Chapter 5

Refining User Stories by Grooming the Product Backlog

You learned in the previous chapter that iterative discovery of desires involves expressing user stories with the help of a product backlog. The purpose of this chapter is to learn how to groom the product backlog so that you can plan sprints that will increase the quality of feedback loops.

In this chapter, you will learn the importance of the product owner for the product backlog. This chapter discusses how the team refines user stories by grooming the product backlog. Grooming is the act of ranking, illustrating, sizing, and splitting user stories. You will see how to use collaboration boards to make explicit the grooming process, with a minimum of formality. Finally, it concludes by explaining how to organize effective sprints with story mapping.
Managing the Product Backlog

Nowadays, it is unlikely that new software must address the needs of a single stakeholder. On average, there are easily between 10 and 20 stakeholders. This requires the involvement of several people. If the product backlog is an ordered list, and the stakeholders are responsible for setting the priority, how do you ensure the list actually gets sorted and that every item does not end up being poorly defined? Assigning the product backlog ownership to a group of people is not a viable solution. Scrum recognizes this issue by defining a specific role for this responsibility, the product owner.

The product owner is responsible for ensuring that the product backlog is always in a healthy state. He is the primary interface between the development team and the stakeholders. The product owner is the definitive authority on all that concerns requirements. His main responsibility is to decide the ordering of what will be built and list these decisions into the product backlog.

One of the primary qualities of the product owner is to be the bearer of the vision. He understands the big picture. This knowledge gives that person the authority to prioritize the importance of the desirements expressed by stakeholders. Faced with the unexpected, the product owner knows how to stay the course and is responsive to the stakeholders’ changes.

There is a lot of responsibility (both explicit and implicit) involved in managing the product backlog. Work will not get done without someone actively collaborating with stakeholders to understand customer/market needs and then communicating with the development team to ensure those needs are met. Being the product owner does not mean that he decides alone. The development team actively takes a hand in backlog management.
Within an agile framework, creating a new user story is an activity open to all. It can be done either by a stakeholder or by a team member. It is strongly recommended that stakeholders write the stories without requiring business analysts to act as a proxy between them and the team. There are cases in which the product owner creates a story in response to a request from stakeholders, but this scenario is not mandatory.

Because of her experience and know-how, there are similarities between the analyst and product owner roles. However, they are two different roles in the Scrum team. There is a major difference between a true analyst and a product owner. Product owners represent the business and have the authority to make decisions that affect their product. Typically, an analyst does not have this decision-making authority.

To have a true business analyst step into the role of product owner is possible but not always the best option. For example, here is a scenario in which a business analyst is probably not the best choice for owning and maintaining the product backlog. Say you are an independent software vendor selling software to thousands of users. In this case, someone must focus on both the customer and market, adapting the iteration plan and evolving the product roadmap. An analyst is not trained for that job.

You must realize that the evolving role of the analyst does not necessarily consist of being a product owner. Someone else with stronger marketing skills than the business analyst could also inherit this responsibility. In the next chapter, you will learn that, by default, the role of the analyst is now more tactical. He handles a myriad of details and still does analysis, but now mostly focuses inward on the delivery team.
This new, strategic role is more than just backlog prioritization. It is about facilitating software development over successive sprints and ensuring appropriate customer/market needs are inserted into that process.

Though this role is typically assigned to someone with technical background, someone from marketing or product management is probably just as qualified. If any of these people cannot fulfill the role, someone with a solid understanding of end users, the marketplace, the competition, or future trends can become the product owner. This is not a solitary role—the product owner is most likely part of a larger team—perhaps in product management (if an independent software vendor) or in a client-facing team (if in consulting).

Collaborating to Groom the Product Backlog

When dealing with emerging needs, it is impossible to keep the entire backlog in a ready state; only the top elements need to be. A healthy backlog provides a set of high-value, ready desirements, about equal in size, that are small enough so that the team can deliver them in the upcoming sprints. To obtain desirements that are ready to iterate, you need to periodically groom the backlog.

Even with all the improvements wrought by the Scrum framework, grooming the backlog remains, and likely will remain, a fundamentally human endeavor, fueled by the insights, ideas, passions, and perceptions of people looking for the best. Rather than letting stakeholders work of their own free will, the product owner must lead everyone by using a sequence of activities that promotes deliberate discovery. Grooming the backlog boils down to a sequence of four activities: ranking, illustrating, sizing, and splitting user stories, as shown in Figure 5.1.
These activities are never performed solo by the product owner. To accomplish these activities, the product owner must collaborate: first with stakeholders and then with the development team. Backlog grooming is a team effort.

**Ranking User Stories with a Dot Voting Method**

Although, according to the development team, the product owner is perceived as the one who decides the ordering of the backlog, it is actually not his decision. He must rely on stakeholders who are the ones who decide the importance of each story.

