal is not completeness. We, as UX research analysts, must write additional user stories to fill in the gaps.

10.2.1.3 Cleaning up the user stories

UX practitioners must edit existing user stories to make them effective in our process. This means we usually have to change what the user said to make it more accurate, more general, more complete, and more sensible—to the point where they usually end up not really being something a user said at all. So, as a practical matter, those in the field have resorted to writing user capability requirements as user stories as though users had expressed them, giving the whole collection a uniform format.

Irrespective of who wrote them, user stories capture a need, and they are effective at naturally dividing up requirements into items of small scope.

10.2.2 What is a User Story?

A user story is a short narrative describing a feature or capability needed by a user in a specific work role and a rationale for why it is needed, used as an agile UX design "requirement."

User stories arise out of usage research data and they can be based on problems users have with an existing product or system or they might reflect needs for new features in a new design. Now it's time to get your work activity notes related to user stories out again as inputs to writing the user stories.

As a practical matter, user stories describe existing everyday product or system usage or desired capabilities. A user story is a kind of small-scope requirement (Section 3.3). Gothelf and Seiden (2016) define a user story as a narrative in the user's voice describing, "the smallest unit of work expressed as a benefit to the end user" (minimum viable product or MVP).

As such, a user story describes a single capability needed for a specific user work role with a specific benefit. Following this definition, a user story tends to be small scope and atomic. Because of its small size and scope, some say a user story should fit on one $3'' \times 5''$ card.

Scope (of Delivery)

Describes how the target system or product is "chunked" (broken into what size pieces) in each iteration or sprint for delivery to the client and users for feedback and to the software engineering team for agile implementation (Section 3.3).

10.2.3 Team Selection

Gather a multiperspective team to write user stories from elemental data note inputs. UX designers, of course, should be well represented, as should be the customer, client, and user perspectives.

10.2.4 Writing a User Story

In our terminology, this translates to a brief narrative statement of a single user need, stated in the voice of a given user work role and in this format:

As a <relevant user work role>
I want to <the desired small-scope capability in everyday product or system usage>
So that <the reason why, the added value this capability will deliver>.

Work Activity Note

A brief, clear, concise, and elemental (relating to exactly one concept, idea, fact, or topic) statement used to document a single point about the work practice as synthesized from raw usage research data (Section 8.1.2).

Example: User Stories for TKS

Consider this raw data note:

> I think of sports events as social events, so I like to go with my friends. The problem is that we often have to sit in different places, so it is not as much fun. It would be better if we could sit together.

This is typical of raw data notes. It is a bit more rambling than concise, so you have to work at it a little to extract the essence of a corresponding user story:

As a student ticket buyer
I want an MU basketball ticket-buying option to get several seats together
So I can sit with my friends.

This is definitely about a desired small-scope capability. It refers to one choice in seat selection as a parameter of the ticket-buying task.
As a further example, consider this raw data note:

> I sometimes want to find events that have to do with my own personal interests. For example, I really like ice skating and want to see what kinds of entertainment events in the nearby areas feature skating of any kind.

This is pretty specific, but you can deduce a slightly more general version for your user story:

As a ticket buyer
I want to be able to search for events by event type or descriptive keyword
So I can find the kinds of entertainment that I like.

Here is another raw data note:

> I occasionally find it convenient to see recommendations for events that are similar to ones I like, like in the listing of what others who looked at this item got from Amazon.com.

This can become a user story for a specific feature:

As a ticket buyer
I want to be able to see recommendations for events similar to ones I find and like
So I can find other events I like that I might otherwise miss.

This user story as a UX requirement has a built-in implied backend system requirement, namely that during a transaction session, the Ticket Kiosk System (TKS) functional software will have to keep track of the kinds of choices made by the ticket buyer along with the choices of other ticket buyers who bought this item. This similar capability at Amazon is a design model for this feature.

10.2.5 Extrapolation Requirements in User Stories: Generalization of Usage Research Data

User statements in raw data notes can be quite narrow and specific. Sometimes you will need to extrapolate from these statements to get a more useful and general case.

