3

Features, Scenarios, and Stories

Some software products are inspired. The developers of these products have a vision of the software that they want to create. They don't have a product manager, they don't do user surveys, they don't collect and document requirements or model how users will interact with the system. They simply get on with developing a prototype system. Some of the most successful software products, such as Facebook, started like this.

However, the vast majority of software products that are solely based on a developer's inspiration are commercial failures. These products either don't meet a real user need or don't fit with the ways in which users really work. Inspiration is important but most successful products are based on an understanding of business and user problems and user interaction. Even when inspiration leads to many users adopting a product, continuing use depends on its developers understanding how the software is used and new features that users may want.

Apart from inspiration, there are three factors that drive the design of software products:

1. *Business and consumer needs that are not met by current products* For example, book and magazine publishers are moving to both online and paper publication, yet few software products allow seamless conversion from one medium to another.
2. *Dissatisfaction with existing business or consumer software products* For example, many current software products are bloated with features that are rarely used. New companies may decide to produce simpler products in the same area that meet the needs of most users.

3. *Changes in technology that make completely new types of products possible* For example, as virtual reality (VR) technology matures and the hardware gets cheaper, new products may exploit this opportunity.

As I explained in Chapter 1, software products are not developed to meet the requirements of a specific client. Consequently, techniques that have been developed for eliciting, documenting, and managing software requirements aren't used for product engineering. You don't need to have a complete and detailed requirements document as part of a software development contract. There is no need for prolonged consultations when requirements change. Product development is incremental and agile, so you can use less formal ways of defining your product.

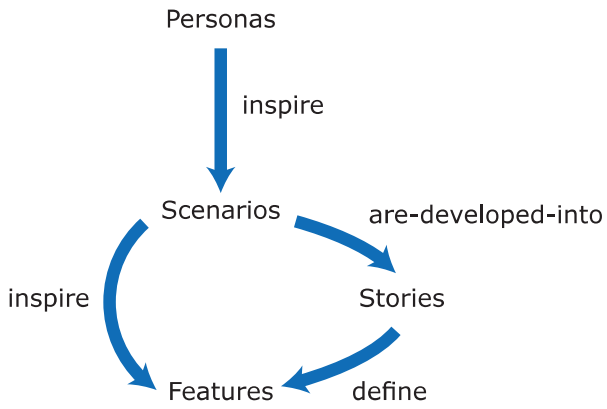
In the early stage of product development, rather than understanding the requirements of a specific client, you are trying to understand what product features will be useful to users and what they like and dislike about the products that they use. Briefly, a feature is a fragment of functionality, such as a Print feature, a Change Background feature, a New Document feature, and so on. Before you start programming a product, you should create a list of features to be included in your product. This is your starting point for product design and development.

It makes sense in any product development to spend time trying to understand the potential users and customers of your product. A range of techniques have been developed for understanding the ways that people work and use software. These include user interviews, surveys, ethnography, and task analysis.[1] Some of these techniques are expensive and unrealistic for small companies. However, informal user analysis and discussions, which simply involve asking users about their work, the software that they use, and its strengths and weaknesses, are inexpensive and very valuable.

One problem with informal user studies for business products is that the users simply may not want new software. For business products, the business buys the product, but its employees are the users. These users may be hostile to new products because they have to change their familiar way of working or perhaps because increased automation may reduce the number of jobs available. Business managers may suggest what they want from a new software product, but this does not always reflect the needs or wishes of the product's users.

[1] I discuss techniques of user analysis to discover software requirements in my general software engineering textbook, *Software Engineering*, 10th edition (Pearson Education, 2015).

Figure 3.1 From personas to features



In this chapter, I assume that informal user consultations are possible. I explain ways of representing users (personas) and communicating with them and other product stakeholders. I focus on how short, natural language descriptions (scenarios and stories) can be used to visualize and document how users might interact with a software product.

Figure 3.1 shows that personas, scenarios, and user stories lead to features that might be implemented in a software product.

If you look on the web, you can find a range of definitions of a "product feature," but I think of a feature as a fragment of functionality that implements some user or system need. You access features through the user interface of a product. For example, the editor that I used to write this book includes a feature to create a "New Group," in which a group is a set of documents that is accessed as a pull-down menu.

A feature is something that the user needs or wants. You can write a user story to make this explicit:

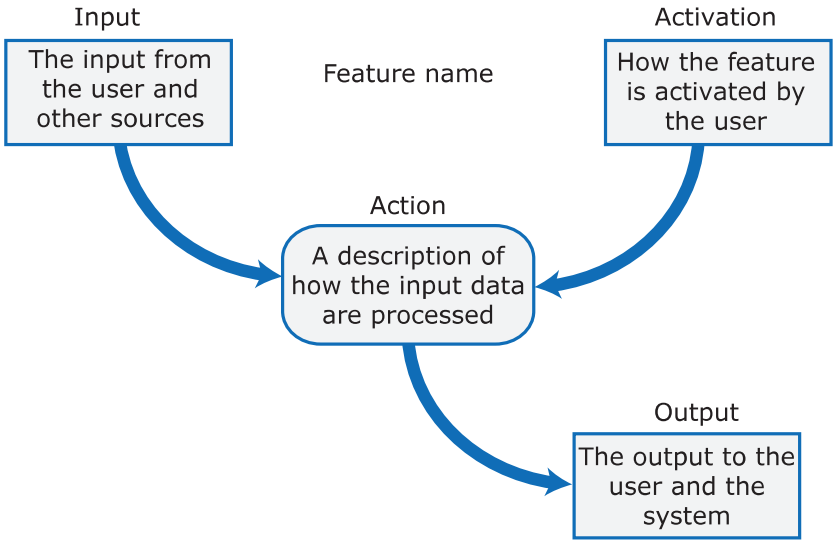
> *As an author I need a way to organize the text of the book that I'm writing into chapters and sections.*

Using the New Group feature, you create a group for each chapter, and the documents within that group are the sections of that chapter. This feature can be described using a short, narrative, feature description:

> *The "New Group" command, activated by a menu choice or keyboard shortcut, creates a named container for a set of documents and groups.*

Alternatively, you can use a standard template where you define the feature by its input, its functionality, its output, and how it is activated. Figure 3.2

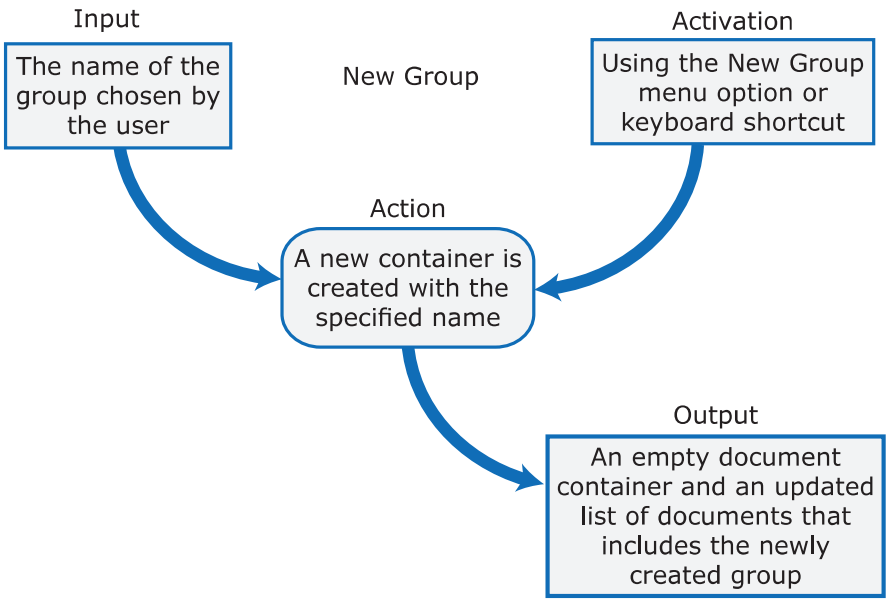
Figure 3.2 Feature description



shows the elements of the standard template, and Figure 3.3 shows how the New Group feature can be defined using this template.