For the product owner, ranking user stories is actually a contact sport with stakeholders. It requires that he brings all his senses to the task and applies the best of his thinking, his feelings, and his communication skills to the challenge of facilitating decision making. The product owner is a facilitator, not a decider. Because he understands the process of grooming the backlog, he can guide stakeholders.
As mentioned in Chapter 3, “Discovering Through Short Feedback Loops and Stakeholders’ Desirements,” you can picture the communication between builders and stakeholders as a large room swathed in darkness. The product owner is in the room having conversations with stakeholders and what he hears is a cacophony of dissident voices. Because there is no light, little knowledge is derived from the situation. Now, imagine that the voice of each stakeholder emits a color when speaking.

The product owner steers the conversation by asking stakeholders what is the most important desirement. Soon, he will be surrounded with multicolored fireflies representing stakeholders’ desirements. By forcing the writing of the desirements as a user story, this can simplify the answers and increase the likelihood that the same desirement can repeated by several stakeholders. When user stories begin to accumulate, all the different colors merge, creating sparkling white lights. Order springs from the cacophony. These white lights are the important stories, those that the product owner must rank at the top of the backlog.

So far, the ranking process as described may seem abstract. Forget the abstract to be more practical. Usually, when discussing ranking, authors prefer to present the most common techniques, such as binary search tree, Kano analysis, MoSCoW (Must-Should-Could-Would), or other numerical assignment techniques. Now do the same by using one of the preferred methods: the dot voting technique (also known as spending your dollar technique). This established facilitation method is widely used by workshop facilitators for prioritizing ideas among a large number of people, and for deciding which are the most important to take forward.

The method is summarized as follows:

1. Post the user stories on the wall using yellow stickies or in some manner that enables each item to receive votes.

2. Give four to five dots to each stakeholder.

The product owner steers the conversation by asking stakeholders what is the most important desirement. Soon, he will be surrounded with multicolored fireflies representing stakeholders’ desirements. By forcing the writing of the desirements as a user story, this can simplify the answers and increase the likelihood that the same desirement can repeated by several stakeholders. When user stories begin to accumulate, all the different colors merge, creating sparkling white lights. Order springs from the cacophony. These white lights are the important stories, those that the product owner must rank at the top of the backlog.

So far, the ranking process as described may seem abstract. Forget the abstract to be more practical. Usually, when discussing ranking, authors prefer to present the most common techniques, such as binary search tree, Kano analysis, MoSCoW (Must-Should-Could-Would), or other numerical assignment techniques. Now do the same by using one of the preferred methods: the dot voting technique (also known as spending your dollar technique). This established facilitation method is widely used by workshop facilitators for prioritizing ideas among a large number of people, and for deciding which are the most important to take forward.

The method is summarized as follows:

1. Post the user stories on the wall using yellow stickies or in some manner that enables each item to receive votes.

2. Give four to five dots to each stakeholder.
3. Ask the stakeholders to place their votes. Stakeholders should apply dots (using pens, markers, or, most commonly, stickers) under or beside written stories to show which ones they prefer.

4. Order the product backlog from the most number of dots to the least.

When you are done with this first pass, it is almost certain that the stakeholders will not be completely happy with the outcome of the vote. If that is the case, you should review the voting and optimize it. Here’s what you can do:

1. Arrange the votes into three groups to represent high, medium, and low priorities.

2. Discuss stories in each group.

3. Move items around to create a high-priority list.

4. Make a new vote with items in the high-priority list.

The goal during this review is to start a discussion about each group. Discuss which user stories are a low or medium priority, and which must be delivered in the near future. Why are they low priority? After discussion, stakeholders may agree to move them into the high-priority list. Also, discuss the stories that are almost high priority and decide if you should move them in the high-priority list. When you are done with the discussion, repeat voting, this time using only the items that belong to the high-priority list. Finish this second vote by ordering the product backlog from the most number of dots to the least.

Identifying the user stories that are top priorities is the first step of a two-step process. The second step is to ensure that the stories are small enough so that the team can build them in a sprint. To achieve this goal, the product owner must shift focus and start discussions with the development team. Unlike stakeholders, the team members are the ones who can measure the size of user stories.
Sizing requires a rough understanding by the development team of the user experience. The user experience enables stakeholders to discuss the success criteria. These criteria say in the words of the stakeholders how they expect the software to behave.

During this second step, seek to quickly define success criteria, so the team estimates the size of stories as soon as possible and with minimum effort. A storyboard is the perfect medium for achieving this goal. If user stories help monitor conversations with stakeholders, storyboards help to illustrate expectations rapidly and cheaply. They are concrete examples that provide the explicit information required by the development team.