For example, suppose ticket buyers using MUTTS, in anticipation of a kiosk, expressed the need to search for events based on a predetermined criterion but said nothing about browsing events to see what is available. So you might write an extrapolation requirement to include the obvious need also to browse events as an extrapolation of the previous user story about being able to search for events by keywords.

MUTTS

MUTTS is the acronym for Middleburg University Ticket Transaction Service, our running example for most of the process chapters (Section 5.5).

As a ticket buyer
I want to be able to browse events with filters for category, description, location, time, rating, and price
So I can find relevant events in context.

As another example, in a raw data note, a ticket buyer says:

> *I really would like to be able to post an MU football ticket for exchange with a ticket in another location in the stadium to be able to sit with my friends.*

In our extrapolated user story, we broadened this to:

As a ticket buyer
I want to be able to post, check status of, and exchange student tickets
So I can change reserved tickets for seats where I can sit with my friends.

This user story implies a corresponding system requirement for customer accounts of some kind, presumably where they can log in using their MU Passport IDs.

Example: Inputs to User Stories as UX Design Requirements for the TKS

Here are some examples of elemental data notes for the ticket buyer work role in the TKS that are on the way to being user stories, along with their first- and second-level structural headings. Space limitations preclude listing the huge number of user stories written for the TKS, but you should get the idea.

Transaction Flow

Existence of feature

I want to be able to pause, save, and later resume a ticket-buying transaction.

System implications: Will require accounts, including logging in and out, etc.

Shopping Cart Issues

Accessibility of shopping cart

I want to be able to view and modify the shopping cart at all times.

Shopping cart versatility

I want to be able to add different kinds/types of items (example, different events, sets of tickets for the same event, etc.).

Note: This is important because it has implications on how to display shopping cart contents with dissimilar types of objects.

Transaction Flow

Timeouts

I want the system to have a timeout feature that removes my transaction from the screen if I haven't made any actions after a certain amount of time.

So it will protect my privacy.

Extrapolation: Ticket buyer shall be made aware of the existence and status of timeout, including progress indicator showing remaining time and audible beep near the end of the time-out period.

Extrapolation: User shall have control to reset the timeout and keep the transaction alive.

Extrapolation: User's need to keep transaction alive shall be supported by automation; any user activity triggers reset.

Immediate exit

I want to be able to make a quick exit and reset to the home screen.

So, if I have to leave (e.g., to catch a bus) suddenly, I can quit in the middle of a transaction.

Corollary to consider: User shall have a way to quickly return later to a specific item they were viewing just prior to an immediate exit.

Notes: Maybe user will use an event ID number for direct access next time. Or the system can remember state by using an account.

Revisiting previous steps (going back)

I want to be able to revisit a previous step easily.

So I don't have to go through multiple back actions.

Transaction progress awareness

I want to be able to track the progress of an entire transaction (what is done and what is left to do) using a "bread crumb trail."

So I don't have any confusion as to what I still have to do for a successful completion.

User reminders

I want to receive a reminder to take the ticket(s) and credit card at the end of each transaction.

So I don't forget and walk away, thinking I have completed the transaction.

Checkout

I want to see, before making a payment, a confirmation page showing exactly what is being purchased and the total cost.

So I can be confident in what I am purchasing.

10.2.6 Organize Sets of User Stories for Use in UX Design

Hierarchical Task Inventory (HTI)

A hierarchical structural representation of task and subtask relationships for cataloguing and representing the hierarchical relationships among tasks and subtasks that must be supported in a system design (Section 9.6).

In real-world projects, you may end up with a large number of diverse user stories. You need a way to organize them. We organize user stories by a kind of design affinity, grouping user stories by sets of related features or by related tasks. Because UX user stories are stories about user capabilities, they are closely related to user tasks and you can use structures similar to the hierarchical task inventory to organize them. This allows us to consider related features together. For example, we should look at the browse-by-event-type task at the same time we look at browsing by location or searching by event type.