Features can be defined using the input/action/output model that I have shown. However, system developers often need only a short feature description; then they fill in the feature details. This is especially true when the

Figure 3.3 The New Group feature description



feature is a "utility" feature that is commonly available in other products. For example, the Cut and Paste feature is well known, so might be defined as:

Cut and Paste – any selected object can be cut or copied, then inserted elsewhere into the document.

Sometimes all you need to say is that a Cut and Paste feature should be included and then rely on the developer's general understanding to implement this.

Features are the fundamental elements of an agile method called Feature-Driven Development (FDD). I have no experience with this method and have not met anyone who uses it for product development, so I can't comment on it directly. One aspect of this approach that I have used, however, is its template for feature description:

<*action*> **the** <*result*> **<by|for|of|to>** <*object*>

So, the New Group feature above could be described as:

Create **the** *container* **for** *documents or groups*

I show another example of this simple approach to feature description in Section 3.4.2.

I return to the topic of features in Section 3.4 after I have described scenarios and user stories. You can use both of these narrative techniques for deriving the list of features to be included in a system.

3.1 Personas

One of the first questions you should ask when developing a software product is "Who are the target users for my product?" You need to have some understanding of your potential users in order to design features that they are likely to find useful and to design a user interface that is suited to them.

Sometimes you know the users. If you are a software engineer developing software tools for other engineers, then you understand, to some extent at least, what they are likely to want. If you are designing a phone or tablet app for general use, you may talk to friends and family to understand what potential users may like or dislike. In those circumstances, you may be able to design using an

Table 3.1 A persona for a primary school teacher

Jack, a primary school teacher
Jack, age 32, is a primary school (elementary school) teacher in Ullapool, a large coastal village in the Scottish Highlands. He teaches children from ages 9 to 12. He was born in a fishing community north of Ullapool, where his father runs a marine fuels supply business and his mother is a community nurse. He has a degree in English from Glasgow University and retrained as a teacher after several years working as a web content author for a large leisure group. Jack's experience as a web developer means that he is confident in all aspects of digital technology. He passionately believes that the effective use of digital technologies, blended with face-to-face teaching, can enhance the learning experience for children. He is particularly interested in using the iLearn system for project-based teaching, where students work together across subject areas on a challenging topic.

intuitive idea of who the users are and what they can do. However, you should be aware that your users are diverse and your own experience may not provide a complete picture of what they might want and how they might work.

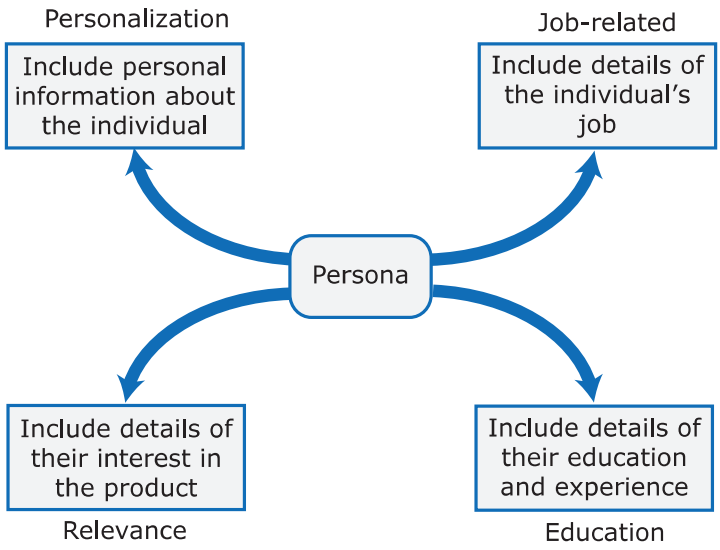
For some types of software products, you may not know much about the background, skills, and experience of potential users. Individuals on the development team may have their own ideas about the product users and their capabilities. This can lead to product inconsistencies as these different views are reflected in the software implementation. Ideally, the team should have a shared vision of users, their skills, and their motivations for using the software. Personas are one way of representing this shared vision.

Personas are about "imagined users," character portraits of types of user that you think might adopt your product. For example, if your product is aimed at managing appointments for dentists, you might create a dentist persona, a receptionist persona, and a patient persona. Personas of different types of users help you imagine what these users may want to do with your software and how they might use it. They also help you envisage difficulties that users might have in understanding and using product features.

A persona should paint a picture of a type of product user. You should describe the users' backgrounds and why they might want to use your product. You should also say something about their education and technical skills. This helps you assess whether or not a software feature is likely to be useful and understandable by typical product users.

An example of a persona that I wrote when designing the iLearn system, described in Chapter 1, is shown in Table 3.1. This is the persona of a teacher who is committed to digital learning and who believes that using digital devices can enhance the overall learning process.

Figure 3.4 Persona descriptions



There is no standard way of representing a persona; if you search the web, you will find a number of different recommendations. Common features of these suggestions are shown in Figure 3.4. The recommended aspects of a persona description—namely, personalization, relevance, education, and job-related—are explained in Table 3.2.

Many recommendations about what should be included in a persona suggest describing the individual's goals. I disagree because it's impossible to pin down what is meant by "goals." Goals are a broad concept. Some people have goals of self-improvement and learning; others have work-oriented goals of career progression and promotion. For some people, goals are not related to work. Their goals are simply to get through the day and earn enough so that they can do the things outside of work that give them pleasure and satisfaction.

My experience is that trying to define "user goals" is not helpful. I have found that most people don't have clearly defined goals when they use software. They may have been told to use the software as part of their job, they may see the software as a way to do their work more effectively, or they may find the software useful in organizing their life. Rather than try to set out goals, I think it's better to try to explain why the software might be useful and to give examples of what potential users may want to do with it.

If your product is targeted at a specific group of users, you may need only one or two personas to represent potential system users. For some products, however, the user group may be very broad and you may think that a large number of personas are needed. In fact, using many personas can make it more difficult to design a coherent system because they inevitably overlap.

Table 3.2 Aspects of persona descriptions

Aspect	Description
Personalization	You should give them a name and say something about their personal circumstances. It is sometimes helpful to use an appropriate stock photograph to represent the person in the persona. Some studies suggest that this helps project teams use personas more effectively.
Job-related	If your product is targeted at business, you should say something about their job and (if necessary) what that job involves. For some jobs, such as a teacher where readers are likely to be familiar with the job, this may not be necessary.
Education	You should describe their educational background and their level of technical skills and experience. This is important, especially for interface design.
Relevance	If you can, you should say why they might be interested in using the product and what they might want to do with it.

In general, you don't need more than five personas to help identify the key features of a system.

Personas should be relatively short and easy to read. For the iLearn system, we developed personas that we described in two or three paragraphs of text. We found that these had enough information to be useful. Two of the personas that we created are shown in Tables 3.3 and 3.4.

The persona in Table 3.3 represents users who do not have a technical background. They simply want a system to provide support for administration.

Table 3.3 A persona for a history teacher

Emma, a history teacher
Emma, age 41, is a history teacher in a secondary school (high school) in Edinburgh. She teaches students from ages 12 to 18. She was born in Cardiff in Wales, where both her father and her mother were teachers. After completing a degree in history from Newcastle University, she moved to Edinburgh to be with her partner and trained as a teacher. She has two children, aged 6 and 8, who both attend the local primary school. She likes to get home as early as she can to see her children, so often does lesson preparation, administration, and marking from home. Emma uses social media and the usual productivity applications to prepare her lessons, but is not particularly interested in digital technologies. She hates the virtual learning environment that is currently used in her school and avoids using it if she can. She believes that face-to-face teaching is most effective. She might use the iLearn system for administration and access to historical films and documents. However, she is not interested in a blended digital/face-to-face approach to teaching.