**Illustrating User Stories with Storyboards**

As experience teaches, stakeholders love to envision the software from the user interface standpoint. As a result, often they specify how the software should work, rather than just what it is supposed to do. This is why illustrating user stories with a storyboard is so efficient.

Storyboards, as we know them today, were developed at the Walt Disney Studios in the 1930s. The first storyboards evolved from comic book-like story sketches. They were used to “preview” an animation before a single animated cartoon was produced.

Figure 5.2 shows an example of a storyboard for an animated film. Not only does a storyboard make possible a dress rehearsal of the final product, but also by posting it on the wall, it elicits early feedback and encourages quick, painless editing, leading to significant savings in time and resources.
For the general public, a storyboard means drawing pre-production pictures for video production, animation, and film making. Unfortunately, too few know that storyboarding also applies to software development. It helps to illustrate the important steps of the user experience.

It is tough to capture the big picture without visually depicting the user story. Explaining requirements from the perspective of the user interface helps to turn unspoken assumptions into explicit information. In addition, explicit information helps stakeholders think and communicate effectively. To keep up a healthy conversation between stakeholders, the product owner, and the development team, each user story should be enhanced with a storyboard. During specifications, the screens required to illustrate the user story are roughly sketched, either on paper or through the use of computer-based software.

Do not expect the storyboard to be a visual prototype that looks like the final user interface. It is an artistic rendition in which many details are missing. A storyboard is a low-fidelity visual aid that communicates the visible behaviors of a user story.
The process of visual thinking enables stakeholders and the product owner to brainstorm together, placing their ideas on storyboards and then arranging them in a structured way. This fosters more ideas and generates consensus within the group. The reason for the usefulness of a storyboard is that it helps stakeholders, as well as the development team, understand exactly how the user story will work. It is also more cost-effective to make changes to a storyboard than to an implemented user story.

The simplest technique for creating storyboards is *paper prototyping* [1]. It involves creating rough, even hand-sketched, drawings of the user interface to use as throwaway prototypes. All interactions within the prototype are simulated. Although paper prototyping is sketchy and incomplete, this simple method of communication with stakeholders can provide a great deal of useful feedback that can result in the design of better user stories. Figure 5.3 demonstrates how you can easily sketch ideas, test them almost instantaneously with stakeholders, and get rapid feedback on what does and does not work.

![Figure 5.3 An example of a paper prototype.](image)
After collecting and visualizing ideas on how the user interface might look, when there is a consensus on the user experience, it is desirable to keep an electronic copy of the storyboard for future reference. The simplest technique is to transform the paper prototype into a low-fidelity computer-based storyboard. The storyboard can then be used as a visual illustration of the user story, which will be shared with the development team. It is important, however, to make sure that you do not use a software tool that attempts to make the user interface similar to the final product. These high-fidelity tools encourage precision, and specifying all the details is time-consuming and deemed inappropriate at this time.

Figure 5.4 shows a low-fidelity computerized storyboard for the following user story, “As a student, I want to select a transit fare so that I can buy it.”

Figure 5.4 A computerized low-fidelity storyboard.
Designing the storyboards is always the responsibility of the product owner. He may nonetheless be assisted by the team members while performing his duties. For example, business analysts can help to complete the computerized storyboards. However, this activity is essential for obtaining a healthy backlog, and the product owner must be fully involved.

**Sizing User Stories Using Comparison**

The biggest and most common problem product owners encounter is stories that are too big. If a story is too big and overly complex while being a top priority, the sprint is at risk of not being properly completed. To avoid this issue, product owners must identify, as early as possible, if a user story is the right size and therefore ready to be built during a sprint.

It is not the responsibility of the product owner to estimate the work that needs to be done to complete each story. Only the development team can identify the size of a story. After the development team makes those estimates, the product owner can then determine if the story is too big. If that is the case, with the help of the team, she will split it into smaller stories.

To estimate the size of the top stories in the backlog, the product owner must organize recurring backlog grooming meetings. All the members of the development team must attend these meetings. To answer any questions addressed during these meetings, subject matter experts (stakeholders) should also participate. Before the meeting occurs, the product owner prioritizes the story list, thereby ensuring the most important stories will be estimated. The meeting is then time-boxed, at usually one hour, and each story is considered. Don’t worry if you don’t have time to discuss all the stories in the backlog. They will be addressed in future meetings.

Sizing a story requires that the development team estimates the work to be done to complete it. This should be simple, but unfortunately human beings are not good at estimating. Actually, we are not good at all.
Cognitive scientists tell us that our brain is wired to imagine the possible. We are reluctant to identify limitations, especially if they are not obvious. It seems that we are too optimistic, and indeed, we would not have survived the evolution of our species without this trait. With this bias built into our genetic background, it is almost impossible for us to accurately estimate, at least in a short time. It is obvious that with a lot of resources and enough time, humans just get there. However, this is not our case as we seek to estimate a user story in less than 5 minutes.