Organizing around the HTI is also a way to manage the collection of user stories and helps us maintain completeness of the collection. This reliance on having an HTI is one big reason we need the early part of the funnel model.

Early Funnel

The part of the funnel (agile UX model) for large-scope activity, usually for conceptual design, before syncing with software engineering (Section 4.4.4).

Think about an HTI-like user story structure with the individual user stories at the lowest level. A group of user stories having the same parent node in the structure will be related by being about the same user task. User stories having the same grandparent node will be a larger group, related to a broader task, and so on. When you consider user stories for UX design, you usually will select the user

stories related to the same parent or, in some cases for a large sprint, the same grandparent.

It is important to note, however, that this hierarchical task-oriented configuration of user stories is for organizing only, and is not a way to prioritize them for design and development. For that, see the next section, Section 10.2.7.

Sprint

A relatively short (not more than a month) time period within an agile software engineering schedule in which a working and releasable product increment is implemented. It is a unit of work being done in an agile SE environment; an iteration associated with a release (to the client and/or users) (Sections 3.3, 29.3.2, and 29.7.2).

Example: Group of User Stories Related to Finding (Browsing and Searching for) Events

Consider the partial user story structure for the ticket buyer work role, as shown in Fig. 10-3.

The box at the lower left contains a group of user stories that are related to the user tasks of browsing and searching to find an event of interest for which to buy tickets. This box can also later hold links to wireframes representing the corresponding UX designs, as each box is prioritized and built.

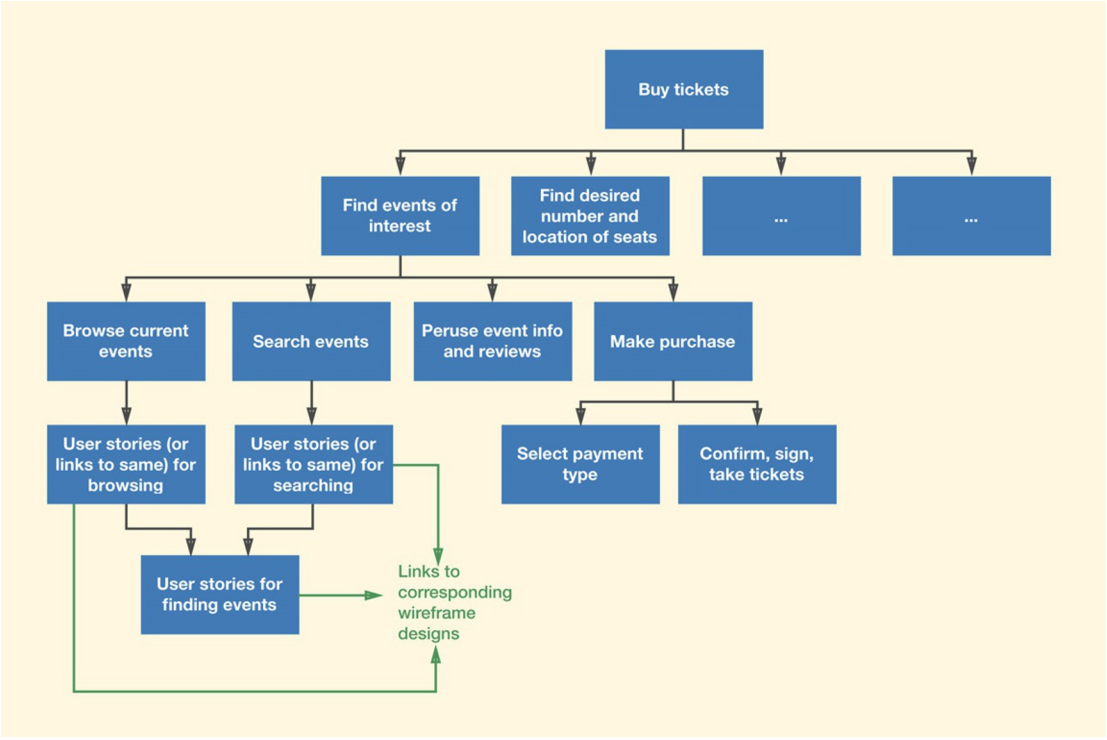

Fig. 10-3
A partial user story structure for the ticket buyer work role.