Table 3.4 A persona for an IT technician

Elena, a school IT technician
Elena, age 28, is a senior IT technician in a large secondary school (high school) in Glasgow with over 2000 students. Originally from Poland, she has a diploma in electronics from Potsdam University. She moved to Scotland in 2011 after being unemployed for a year after graduation. She has a Scottish partner, no children, and hopes to develop her career in Scotland. She was originally appointed as a junior technician but was promoted, in 2014, to a senior post responsible for all the school computers. Although not involved directly in teaching, Elena is often called on to help in computer science classes. She is a competent Python programmer and is a "power user" of digital technologies. She has a long-term career goal of becoming a technical expert in digital learning technologies and being involved in their development. She wants to become an expert in the iLearn system and sees it as an experimental platform for supporting new uses for digital learning.

Elena's persona in Table 3.4 represents technically skilled support staff who may be responsible for setting up and configuring the iLearn software.

I haven't included personas for the students who were intended to be the ultimate end-users of the iLearn system. The reason is that we saw the iLearn system as a platform product that should be configured to suit the preferences and needs of individual schools and teachers. Students would use iLearn to access tools. However, they would not use the configuration features that make iLearn a unique system. Although we planned to include a standard set of applications with the system, we were driven by the belief that the best people to create learning systems were teachers, supported by local technical staff.

Ideally, software development teams should be diverse, with people of different ages and genders. However, the reality is that software product developers are still, overwhelmingly, young men with a high level of technical skill. Software users are more diverse, with varying levels of technical skill. Some developers find it hard to appreciate the problems that users may have with the software. An important benefit of personas is that they help the development team members empathize with potential users of the software. Personas are a tool that allows team members to "step into the users' shoes." Instead of thinking about what they would do in a particular situation, they can imagine how a persona would behave and react.

So, when you have an idea for a feature, you can ask "Would this persona be interested in this feature?" and "How would that persona access and use the feature?". Personas can help you check your ideas to make sure that you are not including product features that aren't really needed. They help you to avoid making unwarranted assumptions, based on your own knowledge, and designing an overcomplicated or irrelevant product.

Personas should be based on an understanding of the potential product users: their jobs, their backgrounds, and their aspirations. You should study and survey potential users to understand what they want and how they might use the product. From these data, you can abstract the essential information about the different types of product users and then use this as a basis for creating personas. These personas should then be cross-checked against the user data to make sure that they reflect typical product users.

It may be possible to study the users when your product is a development of an existing product. For new products, however, you may find it impractical to carry out detailed user surveys. You may not have easy access to potential users. You may not have the resources to carry out user surveys, and you may want to keep your product confidential until it is ready to launch.

If you know nothing about an area, you can't develop reliable personas, so you need to do some user research before you start. This does not have to be a formal or prolonged process. You may know people working in an area and they might be able to help you meet their colleagues to discuss your ideas. For example, my daughter is a teacher and she helped arrange lunch with a group of her colleagues to discuss how they use digital technologies. This helped with both persona and scenario development.

Personas that are developed on the basis of limited user information are called proto-personas. Proto-personas may be created as a collective team exercise using whatever information is available about potential product users. They can never be as accurate as personas developed from detailed user studies, but they are better than nothing. They represent the product users as seen by the development team, and they allow the developers to build a common understanding of the potential product users.

3.2 Scenarios

As a product developer, your aim should be to discover the product features that will tempt users to adopt your product rather than competing software. There is no easy way to define the "best" set of product features. You have to use your own judgement about what to include in your product. To help select and design features, I recommend that you invent scenarios to imagine how users could interact with the product that you are designing.

A scenario is a narrative that describes a situation in which a user is using your product's features to do something that they want to do. The scenario should briefly explain the user's problem and present an imagined way that

Table 3.5 Jack's scenario: Using the iLearn system for class projects

Fishing in Ullapool
Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development, and economic impact of fishing. As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives, and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history archive site) to access newspaper archives and photographs. However, Jack also needs a photo-sharing site as he wants students to take and comment on each others' photos and to upload scans of old photographs that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues. Jack sends an email to a primary school teachers' group to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he use KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account with KidsTakePics. He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that, when they log in, they can immediately use the system to upload photos from their phones and class computers.

the problem might be solved. There is no need to include everything in the scenario; it isn't a detailed system specification.

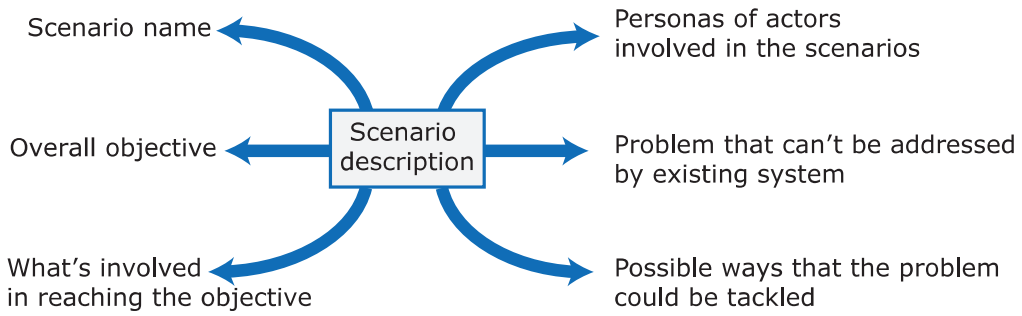
Table 3.5 is an example scenario that shows how Jack, whose persona I described in Section 3.2, might use the iLearn system.

From this description of how iLearn might be used for class projects, you can see some of the key elements of a scenario (Figure 3.5) that may be included to help you think about the product features that you need.

The most important elements of a scenario are:

1. A brief statement of the overall objective. In Jack's scenario, shown in Table 3.5, this is to support a class project on the fishing industry.
2. References to the persona involved (Jack) so that you can get information about the capabilities and motivation of that user.
3. Information about what is involved in doing the activity. For example, in Jack's scenario, this involves gathering reminiscences from relatives, accessing newspaper archives, and so on.
4. If appropriate, an explanation of problems that can't be readily addressed using the existing system. Young children don't understand

Figure 3.5 Elements of a scenario description



issues such as copyright and privacy, so photo sharing requires a site that a teacher can moderate to make sure that published images are legal and acceptable.

5. A description of one way that the identified problem might be addressed. This may not always be included especially if technical knowledge is needed to solve the problem. In Jack's scenario, the preferred approach is to use an external tool designed for school students.

The idea of using scenarios to support software engineering has been around since the 1980s. Different types of scenarios have been proposed, ranging from high-level scenarios like this one about Jack to more detailed and specific scenarios that set out the steps involved in a user's interaction with a system. They are the basis for both use cases, which are extensively used in object-oriented methods, and user stories, which are used in agile methods. Scenarios are used in the design of requirements and system features, in system testing, and in user interface design.

Narrative, high-level scenarios, like Jack's scenario, are primarily a means of facilitating communication and stimulating design creativity. They are effective in communication because they are understandable and accessible to users and to people responsible for funding and buying the system.[2] Like personas, they help developers to gain a shared understanding of the system that they are creating. You should always be aware, however, that scenarios are not specifications. They lack detail, they may be incomplete, and they may not represent all types of user interactions.

[2]I presented some of the scenarios for the iLearn system to a government minister for education. He commented that this was the first time he had ever attended a meeting on an IT system where he actually understood what the system was supposed to do.