Does this mean that the development team should not estimate? Yes, at least according to what the word “estimate” means today. I propose that you estimate differently. Stop measuring absolute values and start comparing relative values. When estimating, you should not measure effort but instead compare efforts using a reference point.

Humans are poor at estimating absolute sizes. However, we are great at assessing relative sizes. For example, imagine that a team must estimate the weight of a young child and an adult. It will be difficult to agree on the exact weight of each. However, it will be extremely easy to decide which one is heavier.

When you measure stories, you need to be concerned with only relative sizes. You can easily do this by using the Fibonacci sequence or series, which is “A sequence of numbers, such as 1, 1, 2, 3, 5, 8, 13..., in which each successive number is equal to the sum of the two preceding numbers”. What is of interest, in this sequence, is the ratio between any number and its counterpart. This series gives you a relative size you can work with to compare effectively.

Our cultural tendency is to estimate based on how many hours it will take to complete a story. Unfortunately, estimating using duration reduces the team to measuring absolute values, which is what we want to avoid. Because of our incapacity to anticipate the unknown and to predict risk, we should steer clear of estimating based on time. There are three reasons for this:
• The time necessary for teams to build one unit of work fluctuates from sprint to sprint. In a complex situation, there is no other choice than to work collaboratively. When a member of the team is absent, due to vacation periods or members leaving the team, the team’s capacity to deliver changes. As a result, if you measure effort based on the number of hours, you must perpetually revisit the estimates in the backlog.

• Estimating based on time requires you to take into account the slack time. This adds accidental complexity, which results in a more imprecise measure. Factoring slack time appropriately is difficult. You must take into consideration the fact that people have to check their emails, participate in other meetings, eat lunch, take breaks throughout the day, and so on.

• Each team will gauge risks differently. Some will plan for a large cushion of time to mitigate risk, whereas others will approach the challenge without compensation.

The best way to evaluate effort is to use a degree of difficulty summarizing numerically the effort, complexity, and risk. For every degree of difficulty, you will assign points. Story points are independent of variations engendered by units of time. Furthermore, they are the perfect unit for comparing relative values.

The challenge of using a points system is calibrating what the number of points means. Some team members may think a story is worth one point, whereas others may think it is worth 10 points. So, how do you solve such a problem? One of the ways of calibrating stories, and getting a joint agreement by all team members, is to look at previous examples of stories as a referential. The team ranks the stories from most difficult to least difficult. The most difficult will have more story points than the least difficult. The goal is to end up with representative stories of 1, 2, 3, 5, 8, 13, and 20 points. After those representative stories have been identified, the team can then decide how many points the new stories
should be awarded. Calibrating by story points enables a team to easily reach a consensus.

During backlog grooming meetings, you want insights from all team members. As a result, you should favor a consensus-based estimation technique. A well-known and effective technique is the planning poker technique. It was first introduced by James Grenning and later popularized by Mike Cohn in his book, *Agile Estimating and Planning* [3].

Approach this technique as though you were playing a game of poker. Each bet should target one story. Before each bet, the product owner presents a short overview of the story and demonstrates the storyboard to define the success criteria needed to finish it. While answering questions posed by team members, the product owner enhances these criteria, which could double or even triple the work needed for each story. When the question period is over, the Scrum master then chairs the meeting and gives each team member a deck of Fibonacci cards, as shown in Figure 5.5.

![Figure 5.5 Deck of Fibonacci cards.](image-url)
The idea is to have all participants use one of the Fibonacci cards to give a rough estimate of how many points she thinks a story is worth. When betting, everyone turns over their card simultaneously so as not to influence others. Those who have placed high estimates, as well as those who have low estimates, are given the opportunity to justify their reasoning. After the members have explained their choices, the team bets again until a consensus is reached. This estimation period is usually time-boxed at five minutes by the Scrum master to ensure structure and efficiency. If consensus is not reached within the set time, the product owner moves to the next story where the betting process begins again. The goal is to address and reach an agreement on as many stories as possible within one meeting.

When the meeting is over, the product owner takes into consideration the number of points assigned to each story. Some stories may be worth 20 points, whereas others are worth 5 points. The product owner must determine which stories are too large and therefore need to be divided into smaller stories. Splitting large stories allows the development team to approach each smaller story in a more productive manner.