This box will contain, possibly among others, the user stories corresponding to the following example requirement notes taken from the example following Section 8.7.7.

Searching

83: I would like to be able to search for events by price, artist, venue, and/or date.

104: If I am looking for a specific band, I expect a Google-style search where I type in the name of the band.

112: It would be really nice to be able to search for interesting events say within two blocks from where I am currently.

Browsing

74: I would like to be able to browse all events on the kiosk.

106: I would like to be able to browse by different topics (type of event). For example, in big cities, even in music there will be different genres.

107: I would like to be able to browse by locations.

Sorting

75: I would like to be able to sort the results I find using some criteria.

The corresponding user stories are easy to visualize as:

Searching

User Story TKS 83:

As a ticket buyer
I would like to be able to search for events by price, artist, venue, and/or date
Because I want maximum flexibility in searching.

User Story TKS 104:

As a ticket buyer
I expect a Google-style search where I type in, for example, the name of a band
Because I don't want to be constrained by special formats in searches.

User Story TKS 112:

As a ticket buyer
I want to be able to search for events by proximity to a given location
Because I seek maximum power in searching.

Browsing

User Story TKS 74:

As a ticket buyer
I would like to be able to browse all events on the kiosk
Because sometimes direct searches don't reveal enough context.

User Story TKS 106:

As a ticket buyer
I would like to be able to browse by different event characteristics, such as music genre
So I can find events I like when I only know the types of events I'm looking for.

User Story TKS 107:

As a ticket buyer
I would like to be able to browse by locations
So I can find something nearby, not requiring transportation.

Sorting

User Story TKS 75:

As a ticket buyer
I would like to be able to sort the results I find using some criteria
Because sometimes searching produces too many results to go through without any ordering.

10.2.7 Prioritizing User Stories for Design and Development

In the previous section, we organized by related tasks to manage the collection of user stories. This is not the same as prioritizing for design and development. The task-oriented structure is just an organization scheme, but not necessarily an execution plan, meaning that we don't attempt to design and develop the complete user stories at once in one release.

Rather, this prioritization is guided by a business-oriented concept, namely of delivering minimum viable product (MVP) releases—that is, to produce something initially, however limited, that can be deployed immediately and evaluated to get feedback from actual use. For example, a system might get only a browse capability but no search function in the first version, or a first version might accept only credit cards but not other forms of payment such as the MU passport card. Such an MPV is how you get user feedback quickly, which is the whole idea of an agile process.

Early Funnel

The part of the funnel (agile UX model) for large-scope activity, usually for conceptual design, before syncing with software engineering (Section 4.4.4).

For this kind of prioritizing of user stories, Patton (2014) uses a construct he calls user story maps, which are well described in his book but are beyond our scope here.

At design time, even though the UX design team might deliver the designs of detailed tasks for each of these user stories separately, it is very likely that they will all be considered together as a group because they are so closely related. This is exactly the reason why UX cannot always do agile as well as SE. Hence the need for early funnel to organize the user stories by related tasks in a hierarchical structure such as the HTI before designing and implementing them in sprints. In a purely agile shop, though, UX may not have a choice and may be forced to do fragmented design.

This topic of prioritizing for design is not in the scope of this chapter. See Chapter 29 for more about how user stories are prioritized for design and implementation.

10.3 UX DESIGN REQUIREMENTS

Your user story structure will be an effective approach to UX needs for user capabilities. Do you even need more formal requirements? It's rare, but you might use formal requirements if:

- You have a need for them in your own process.
- If your customer or client must see a requirements document.
- A formal requirements document is a contractually required deliverable.

And, if you have to do formal requirements for UX design, this section tells you how.