Some people recommend that scenarios should be structured with different fields such as what the user sees at the beginning of a scenario, a description of the normal flow of events, a description of what might go wrong, and so on. If you are using scenarios to elicit detailed requirements, the benefit of structure is that you have a consistent presentation, which means you are less likely to forget to include elements that relate to critical system requirements. Engineers usually like this structured approach. However, my experience is that system users, who read and check the scenarios, find structured scenarios to be intimidating and hard to understand.

Consequently, when you are using scenarios at the early stages of product design, I recommend narrative rather than structured scenarios. These scenarios may range from two or three paragraphs of text, like Jack's scenario in Table 3.5, to longer descriptions. You may need to write longer descriptions when your software will be used in existing processes and must interoperate with other software. Your scenarios may then have to include descriptions of interactions with other processes and software systems.

Emma's scenario (shown in Table 3.6) is an example of a longer scenario. In it, she uses the iLearn system to help her with the administration involved in setting up a class trip.

Emma's scenario is different from Jack's scenario because it describes a common and well-understood process rather than something new. The scenario discusses how parts of the process (setting up an email group and web page) are automated by the iLearn system. Remember that Emma is an e-learning skeptic; she is not interested in innovative applications. She wants a system that will make her life easier and reduce the amount of routine administration that she has to do.

In this type of scenario, you are much more likely to include specific details of what might be in the system. For example, it explains that Emma logs in to the system with her Google credentials. This means she doesn't have to remember a separate login name and password. I discuss this approach to authentication in Chapter 7.

When you see this kind of information in a scenario, you need to check whether this is what the user really needs or whether it represents a more general requirement. The statement that the software has to support login using Google credentials might actually reflect a more general need—to provide a login mechanism in which users don't have to remember yet another set of credentials. You may decide on an alternative approach to authentication, such as fingerprint or face recognition on a mobile phone, that avoids the need for system-specific login credentials.

Table 3.6 Using the iLearn system for administration

Emma is teaching the history of World War I to a class of 14-year-olds (S3). A group of S3 students are visiting the historic World War I battlefields in northern France. She wants to set up a "battlefields group" where the students who are attending the trip can share their research about the places they are visiting as well as their pictures and thoughts about the visit. From home, she logs onto the iLearn system using her Google account credentials. Emma has two iLearn accounts—her teacher account and a parent account associated with the local primary school. The system recognizes that she is a multiple account owner and asks her to select the account to be used. She chooses the teacher account and the system generates her personal welcome screen. As well as her selected applications, this also shows management apps that help teachers create and manage student groups. Emma selects the "group management" app, which recognizes her role and school from her identity information and creates a new group. The system prompts for the class year (S3) and subject (history) and automatically populates the new group with all S3 students who are studying history. She selects those students going on the trip and adds her teacher colleagues, Jamie and Claire, to the group. She names the group and confirms that it should be created. The app sets up an icon on her iLearn screen to represent the group, creates an email alias for the group, and asks Emma if she wishes to share the group. She shares access with everyone in the group, which means that they also see the icon on their screen. To avoid getting too many emails from students, she restricts sharing of the email alias to Jamie and Claire. The group management app then asks Emma if she wishes to set up a group web page, wiki, and blog. Emma confirms that a web page should be created and she types some text to be included on that page. She then accesses Flickr using the icon on her screen, logs in, and creates a private group to share trip photos that students and teachers have taken. She uploads some of her own photos from previous trips and emails an invitation to join the photo-sharing group to the battlefields email list. Emma uploads material from her own laptop that she has written about the trip to iLearn and shares this with the battlefields group. This action adds her documents to the web page and generates an alert to group members that new material is available.

3.2.1 Writing scenarios

Your starting point for scenario writing should be the personas you have created. You should normally try to imagine several scenarios for each persona. Remember that these scenarios are intended to stimulate thinking rather than provide a complete description of the system. You don't need to cover everything you think users might do with your product.

Scenarios should always be written from the user's perspective and should be based on identified personas or real users. Some writers suggest that scenarios should focus on goals—what the user wants to do—rather than the mechanisms they use to do this. They argue that scenarios should not include

specific details of an interaction as these limit the freedom of feature designers. I disagree. As you can see from Emma's scenario, it sometimes makes sense to talk about mechanisms, such as login with Google, that both product users and developers understand.

Furthermore, writing scenarios in a general way that doesn't make assumptions about implementation can be potentially confusing for both users and developers. For example, I think that "X cuts paragraphs from the newspaper archive and pastes them into the project wiki" is easier to read and understand than "X uses an information transfer mechanism to move paragraphs from the newspaper archive to the project wiki."

Sometimes there may be a specific requirement to include a particular feature in the system because that feature is widely used. For example, Jack's scenario in Table 3.5 discusses the use of an iLearn wiki. Many teachers currently use wikis to support group writing and they specifically want to have wikis in the new system. Such user requirements might be even more specific. For example, when designing the iLearn system, we discovered that teachers wanted Wordpress blogs, not just a general blogging facility. So, you should include specific details in a scenario when they reflect reality.

Scenario writing is not a systematic process and different teams approach it in different ways. I recommend that each team member be given individual responsibility for creating a small number of scenarios and work individually to do this. Obviously, members may discuss the scenarios with users and other experts, but this is not essential. The team then discusses the proposed scenarios. Each scenario is refined based on that discussion.

Because it is easy for anyone to read and understand scenarios, it is possible to get users involved in their development. For the iLearn system, we found that the best approach was to develop an imaginary scenario based on our understanding of how the system might be used and then ask users to tell us what we got wrong. Users could ask about things they did not understand, such as "Why can't a photo-sharing site like Flickr be used in Jack's scenario?" They could suggest how the scenario could be extended and made more realistic.

We tried an experiment in which we asked a group of users to write their own scenarios about how they might use the system. This was not a success. The scenarios they created were simply based on how they worked at the moment. They were far too detailed and the writers couldn't easily generalize their experience. Their scenarios were not useful because we wanted something to help us generate ideas rather than replicate the systems that they already used.

Scenarios are not software specifications but are ways of helping people think about what a software system should do. There is no simple answer

Table 3.7 Elena's scenario: Configuring the iLearn system

Elena has been asked by David, the head of the art department in her school, to help set up an iLearn environment for his department. David wants an environment that includes tools for making and sharing art, access to external websites to study artworks, and "exhibition" facilities so that the students' work can be displayed. Elena starts by talking to art teachers to discover the tools that they recommend and the art sites that they use for studies. She also discovers that the tools they use and the sites they access vary according to the age of their students. Consequently, different student groups should be presented with a toolset that is appropriate for their age and experience. Once she has established what is required, Elena logs into the iLearn system as an administrator and starts configuring the art environment using the iLearn setup service. She creates sub-environments for three age groups plus a shared environment that includes tools and sites that may be used by all students. She drags and drops tools that are available locally and the URLs of external websites into each of these environments. For each of the sub-environments, she assigns an art teacher as its administrator so that they can't refine the tool and website selection that has been set up. She publishes the environments in "review mode" and makes them available to the teachers in the art department. After discussing the environments with the teachers, Elena shows them how to refine and extend the environments. Once they have agreed that the art environment is useful, it is released to all students in the school.

to the question "How many scenarios do I need?" In the iLearn system, 22 scenarios were developed to cover different aspects of system use. There was quite a lot of overlap among these scenarios, so it was probably more than we really needed. Generally, I recommend developing three or four scenarios per persona to get a useful set of information.

Although scenarios certainly don't have to describe every possible use of a system, it is important that you look at the roles of each of the personas that you have developed and write scenarios that cover the main responsibilities for that role. Jack's scenario and Emma's scenario are based on using the iLearn system to support teaching.