**Splitting User Stories Along Business Values**

Most user stories are too large; at least, this is the trend we noted with teams transitioning to agile software development. We guess this is because it is difficult to understand the gist of what a user story is. We must go back to basics and remember that it was initiated by Extreme Programming (XP). In *Planning Extreme Programming* [4] by Kent Beck and Martin Fowler, a user story is defined in the following way:

“We demonstrate progress by delivering tested, integrated code that implements a story. A story should be understandable to customers and developers, testable, valuable to the customer and small enough so that the programmers can build half a dozen in an iteration.”
A story is a short description of a unit of software that works, delivers value, and generates feedback from stakeholders.

A rule of thumb used to determine whether a story is small enough is to take the average velocity of the team per iteration and divide it by two. The velocity is the number of story points completed during a sprint. The product owner should not plan stories that are bigger than one-half the velocity.

A common mistake made when splitting stories is to slice and dice along technical issues, such as along the development process line (design, code, test, and deploy) or along the architectural line (user interface, business logic, and database). In addition to being difficult to deliver and deploy, technical decomposition creates stories that generate little feedback because they are incomprehensible by stakeholders. These stories negatively affect the iterative discovery of the stakeholders’ desiresments. This is not the path to follow.

You should focus on the perspective of stakeholders by thin slicing stories that favor the business value. Thin slicing is based on evolutionary architecture; it provides stories that implement only a small bit of functionality, but all the way through the architecture layers of the software. Thin slicing always splits stories along self-contained increments of value and along self-contained bundles of work that include “design, code, tests, and deploy.” There are two usual patterns for thin slicing stories in a self-contained unit:

- **Division:** The division pattern provides smaller stories, often of equal size.

  When there are clear boundaries about operational workflow or data manipulation, our first choice is to divide along these lines. For example, if it makes sense, you should split along the workflow steps involved or split according to each variation in business rules. If this is not a successful track, try to split by the type of data the story manipulates or along create-read-update-delete (CRUD) boundaries.
• **Simplification:** The simplification pattern aims to remove what is not necessary.

• When division is not an option, you should reduce the scope of a large story by keeping only the bare minimum. This is not a popular choice with stakeholders. As always, everything seems essential, and this requires more demanding conversations. Consider applying the XP principle: Do the Simplest Thing That Could Possibly Work. Remove from the large story everything that is not indispensable. Create one or more stories to safeguard what is not essential. These non-essential stories will be placed at the bottom of the backlog, whereas the remainder and thinner story will continue its journey to the top of the backlog.

### Tracking User Stories with a Collaboration Board

Backlog grooming is a team effort. Everyone, including stakeholders, must collaborate to evolve user stories from the bottom to the top of the backlog. As shown in Figure 5.6, it is a dynamic and active workflow where stories are constantly enhanced.

![Figure 5.6](image-url)

*Figure 5.6 The backlog grooming workflow.*

The team must master each step of the workflow; otherwise, the story will not progress as expected. Team members must synchronize their efforts on a daily basis. Unfortunately, the backlog is of little use to guide this work. At best, you can add a status field to follow the process, but it
does not encourage collaboration. Instead, it is more efficient to use visual aids inspired by collaboration boards such as those offered through the Slingboards [5] platform.

A collaboration board communicates information by using sticky notes instead of texts or other written instructions. As shown in Figure 5.7, a collaboration board is a two-dimensional grid on which you move yellow stickies from column to column to guide the actions of team members.

Each column represents a state of the process, and each sticky note is a visual signal for guiding the collaboration. The aim is to move each sticky note from state to state to accomplish a workflow. The rows are used to group and organize the yellow stickies in a logical manner. If you expect to have only a few stickies, you can have a single row without any grouping.
A well-known example of a collaboration board heavily used by agile teams is a task board. A task board is a visual aid that guides the work of a team during a sprint. As shown in Figure 5.8, a task board is a constantly evolving summary of the team’s forecasts for the current sprint. It enables you to see at a glance what is done, what remains to be done, and who is working on what.

Figure 5.8 A task board is a well-known example of a collaboration board.

When a sticky note is moved from column to column, it serves as a signal for guiding the collaboration. More and more teams consider a task board as essential to ensuring a rich collaboration during the sprint. I believe the same is true during backlog grooming except that we must use a different collaboration board. Now see how you could create a grooming board to get the same benefits.
The most important items in a collaboration board are the columns because they make it possible to visualize the process. Several options are available to define the columns. As shown in Figure 5.9, a simple option would be to have a column for each step. A major disadvantage of this option is that you can hardly know when a step is completed. There is no visual signal to initiate collaboration between teammates.

![Figure 5.9](image)

A second option, as shown in Figure 5.10, is to alert collaborators by being explicit when a step is done. There are two disadvantages to this approach for grooming. First, this approach assumes that the process is linear, which is not true. Grooming requires a lot of backtracking, such as when splitting a story. Second, we are uncomfortable with a condition that states that the ranking is completed. Ranking is never completely finished and can occur at any time during the grooming.