10.3.1 Degree of Formality Can Vary

System Complexity Space

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

The formality of requirement statements can vary depending on project needs. Formal design requirements can be necessary in some projects, including projects:

- In the high-rigor quadrants of the system complexity space.
- With constraints for compliance to certain standards.
- With a high need to avoid risk.

For projects not requiring maximum rigor, a simple structured list will suffice as an effective way to keep track of the system capabilities for which we will be designing.

10.3.2 Team Selection

For creating requirements, a broad team is best. Select a cross-disciplinary team, including UX designers, software people, and client representatives, plus possibly system architects and maybe managers.

10.3.3 The Requirements Structure Evolves

As you edit elemental data notes used as inputs to requirement statements, you will bring them together into a requirements structure, organized hierarchically by issues and features, much like the HTI. In addition, many of the categories in the WAAD will represent design needs. The hierarchical categories will evolve as needed to accommodate each new elemental data note.

Example: Initial TKS Requirements Structure

The following is just an example of what we might see emerging as an initial requirements structure for TKS:

Personal privacy and trust issues.

Items in the artifact model (e.g., printed tickets, credit cards).

Items in the physical environment model (e.g., physical aspects of kiosks and their locations).

Business issues, decisions:

- Branding and appearance.
- Kiosk locations.
- Credit card usage.
- Cash transactions.
- Printing tickets.
- Keyboards versus touchscreens.
- Kinds of events supported (including restaurant reservations and transportation tickets).
- Help, customer support.
- Customer accounts and logins.

Artifact Model

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

10.3.4 Compose Requirements Statements

As you will see, most applicable elemental data notes from usage research will be easy to express as requirements. As you consider each note:

- Ask what user needs are reflected by the note.
- Write it as a requirement statement.
- Find where it fits into the evolving requirements structure.

10.3.5 The Requirement Statement and Requirements Document

A generic structure of a requirement statement that has worked for us is shown in Fig. 10-4. A requirements document is essentially a structured set of requirement statements organized on headings at two or more levels.

Name of major feature or category within structure

Name of second-level feature or category

Rationale (if useful): Rationale statement (the reason for the requirement)

Note (optional): Commentary about this requirement

Fig. 10-4
Generic structure of a requirement statement.

We show two levels of headings, but you should use as many levels as necessary for your requirements.

Not every point in a data note will generate a need or requirement. Sometimes one note can reflect more than one need. A single need can also lead to more than one requirement. Examples of data notes, user needs, and corresponding requirements are coming soon.

Example: Writing a TKS Elemental Data Note as a Requirement Statement

Consider this usage research data note from the TKS:

I am concerned about the security and privacy of my transactions.

Concern about security and privacy is a system-wide issue, so this note can directly imply a high-level design requirement statement:

Shall protect the security and privacy of ticket-buyer transactions.

Note that at this level, requirements can involve both UX and functional requirements.

Finally, it is easy to fit this into the requirements structure, assuming there is already a category for personal privacy and trust issues, as shown in Fig. 10-5.

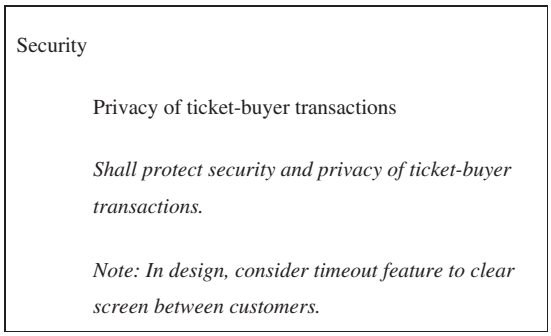

Fig. 10-5
Example requirement statement.

Example: Generalizing Data Notes to Express TKS Needs as Requirement Statements

If the elemental data note is very specific, you might have to abstract it to express a general need. Consider this data note:

> Normally, I buy tickets at the venue of the event just before the event. For special events, I might buy online. Then I usually have to buy Metro tickets to get there at a different kiosk in a totally different location. If I could buy all these different tickets at one kiosk, that would give me one-stop shopping for events.