Like other system products designed for use in an organization, however, iLearn needs to be configured for use. While some of this configuration can be done by tech-savvy teachers, in many schools it is technical support staff who have this responsibility. Elena's scenario, shown in Table 3.7, describes how she might configure the iLearn software.

Writing scenarios always gives you ideas for the features that you can include in the system. You may then develop these ideas in more detail by analyzing the text of the scenario, as I explain in the next section.

3.3 User stories

I explained in Section 3.2 that scenarios are descriptions of situations in which a user is trying to do something with a software system. Scenarios are high-level stories of system use. They should describe a sequence of interactions with the system but should not include details of these interactions.

User stories are finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system. I presented a user story at the beginning of the chapter:

> *As an author I need a way to organize the book that I'm writing into chapters and sections.*

This story reflects what has become the standard format of a user story:

> *As a <role>, I <want / need> to <do something>*

Another example of a user story taken from Emma's scenario might be:

> *As a teacher, I want to tell all members of my group when new information is available.*

A variant of this standard format adds a justification for the action:

> *As a <role> I <want / need> to <do something> so that <reason>*

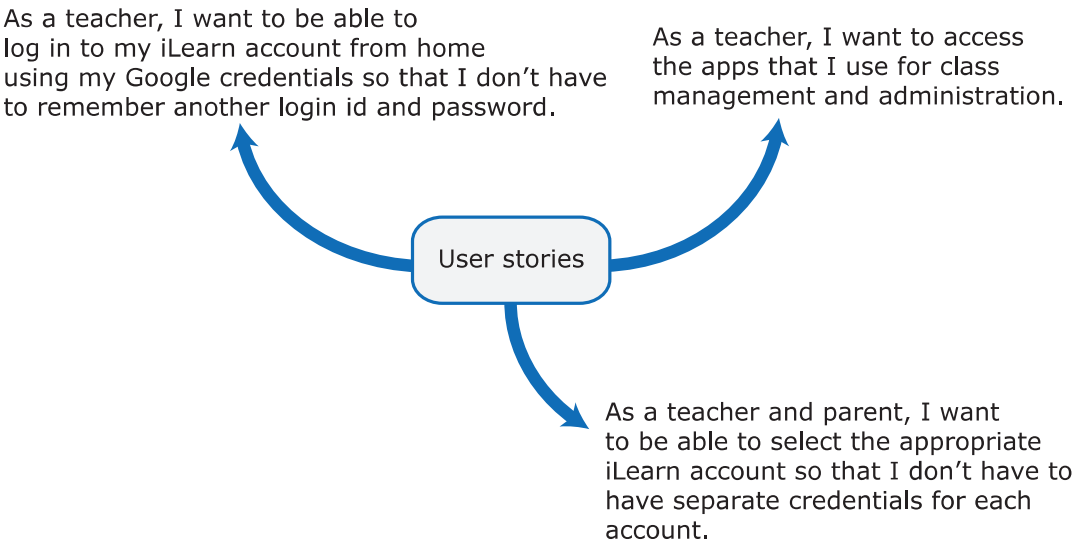
For example:

> *As a teacher, I need to be able to report who is attending a class trip so that the school maintains the required health and safety records.*

Some people argue that a rationale or justification should always be part of a user story. I think this is unnecessary if the story makes sense on its own. Knowing one reason why this might be useful doesn't help the product developers. However, in situations where some developers are unfamiliar with what users do, a rationale can help those developers understand why the story has been included. A rationale may also help trigger ideas about alternative ways of providing what the user wants.

An important use of user stories is in planning, and many users of the Scrum method represent the product backlog as a set of user stories. For this purpose, user stories should focus on a clearly defined system feature or

Figure 3.6 User stories from Emma's scenario



aspect of a feature that can be implemented within a single sprint. If the story is about a more complex feature that might take several sprints to implement, then it is called an "epic." An example of an epic might be:

> *As a system manager, I need a way to back up the system and restore individual applications, files, directories, or the whole system.*

A lot of functionality is associated with this user story. For implementation, it should be broken down into simpler stories, with each story focusing on a single aspect of the backup system.

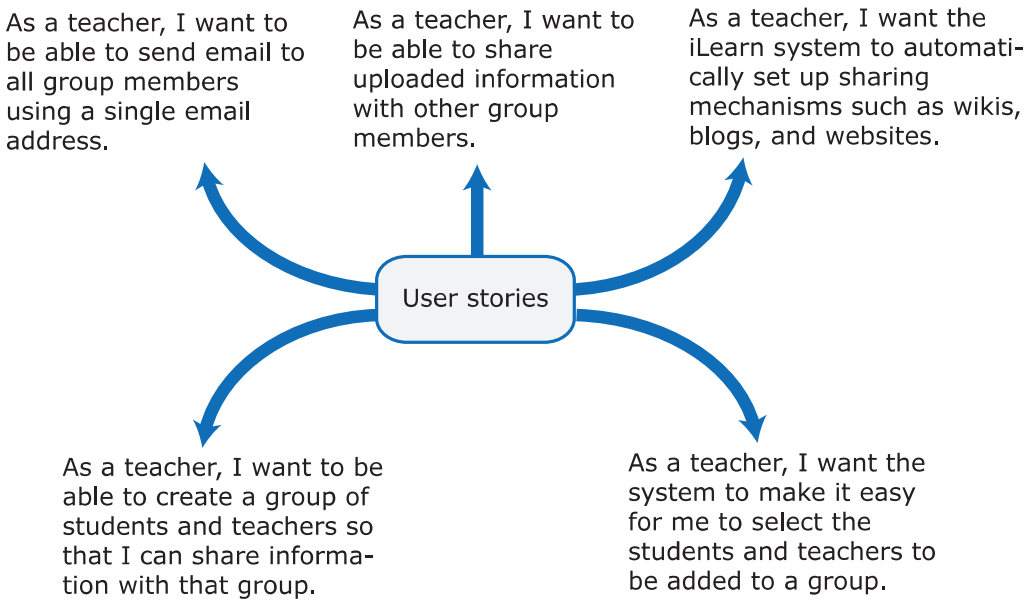
When you are thinking about product features, user stories are not intended for planning but for helping with feature identification. Therefore, you don't need to be overly concerned about whether your stories are simple stories or epics. You should aim to develop stories that are helpful in one of two ways:

- as a way of extending and adding detail to a scenario;
- as part of the description of the system feature that you have identified.

As an example of scenario refinement, the initial actions in Emma's scenario shown in Figure 3.6 can be represented by three user stories. Recall that the scenario says:

> From home, she logs onto the iLearn system using her Google account credentials. Emma has two iLearn accounts—her teacher account and

Figure 3.7 User stories describing the Groups feature



a parent account associated with the local primary school. The system recognizes that she is a multiple account owner and asks her to select the account to be used. She chooses the teacher account and the system generates her personal welcome screen. As well as her selected applications, this also shows management apps that help teachers create and manage student groups.

You can create user stories from this account, as I show in Figure 3.6.

You can see from the stories in Figure 3.6 that I have included a rationale in the story that explains why Emma wants to work in the way specified in the scenario. It is best not to include rationale in a scenario, as it tends to disrupt the flow of the description and make it more difficult to read and understand.

When you define user stories from a scenario, you provide more information to developers to help them design the product's features. We can see an example of this in the stories shown in Figure 3.6. Emma wants an easy way to authenticate herself to the system, either as a teacher or as a parent. She doesn't want to have to remember more login credentials, and she doesn't want to have two accounts with different credentials.

As an example of how stories can be used to describe the features of a system, Emma's scenario discusses how a group can be created and explains system actions that take place on group creation. The user stories shown in Figure 3.7 can be derived from the scenario to describe the Groups feature in the iLearn system.