![Figure 5.10](image)

A third option is to alert collaborators by signaling that a step is ready for processing. This is the option that you can adopt, as it applies well to the grooming process. Figure 5.11 shows what the collaboration board would look like with one row for the backlog.
Chapter 5  Refining User Stories by Grooming the Product Backlog

Figure 5.11  A collaboration board with “Ready” signals.

The contents of the sticky note, which are moved from column to column, should display relevant information to help teammates understand what is going on. Significant information improves communication and reduces interruptions. As shown in Figure 5.12, there are nine potential display areas on a collaboration sticker.

Figure 5.12  A collaboration sticker has nine display areas.

When we want to create a collaboration board to facilitate the grooming process, each sticky note is going to represent a user story. Figure 5.13 shows the final result for this type of sticker. Note that we have not used all display areas, only those that we considered necessary.
The blocked indicator is visually pinning a status tag to the sticker. This status tag enables you to visualize work that is not directly associated with the value-added steps being performed. It creates visibility and awareness and enables the right people to react quickly to that new status. A visual alternative to pinning is creating special columns in your collaboration board that fulfill the same purpose. Although this is valid, and many people do it, we prefer pinning to expose that something is going wrong, or not happening. Board real estate is expensive. If you start creating special columns for each status a sticky note can have, you might quickly fill the board with empty zones.

A collaboration board is a clear, simple, and effective way to organize and present work during grooming. It increases the efficiency and effectiveness of the work by making visible the rules of collaboration and thus facilitating the flow. Flow is the mental state of operation in which a person performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment in the process of the activity.

Visual collaboration keeps the group members in the flow united around common performance measures. It enhances communication and reduces friction by making explicit the information teammates care about. It helps teammates
• Understand and indicate priorities.
• Identify the flow of work and what is being done.
• Identify when something is going wrong or not happening.
• Cut down on meetings to discuss work issues.
• Provide real-time feedback to everyone involved in the whole process.
• See whether performance criteria is met.

Collaboration boards increase accountability and positively influence the behavior and attitude of team members and stakeholders. Team members define and choose their own work instead of having work assigned to them. High-visibility and clear guidelines ensure teammates cannot hide work (or nonwork) from each other. They know that at any moment, if they want to, they can, with zero overhead and without causing any discomfort to anyone, see exactly what everybody is doing. Boards tend to expose the flow, but it is done with ground rules that people find quite reasonable. Thus, accountability is achieved in a harmonious way because it boils down to the individual responsibility of updating the board. This builds transparency among team members, which in turn builds trust.

**Delivering a Coherent Set of User Stories**

Unfortunately, in an iterative and empirical process, it is not because collaborative work produces high-value desirements that you necessarily get a “usable” sprint. Often, collaboration also requires prioritizing low-value desirements to obtain a coherent whole with optimal value. The use of a visual aid is essential in achieving this know-how. In this regard, over the years, experienced practitioners have acknowledged the necessity of structuring the backlog along a two-dimensional collaboration
Delivering a Coherent Set of User Stories

board. This way of organizing the stories to avoid half-baked incremental iterations was initially promoted by Jeff Patton [6] and is now known as story mapping.

Story mapping is the act of using a collaboration board to help in planning sprints and ordering the backlog. As illustrated in Figure 5.14, it combines high-value and low-value user stories in a coherent set, thereby revealing sprints that are of perceptible value to the stakeholders.

Figure 5.14 Planning sprints with story mapping.

The yellow stickies are the user stories from the backlog. They are distributed along the process line on the horizontal axis and simultaneously along the level of necessity on the vertical axis. Finally, they are ordered in “usable” sprints by assessing the expected necessity. Visualizing the desirments according to the process lines enables you to iteratively cut ever closer to the heart of the prioritization challenge. By doing this, you can combine the low-value functionalities and hold everything together.
As is always the case with a collaboration board, it all starts by identifying the columns. Create as many columns as there are features in the process lines. A feature is a piece of high-level functionality; a business activity that delivers value and separates into several stories. Arrange features by usage sequence, with features used early on, on the left, and later on, on the right.

Continue by creating as many rows as there are upcoming sprints. In each sprint, split stories along its feature by placing them in the appropriate column and make them overlap if they are numerous.

Even if the horizontal axis organizes stories along process lines, it does not ensure small and testable stories. Small stories should typically represent a few days of work. Initially, this is not the case as almost all new user stories are too big. They are desirements that need to be disaggregated into a set of constituent stories. Splitting desirements along the level of necessity ensures the identification of simple stories that can be forecast in a sprint. By differentiating the bare minimum necessity from usefulness and delightfulness, the product owner can divide large stories into smaller ones. These smaller stories provide immediate value and can be delivered in a sprint.