This is about treating events as broad entertainment activities and having sufficiently broad coverage of tickets for "event content" so the kiosk can be a one-stop shopping location for entertainment events. This means extending "event content" to include, for example, restaurant reservations and transportation tickets in conjunction with events, as in Fig. 10-6.

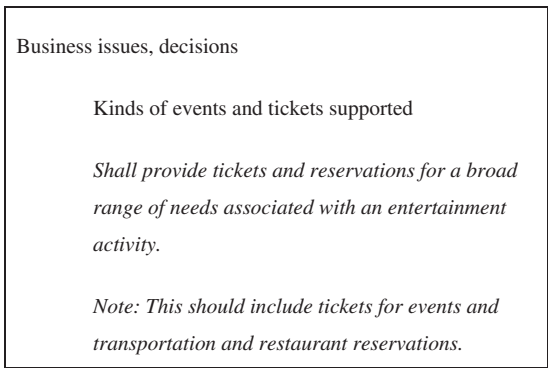

Fig. 10-6
Example requirement statement.

Example: A Possibly Unexpected TKS Requirement

Data note:

> Although marketing people might not think to put a kiosk next to the ticket office, it would be very helpful for people who get there and find the ticket office too busy or closed.

This data note, about locating a kiosk near a ticket office, might reveal an unexpected need because you might not think of it. A possible resulting requirement statement is shown in Fig. 10-7.

Business issues, decisions

Kiosk locations

Shall provide a kiosk located near the ticket office.

Note: This requirement serves users when the ticket office is too busy or is closed.

Fig. 10-7
Example requirement statement.

Emotional Impact

An affective component of user experience that influences user feelings. Includes such effects as enjoyment, pleasure, fun, satisfaction, aesthetics, coolness, engagement, and novelty and can involve deeper emotional factors such as self-expression, self-identity, a feeling of contribution to the world, and pride of ownership (Section 1.4.4).

Work Activity Note

A brief, clear, concise, and elemental (relating to exactly one concept, idea, fact, or topic) statement used to document a single point about the work practice as synthesized from raw usage research data (Section 8.1.2).

10.3.6 Things to Look for in Your Requirements Notes

10.3.6.1 Keep an eye out for emotional impact requirements and other ways to enhance the overall user experience

When writing requirements, don't forget that we are on a quest to design for a fabulous user experience and this is where you will find opportunities for that, too.

Work activity notes with user concerns, frustration, excitement, and likings offer opportunities to address emotional issues. Especially look out for work activity notes that make even an oblique reference to "fun" or "enjoyment" or to things such as data entry being too boring or the use of colors being unattractive. Any of these could be a clue to ways to provide a more rewarding user experience. Also, be open minded and creative in this phase; even if a note implies a need that is technologically difficult to address, record it. You can revisit these later to assess feasibility and constraints.

10.3.6.2 Questions about missing data

Sometimes, as you go deeper into the implications of usage research data, you realize there are still some open questions. For example, in our usage data elicitation for MUTTS, while we were putting together requirements for the

accounting system to aggregate sales at the end of the day, we had to face the fact that the existing business manages tickets from two independent systems. One is the local ticket office sales and the other is from the national affiliate, Tickets4ever.com. During our usage data elicitation and analysis, we neglected to probe the dependencies and connections between those two and how they reconciled sales across those two systems.

MUTTS

MUTTS is the acronym for Middleburg University Ticket Transaction Service, our running example for most of the process chapters (Section 5.5).

10.3.6.3 System support needs

You may also occasionally encounter system requirements for issues outside the user experience or software domains, such as expandability, reliability, security, and communications bandwidth. These are dealt with in a manner similar to that used for the software requirement inputs.

Example: System Support Requirements for TKS

A few examples from the TKS WAAD illustrate:

> Work activity note: "Identity theft and credit card fraud are huge concerns for me."
> System requirement: "System shall have specific features to address protecting ticket buyers from identity theft and credit card fraud." (This "requirement" is vague but it is really only a note for us to contact the systems people to figure out potential solutions to this problem.)