The set of stories shown in Figure 3.7 is not a complete description of the Groups feature. No stories are concerned with deleting or changing a group, restricting access, and other tasks. You start by deriving stories from a scenario, but you then have to think about what other stories might be needed for a complete description of a feature's functionality.

A question that is sometimes asked about user stories is whether you should write "negative stories" that describe what a user doesn't want. For example, you might write this negative story:

> *As a user, I don't want the system to log and transmit my information to any external servers.*

If you are writing stories to be part of a product backlog, you should avoid negative stories. It is impossible to write system tests that demonstrate a negative. In the early stages of product design, however, it may be helpful to write negative stories if they define an absolute constraint on the system. Alternatively, you can sometimes reframe negative stories in a positive way. For example, instead of the above story, you could write:

> *As a user, I want to be able to control the information that is logged and transmitted by the system to external servers so that I can ensure that my personal information is not shared.*

Some of the user stories that you develop will be sufficiently detailed that you can use them directly in planning. You can include them in a product backlog. Sometimes, however, to use stories in planning, you have to refine the stories to relate them more directly to the implementation of the system.

It is possible to express all of the functionality described in a scenario as user stories. So, an obvious question that you might ask is "Why not just develop user stories and forget about scenarios?" Some agile methods rely exclusively on user stories, but I think that scenarios are more natural and are helpful for the following reasons:

1. Scenarios read more naturally because they describe what a user of a system is actually doing with that system. People often find it easier to relate to this specific information rather than to the statement of wants or needs set out in a set of user stories.
2. When you are interviewing real users or checking a scenario with real users, they don't talk in the stylized way that is used in user stories. People relate better to the more natural narrative in scenarios.

3. Scenarios often provide more context—information about what users are trying to do and their normal ways of working. You can do this in user stories, but it means that they are no longer simple statements about the use of a system feature.

Scenarios and stories are helpful in both choosing and designing system features. However, you should think of scenarios and user stories as "tools for thinking" about a system rather than a system specification. Scenarios and stories that are used to stimulate thinking don't have to be complete or consistent, and there are no rules about how many of each you need.

3.4 Feature identification

As I said in the chapter introduction, your aim at this early stage of product design is to create a list of features that define your software product. A feature is a way of allowing users to access and use your product's functionality so that the feature list defines the overall functionality of the system. In this section, I explain how scenarios and stories can be used to help identify product features.

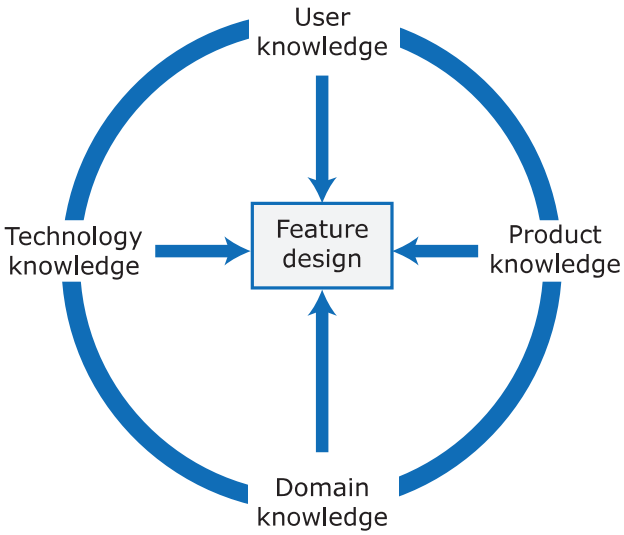
You should, ideally, identify product features that are independent, coherent and relevant:

1. *Independence* A feature should not depend on how other system features are implemented and should not be affected by the order of activation of other features.
2. *Coherence* Features should be linked to a single item of functionality. They should not do more than one thing, and they should never have side effects.
3. *Relevance* System features should reflect the way users normally carry out some task. They should not offer obscure functionality that is rarely required.

There is no definitive method for feature selection and design. Rather, the four important knowledge sources shown in Figure 3.8 can help with this.

Table 3.8 explains these knowledge sources in more detail. Of course, these are not all of equal importance for all products. For example, domain knowledge is very important for business products but less important for generic consumer products. You therefore need to think carefully about the knowledge required for your specific product.

Figure 3.8 Feature design

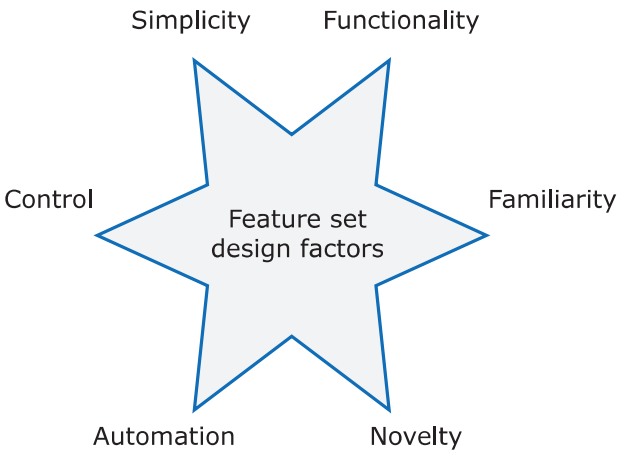


Innovation often stems from a combination of domain and technology knowledge. A good example is the Git system for code management that I cover in Chapter 10. Git works in a completely different way from previous code management systems. These older systems were based on a technology model in which storage was expensive, so they focused on limiting storage use and delivering information to users as required. The developer of Git realized

Table 3.8 Knowledge required for feature design

Knowledge	Description
User knowledge	You can use user scenarios and user stories to inform the team of what users want and how they might use the software features.
Product knowledge	You may have experience of existing products or decide to research what these products do as part of your development process. Sometimes your features have to replicate existing features in these products because they provide fundamental functionality that is always required.
Domain knowledge	This is knowledge of the domain or work area (e.g., finance, event booking) that your product aims to support. By understanding the domain, you can think of new innovative ways of helping users do what they want to do.
Technology knowledge	New products often emerge to take advantage of technological developments since their competitors were launched. If you understand the latest technology, you can design features to make use of it.

Figure 3.9 Factors in feature set design



that storage had become much cheaper, so it was possible for all users to have a complete copy of all information. This allowed for a new approach that dramatically simplified software development by distributed teams.

When you are designing a product feature set and deciding how features should work, you have to consider the six factors shown in Figure 3.9.

Unfortunately, it is impossible to design a feature set in which all of these factors are optimized, so you have to make some trade-offs:

1. *Simplicity and functionality* Everyone says they want software to be as simple as possible to use. At the same time, they demand functionality that helps them do what they want to do. You need to find a balance between providing a simple, easy-to-use system and including enough functionality to attract users with a variety of needs.

2. *Familiarity and novelty* Users prefer that new software should support the familiar everyday tasks that are part of their work or life. However, if you simply replicate the features of a product that they already use, there is no real motivation for them to change. To encourage users to adopt your system, you need to include new features that will convince users that your product can do more than its competitors.

3. *Automation and control* You may decide that your product can automatically do things for users that other products can't. However, users inevitably do things differently from one another. Some may like automation, where the software does things for them. Others prefer to have control. You therefore have to think carefully about what can be automated, how it is automated, and how users can configure the automation so that the system can be tailored to their preferences.

Your choices have a major influence on the features to be included in your product, how they integrate, and the functionality they provide. You may make a specific choice—for example, to focus on simplicity—that will drive the design of your product.

One problem that product developers should be aware of and try to avoid is "feature creep." Feature creep means that the number of features in a product creeps up as new potential uses of the product are envisaged.