Even if desirements expressed as user stories are a starting point in understanding requirements, because they help determine the scope of work during sprints, they are mainly used as a unit of planning and delivery. This is why the overall goal of story mapping is to create a suitable scope to establish a delivery plan.

Planning Work with User Stories

There is a close link between executable specifications and agile project management. The purpose of this book is not to discuss agile project management. There are good books that cover this topic. [7] That being
said, we cannot ignore that desiresments provide an effective unit of planning. As shown in story mapping, we plan sprints around desiresments. Actually, the strong adoption of story mapping by the agile community leads me to believe that we are not alone in thinking that agile planning is closely linked to requirements discovery.

Summary

In this chapter, you saw how to groom the product backlog by ranking, illustrating, sizing, and splitting user stories. You learned the importance of having a product owner—someone who not only leads backlog grooming, but also ensures that it is done in collaboration with stakeholders and the development team. You learned how to use collaboration boards to track user stories during the grooming process. Finally, this chapter concluded by explaining how to organize a delivery plan that provides immediate value to the stakeholders through the use of story mapping.

When a story has gone through the process of grooming, you have reached an important milestone, which is the transition from conversation to confirmation. If user stories and their storyboards help monitor conversations with stakeholders, success criteria help confirm expectations. Success criteria convey additional information about the story and establish the conditions of acceptation. They enable the team to know when it is done and they say, in the words of the stakeholders, how they expect to verify the desirable outcome. In this perspective, success criteria are a specification as important, if not more important, than the story. Success criteria are a key element of executable specifications. Therefore, the next chapter is dedicated specifically to the issue of confirming user stories.
References


Index

Numbers
80/20 rule, 3

A
Acceptance Test-Driven Development (ATDD), 77
acceptance tests, converting scenarios to, 98-101
CI (continuous integration) versus, 118-119
connecting with interface, 110-114
enhancing with test results, 119-120
implementing the interface, 115-117
internal DSL (domain-specific language), 104-109
red-green-refactor cycle, 101-103
refactor stage, 117
accessibility
defined, 125
testing practice for, 135
actions, defined, 75
adaptation events, 15-16
Adzic, Gojko, 77, 86, 87
agile, origin of term, 1
Agile Estimating and Planning (Cohn), 59
agile zone (uncertainty diagram)
described, 6
handling uncertainty in, 7-9
analysts
defined, 149
product owners versus, 46
role in specification workshops, 88-89
anarchy zone (uncertainty diagram), described, 6
annotations, 106
architects
defined, 150
role in nonfunctional requirements, 126
Are Your Lights On? (Weinberg and Gauss), 31
assumptions, verifying, 98-101
ATDD (Acceptance Test-Driven Development), 77
automating scenario confirmation, 101-103

B
BDD (Behavior-Driven Development), 77, 79, 105, 107-108
Beck, Kent, 60, 101
benefits in user stories, 38-40
boundaries (guardrails)
“can-exist” assumption, 22
common goal, 20
high-level feature set, 21-22
list of, 14
shared vision, 17-20
stakeholder involvement, 16-17
team creation, 14-16
bugs in product backlog, 42
burn-down charts, 120
business analysts. See analysts
business value, splitting user stories along, 60-62

C
calibrating tests, 114
“can-exist” assumption for software, 22
chaining context, 113
chaotic zone (uncertainty diagram), described, 6
Chelimsky, David, 79
CI (continuous integration), acceptance tests versus, 118-119
Cohn, Mike, 35-36, 42, 59
collaboration
confirming scenarios, 85-89
grooming product backlog, 48-49
collaboration boards
story mapping with, 68-70
tracking user stories, 62-68
collaborative construction, 140-141
collective code ownership, 140
commands, splitting scenarios into, 83-84
common goal for software, 20
comparison method, sizing user stories, 56-60

D
Definition of Done checklist, 138
deliberate discovery. See trial-and-error process
desirements. See also product backlog
communicating via feedback loops, 31-33
describing with user stories, 35-38
discovering with roles and benefits, 38-40
establishing ubiquitous language, 40-41
planning with, 71
prioritizing in product backlog, 41-43, 49-52
desires, perceptions versus, 32
developers, as team members, 15
division pattern (splitting user stories), 61
documenting scenarios by feature, 93
domain models, connecting acceptance tests with, 112
Domain-Driven Design (Evans), 40
dot voting method, 49-52
DSL (domain-specific language), 104-109, 112
duplicate scenarios, avoiding, 94-95