Legacy System

An old process, technology, computer system, or application program, of, relating to, or being a previous or outdated computer system with maintenance problems that date back possibly many years (Section 3.2.4).

Another systems constraint for the existing working environment of MUTTS is the necessity of keeping the secure credit card server continuously operational. An inability of the ticket office to process credit card transactions would essentially bring their business to a halt. That constraint will be even more important in the transition to TKS, where human operators won't be present to notice or fix such problems.

Here is another example of how a UX requirement also creates a system support requirement, which you should capture here:

> UX requirement: "Ticket buyers shall be able to see a real-time preview of available seating for a venue."
> Corresponding system requirement: "System shall have networked infrastructure and a common database to 'lock and release' selected seats in a given venue for a given date and time and to update ticket and seat availability as kiosk transactions occur."

This is a good time for the software team members to work in parallel with your team to capture those inputs to software and system requirements so they are not lost.

10.3.6.4 Constraints as requirements

Constraints, such as from legacy systems, implementation platforms, and system architecture, are a kind of requirement in real-world development

projects. Although, as we have said, much of the UX design can and should be done independently from concerns about software design and implementation, your UX design must eventually be considered as an input to software requirements and design.

Therefore, eventually, you and your UX design must be reconciled with constraints coming from systems engineering, hardware engineering, software engineering, management, and marketing. Not the least of which includes development cost and schedule as well as profitability in selling the product.

What restrictions will these constraints impose on the product? Will kiosk size or weight be taken into account if, for example, the product will be on portable or mobile equipment? Does your system have to be integrated with existing or other developing systems? Are there compliance issues that mandate certain features?

Example: Physical Hardware Requirements for TKS

Consider this TKS example about privacy:

> Work activity note: "When I am getting tickets for, say, a controversial political speaker, I do not want people in line behind me to know what I am doing."
>
> Physical hardware requirement: "Physical design of kiosk shall address protecting privacy of a ticket buyer from others nearby."

Here are some other example physical hardware requirements that might be anticipated in the TKS:

- Special-purpose hardware needed for the kiosk.
- Rugged, "hardened" vandal-proof outer shell.
- All hardware to be durable, reliable.
- Touchscreen interaction, no keyboard.
- Network communications possibly specialized for efficiency and reliability.
- If have a printer for tickets (likely), maintenance must be an extremely high priority; cannot have any customers pay and not get tickets (e.g., from paper or ink running out).
- Need a "hotline" communication feature as a backup, a way for customers to contact company representatives in case this does happen.

Exercise 10.1: Constraints for Your Product or System

Goal: Get a little experience in specifying constraints for system development.

Activities: Extract and deduce what you can about development and implementation constraints from contextual data for the product or system of your choice.

Deliverables: A short list of same.

Schedule: A half hour should do it.

Example: UX Design Requirements for the TKS

Here are some example UX requirements for the TKS, along with their first- and second-level structural headings. Space limitations preclude listing the huge number of requirements for the TKS, but you should get the idea.

User Accounts

Existence of accounts

User shall be able to create accounts through a web interface and access them through the kiosk. (optional, future?).

Shopping Cart

Existence of feature

Ticket buyer shall have a shopping cart concept with which they can buy multiple items and pay only once.

Reserved Seat Selection

Existence of feature

Ticket buyer shall have a shopping cart concept with which they can buy multiple items and pay only once, and be able to select seats from all available seats in various price categories.

Implied requirements: Display of seat layout within event venue. Display seating availability and to be able to filter that list of available seats by categories such as price.

Another important implied requirement: Seat selection requires the existence of a lock-and-release mechanism of some sort, something we perhaps did not yet have in the requirements structure. This is a technical requirement to give the buyer a temporary option on the selected seats until the transaction is completed or abandoned.

User Checkout

Use of cash, credit cards

Ticket buyer shall be able to use cash, credit cart, or debit card for payment.

Notes: Cash transactions have several drawbacks: Cash denominations are difficult to recognize, counterfeit bills are difficult to identify, dispensing change can be problematic, and cash in the kiosk can attract vandals and thieves.