The size and complexity of many large software products such as Microsoft Office and Adobe Photoshop are a consequence of feature creep. Most users use only a relatively small subset of the features of these products. Rather than stepping back and simplifying things, developers continually added new features to the software.

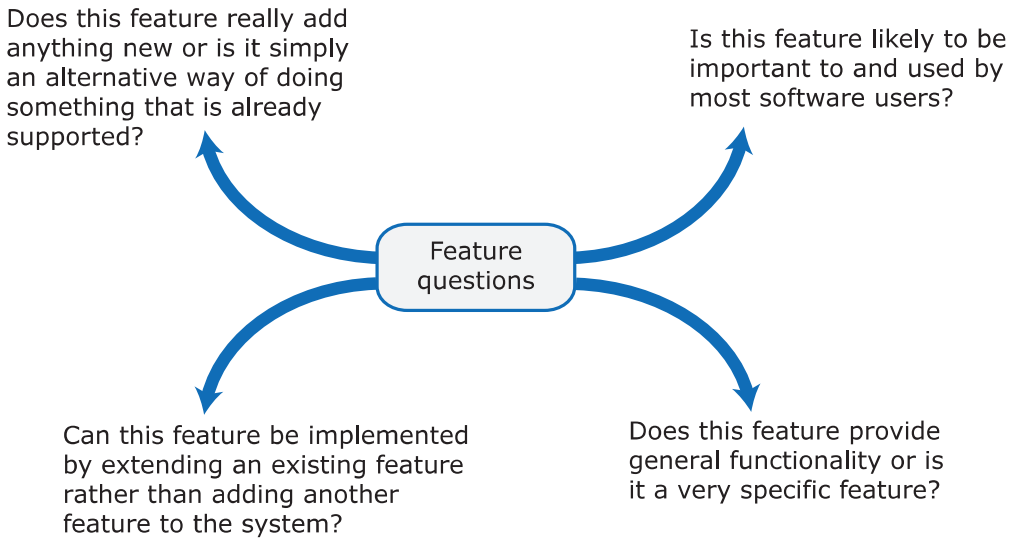
Feature creep adds to the complexity of a product, which means that you are likely to introduce bugs and security vulnerabilities into the software. It also usually makes the user interface more complex. A large feature set often means that you have to bundle vaguely related features together and provide access to these through a higher-level menu. This can be confusing, especially for less experienced users.

Feature creep happens for three reasons:

1. Product managers and marketing executives discuss the functionality they need with a range of different product users. Different users have slightly different needs or may do the same thing but in slightly different ways. There is a natural reluctance to say no to important users, so functionality to meet all of the users' demands ends up in the product.
2. Competitive products are introduced with slightly different functionality to your product. There is marketing pressure to include comparable functionality so that market share is not lost to these competitors. This can lead to "feature wars," where competing products become more and more bloated as they replicate the features of their competitors.
3. The product tries to support both experienced and inexperienced users. Easy ways of implementing common actions are added for inexperienced users and the more complex features to accomplish the same thing are retained because experienced users prefer to work that way.

To avoid feature creep, the product manager and the development team should review all feature proposals and compare new proposals to features that have already been accepted for implementation. The questions shown in Figure 3.10 may be used to help identify unnecessary features.

Figure 3.10 Avoiding feature creep



3.4.1 Feature derivation

When you start with a product vision or writing scenarios based on that vision, product features immediately come to mind. I discussed the iLearn system vision in Chapter 1 and I repeat it in Table 3.9.

I have highlighted phrases in this vision suggesting features that should be part of the product, including:

- a feature that allows users to access and use existing web-based resources;
- a feature that allows the system to exist in multiple different configurations;
- a feature that allows user configuration of the system to create a specific environment.

These features distinguish the iLearn system from existing VLEs and are the central features of the product.

Table 3.9 The iLearn system vision

FOR teachers and educators WHO need a way to *help students use web-based learning resources and applications,* THE iLearn system is an open learning environment THAT *allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves.*
UNLIKE Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process rather than the administration and management of materials, assessments, and coursework. OUR product *enables teachers to create subject and age-specific environments for their students* using any web-based resources, such as videos, simulations, and written materials that are appropriate

Table 3.10 Jack's scenario with highlighted phrases

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development, and economic impact of fishing.

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives, and collect old photographs related to fishing and fishing communities in the area. *Students use an iLearn wiki* to gather together fishing stories and *SCRAN (a history archive) to access newspaper archives and photographs.* However, Jack also needs a photo-sharing site as he wants *pupils to take and comment on each others' photos and to upload scans of old photographs* that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack *sends an email to a primary school teachers' group* to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he use KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics *is not integrated with the iLearn authentication service,* he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that when they log in, they can immediately use the system to upload photos from their phones and class computers.

This approach of highlighting phrases in a narrative description can be used when analyzing scenarios to find system features. You read through the scenarios, look for user actions (usually denoted by active verbs, such as "use," "choose," "send," "update," and so on), and highlight the phrases where these are mentioned. You then think about the product features that can support these actions and how they might be implemented.

In Table 3.10, I have done this with Jack's scenario (see Table 3.5), in which he sets up a system for his students' project work.

The highlighted text identifies features and feature characteristics that should be part of the iLearn system:

- a wiki for group writing;
- access to the SCRAN history archive, which is a shared national resource that provides access to historical newspaper and magazine articles for schools and universities;
- the ability to set up and access an email group;
- the ability to integrate some applications with the iLearn authentication service.

It also confirms the need for the configuration feature that has already been identified from the product vision.

Feature identification should be a team activity, and as features are identified, the team should discuss them and generate ideas about related features. Jack's scenario suggests that there is a need for groups to write together. You should therefore think about age-appropriate ways to design features for:

- collaborative writing, where several people can work simultaneously on the same document;
- blogs and web pages as a way of sharing information.

You can also think about generalizing the features suggested by the scenario. The scenario identifies the need for access to an external archive (SCRAN). However, perhaps the feature that you add to the software should support access to any external archive and allow students to transfer information to the iLearn system.

You can go through a similar process of highlighting phrases in all of the scenarios that you have developed and using them to identify and then generalize a set of product features. If you have developed user stories to refine your scenarios, these may immediately suggest a product feature or feature characteristic. For example, this story was derived from Emma's scenario (see Table 3.6):

> *As a teacher and a parent, I want to be able to select the appropriate iLearn account so that I don't have to have separate credentials for each account.*

This story states that the account feature of the iLearn system has to accommodate the idea that a single user may have multiple accounts. Each account is associated with a particular role the user may adopt when using the system. When logging in, users should be able to select the account they wish to use.

3.4.2 The feature list

The output of the feature identification process should be a list of features that you use for designing and implementing your product. There is no need to go into a lot of detail about the features at this stage. You add detail when you are implementing the feature.

You may describe the features on the list using the input/action/output model that I showed in Figure 3.2. Alternatively, you can use a standard template, which includes a narrative description of the feature, constraints that have to be considered, and other relevant comments.

Figure 3.11 The iLearn authentication feature

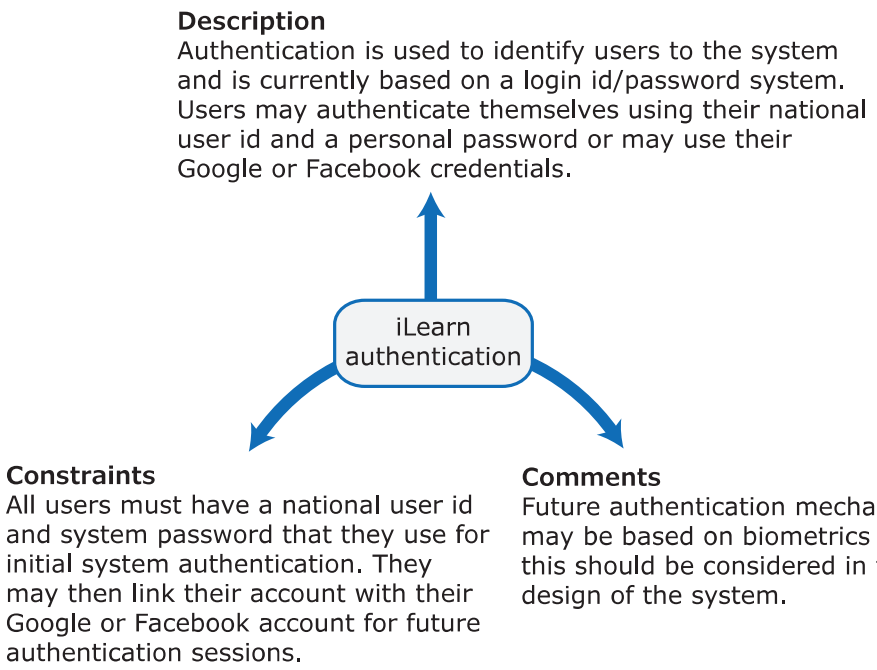


Figure 3.11 is an example of this feature template that is used to describe the system authentication feature in the iLearn system.

The descriptions associated with the feature can sometimes be very simple. For example, a Print feature might be described using the simple feature template that I introduced at the beginning of the chapter:

Print **the** *document* **to** *a selected printer* **or to** *PDF.*

Alternatively, you can describe a feature from one or more user stories. Descriptions based on user stories are particularly useful if you intend to use Scrum and story-based planning when developing the software.

Table 3.11 shows how you can describe the configuration feature of the iLearn system using user stories and the feature template shown in Figure 3.11. In this example, I have used an alternative text-based form of the feature template. This is useful when you have relatively long feature descriptions. Notice that the table includes user stories from the system manager and a teacher.

The product development team should meet to discuss the scenarios and stories, and it makes sense to set out the initial list of features on a whiteboard. This can be done using a web-based discussion, but these are less effective than a face-to-face meeting. The feature list should then be recorded in a

Table 3.11 Feature description using user stories

iLearn system configuration
Description As a system manager, I want to create and configure an iLearn environment by adding and removing services to/from that environment so that I can create environments for specific purposes. As a system manager, I want to set up sub-environments that include a subset of services that are included in another environment. As a system manager, I want to assign administrators to created environments. As a system manager, I want to limit the rights of environment administrators so that they cannot accidentally or deliberately disrupt the operation of key services. As a teacher, I want to be able to add services that are not integrated with the iLearn authentication system.
Constraints The use of some tools may be limited for license reasons so there may be a need to access license management tools during configuration.
Comments Based on Elena's and Jack's scenarios

shared document such as a wiki, a Google Sheet, or an issue-tracking system such as JIRA. Feature descriptions may then be updated and shared as new information about the features emerges.

When you have developed an initial list of feature ideas, you should either extend your existing prototype or create a prototype system to demonstrate these features. As I said in Chapter 1, the aim of software prototyping is to test and clarify product ideas and to demonstrate your product to management, funders, and potential customers. You should focus on the novel and critical features of your system. You don't need to implement or demonstrate routine features such as Cut and Paste or Print.

When you have a prototype and have experimented with it, you will inevitably discover problems, omissions, and inconsistencies in your initial list of features. You then update and change this list before moving on to the development of your software product.

I think that scenarios and user stories should always be your starting point for identifying product features. However, the problem with basing product designs on user modeling and research is that it locks in existing ways of working. Scenarios tell you how users work at the moment; they don't show how they might change their ways of working if they had the right software to support them.

User research, on its own, rarely helps you innovate and invent new ways of working. Famously, Nokia, then the world leader in mobile (cell) phones, did

extensive user research and produced better and better conventional phones. Then Apple invented the smartphone without user research, and Nokia is now a minor player in the phone business.

As I said, stories and scenarios are tools for thinking; the most important benefit of using them is that you gain an understanding of how your software might be used. It makes sense to start by identifying a feature set from stories and scenarios. However, you should also think creatively about alternative or additional features that help users to work more efficiently or to do things differently.

KEY POINTS

- A software product feature is a fragment of functionality that implements something a user may need or want when using the product.
- The first stage of product development is to identify the list of product features in which you name each feature and give a brief description of its functionality.
- Personas are "imagined users"— character portraits of types of users you think might use your product.
- A persona description should paint a picture of a typical product user. It should describe the user's educational background, technology experience, and why they might want to use your product.
- A scenario is a narrative that describes a situation where a user is accessing product features to do something that they want to do.
- Scenarios should always be written from the user's perspective and should be based on identified personas or real users.
- User stories are finer-grain narratives that set out, in a structured way, something that a user wants from a software system.
- User stories may be used to extend and add detail to a scenario or as part of the description of system features.
- The key influences in feature identification and design are user research, domain knowledge, product knowledge, and technology knowledge.
- You can identify features from scenarios and stories by highlighting user actions in these narratives and thinking about the features that you need to support these actions.

RECOMMENDED READING

"An Introduction to Feature-Driven Development" This article is an introduction to this agile method that focuses on features, a key element of software products. (S. Palmer, 2009)

https://dzone.com/articles/introduction-feature-driven

"A Closer Look at Personas: What they are and how they work" This excellent article on personas explains how they can be used in different situations. Lots of links to relevant associated articles. (S. Golz, 2014)

https://www.smashingmagazine.com/2014/08/a-closer-look-at-personas-part-1/

"How User Scenarios Help to Improve Your UX" Scenarios are often used in designing the user experience for a system. However, the advice here is just as relevant for scenarios intended to help discover system features. (S. Idler, 2011)

https://usabilla.com/blog/how-user-scenarios-help-to-improve-your-ux/

"10 Tips for Writing Good User Stories" Sound advice on story writing is presented by an author who takes a pragmatic view of the value of user stories. (R. Pichler, 2016)

http://www.romanpichler.com/blog/10-tips-writing-good-user-stories/

"What Is a Feature? A qualitative study of features in industrial software product lines" This academic paper discusses a study of features in four different systems and tries to establish what a "good" feature is. It concludes that good features should describe customer-related functionality precisely. (T. Berger et al., 2015)

https://people.csail.mit.edu/mjulia/publications/What_Is_A_Feature_2015.pdf

PRESENTATIONS, VIDEOS, AND LINKS

https://iansommerville.com/engineering-software-products/features-scenarios-and-stories

EXERCISES

3.1. Using the input/action/output template that I introduced at the beginning of this chapter, describe two features of software that you commonly use, such as an editor or a presentation system.

3.2. Explain why it is helpful to develop a number of personas representing types of system user before you move on to write scenarios of how the system will be used.

3.3. Based on your own experience of school and teachers, write a persona for a high school science teacher who is interested in building simple electronic systems and using them in class teaching.

3.4. Extend Jack's scenario, shown in Table 3.5, to include a section in which students record audio reminiscences from their older friends and relatives and include them in the iLearn system.

3.5. What do you think are the weaknesses of scenarios as a way of envisaging how users might interact with a software system?

3.6. Starting with Jack's scenario (Table 3.5), derive four user stories about the use of the iLearn system by both students and teachers.

3.7. What do you think are the weaknesses of user stories when used to identify system features and how they work?

3.8. Explain why domain knowledge is important when identifying and designing product features.

3.9. Suggest how a development team might avoid feature creep when "it is" to be in agreement with "a team" faced with many different suggestions for new features to be added to a product.

3.10. Based on Elena's scenario, shown in Table 3.7, use the method of highlighting phrases in the scenario to identify four features that might be included in the iLearn system.