Exercise 10.2: Writing Requirement Statements for Your Product or System

Following the descriptions in this chapter, write a few formal requirements statements for the product or system of your choice.

Goal: Get some practice with requirements extraction.

Activities: Assemble your UX team in your UX studio with:

- The elemental data notes you have set aside as inputs to requirement statements.
- Your WAAD for your product or system.

Choose a leader and recorder.

Do a walkthrough of the elemental data notes and the WAAD.

For each work activity note in the WAAD, work as a team to deduce user need(s) and UX design requirements to support the need(s).

As you go, have the recorder capture requirements in the format shown in this chapter, including extrapolation requirements and rationale statements, where appropriate.

To speed things up, you might consider having each person be responsible for writing the requirement statements extracted from a different subtree in the WAAD structure. Set aside any work activity notes that require additional thought or discussion to be dealt with at the end by the team as a whole.

If time permits, have the whole team read all requirement statements to assure agreement.

Deliverables: A requirements document covering at least one subtree of the WAAD for your system.

Notes and lists of the other kinds of information (discussed in this section) that come out of this process.

Schedule: We expect that this exercise could take at least a couple of hours. If you simply do not have that kind of time to devote to it, do as much as you can to at least get a flavor of how this exercise works.

10.4 VALIDATING USER STORIES AND REQUIREMENTS

After the user stories are gathered and organized into a hierarchical user story structure, it's a good time to take them back to users and other stakeholders to be sure they are the right ones. It's also a good time to coordinate between the UX and SE teams.

10.4.1 Coordinating Requirements, Terminology, and Consistency

Many UX requirements for tasks imply an SE (backend) requirement for functionality, and vice versa. Both teams need to be working toward the same set of tasks/functions. This is also a key time to standardize terminology and build consistency. Your usage research data will be full of user comments about usage and design concepts and issues. It is natural that they will not all

use exactly the same terms for the same concepts. The same is true for the SE and systems people and other stakeholders.

For example, users of a calendar system might use the terms "alarm," "reminder," "alert," and "notification" for essentially the same idea. Sometimes differences in terminology may reflect subtle differences in usage, too. So it is your responsibility to sort out these differences and act to help standardize the terminology for consistency issues in the requirements statements.

10.4.2 Take User Stories and Requirements Back to Customers and Users for Validation

Abstraction

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

Your structured sets of user stories and requirements offer you, your client, and your users a lens through which you can contemplate and discuss groups of related user stories and requirements at various levels of abstraction. This is an important step for you both because it gives you a chance to get inputs and to correct misconceptions before you get into design. It also helps solidify your relationship as partners in the process.

For each work role, schedule a meeting with interested client representatives and representative users, preferably some from the ones you have interviewed or otherwise interacted with before. Walk them through the requirements to make sure your interpretation of user stories and requirements is accurate and relatively complete.

10.4.3 Resolve Organizational, Social, and Personal Issues Arising Out of Work Practice Changes

When you take your user stories and requirements to the customer for validation, it is also a good opportunity to resolve organizational, social, and personal issues that might arise out of work practice changes in the new design. Because your requirements reflect what you intend to put into the design, if heeded, they can flash early warning signs to customers and users about issues of which your team may be unaware, even after thorough usage data elicitation. Especially if your requirements are pointing toward a design that changes the work environment, the way work is done, or the job descriptions of workers, your requirements may give rise to issues of territoriality, fear, and control.

Changes in the workflow may challenge established responsibilities and authorities. There may also be legal requirements or platform constraints for doing things in a certain way, a way you cannot change regardless of arguments for efficiency or a better user experience. Organizational, social, and personal

issues can catch your team by surprise because they may well be thinking mostly about technical aspects and UX design at this point.

For a discussion of how user stories and requirements are used to drive design, see Chapter 14 on design creation. See also Chapter 29 (on UX + SE) for how user stories are the focus of how the UX designers and the SE team communicate to synchronize design and implementation in agile sprints.