F
failed tests, calibrating with, 114
failure. See trial-and-error process
features defined, 70
documenting scenarios, 93
high-level feature set, identifying, 21-22
organizing scenarios by, 92
usage statistics, 2
feedback loops, 29-30
communicating stakeholder desiresments, 31-33
prioritizing, 31
Fibonacci sequence, 57, 59
FIT tabular format
Given-When-Then syntax versus, 77-79
scripting scenarios, 76-78
Fleming, Alexander, 26-27
fluuent interfaces, 107
formalizing in scenarios, 76
ubiquitous language, 81-83
Fowler, Martin, 60, 105, 112
functional quality, defined, 124

G
Gauss, Don, 31
Given-When-Then syntax
FIT tabular format versus, 79-81
scripting scenarios, 79-80
goals, common goal for software, 20
God Complex, 26
green stage for acceptance tests, 115-117
Grenning, James, 59
Index

grooming product backlog, 48-49
guardrails
  “can-exist” assumption, 22
common goal, 20
high-level feature set, 21-22
list of, 14
shared vision, 17-20
stakeholder involvement, 16-17
team creation, 14-16

H
Harford, Tim, 26
high-level feature set, 21-22
“How.” See solutions

I
illustrating user stories, 52-56
inspection events, 15-16
interfaces
  connecting acceptance tests with, 110-114
  implementing, 115-117
internal DSL (domain-specific language), 104-109, 112
internal quality
  collaborative construction and, 140-141
  defined, 124
  improving software with proven practices, 137-139
  nonfunctional requirements affecting, 137
internal stakeholders, 16-17
INVEST mnemonic, 37

J
Jeffries, Ron, 38
Jobs, Steve, 27
Joyce, James, 27

K
Keepence, Barry, 127

L
language, ubiquitous
  establishing, 40-41
  formalizing, 81-83

M
maintainability
  defined, 137
  proven practices for, 138
managing product backlog, 46-48
Mannion, Mike, 127
Matisse, Henri, 28
measurable quality objectives, setting for restrictions, 131-135
merging scenarios, 94-95
Model-View-Controller design pattern, 110
Model-View-Presenter design pattern, 110
Model-View-ViewModel design pattern, 110
Mugridge, Rick, 76
music, trial-and-error process in, 28
Index

N
naming
  concepts, 81
  projects, 17-20
nonfunctional requirements
  affecting external quality, 125
  affecting internal quality, 137
  collaborative construction and, 140-141
  defined, 124-125
  improving software with proven practices, 137-139
  translating to restrictions, 127-136
  as user stories, 127
North, Dan, 77

O
ordering. See prioritizing
organizing scenarios, 91-95

P
painting, trial-and-error process in, 28
pair programming, 141
paper prototyping, 54
Pareto Principle, 3
pattern matching, 106-107
Patterns of Enterprise Application Architecture (Fowler), 112
Patton, Jeff, 69
perceptions, desires versus, 32
performance
  defined, 125
  testing practice for, 135
planning
  sprints, 68-70
  with user stories, 71
Planning Extreme Programming (Beck and Fowler), 60
planning poker technique (estimating user story size), 59-60
portability
  defined, 137
  proven practices for, 138
preconditions, defined, 75
prioritizing
  sprints, 31
  user stories in product backlog, 41-43, 49-52
problems, defined, 31
problem-solving (trial-and-error process)
  feedback loops, 29-30
  communicating stakeholder desires, 31-33
  prioritizing, 31
for software requirements, 25-29
process summary, 148
product backlog
  grooming, 48-49
  managing, 46-48
  prioritizing user stories in, 41-43, 49-52
  story mapping, 68-70
  tracking user stories, 62-68
product owners, 15
  analysts versus, 46
  defined, 149
  designing storyboards, 52-56
  grooming product backlog, 48-49
managing product backlog, 46-48
sizing user stories, 56-60
programmers, defined, 150
programming interface
  Model-View-Controller design pattern for, 110
  purpose of, 112
proven practices
  improving software with, 137-139
  testing restrictions, 135-136

Q
  quality
    defined, 123
    functional quality, defined, 124
    measurable quality objectives, setting for restrictions, 131-135
  nonfunctional requirements
    affecting external quality, 125
    affecting internal quality, 137
    collaborative construction and, 140-141
    defined, 124-125
    improving software with proven practices, 137-139
    translating to restrictions, 127-136
  Quality Is Free (Crosby), 123
  Quality Software Management Systems Thinking (Weinberg), 123
  queries, splitting scenarios into, 83-84

R
  R&D (Research & Development), handling uncertainty in, 7-9
  ranking. See prioritizing
  red stage for acceptance tests, 110-114
  red-green-refactor cycle, 101-103
    green stage, 115-117
    red stage, 110-114
    refactor stage, 117
  refactor stage for acceptance tests, 117
  releases, sprints versus, 30
  reliability
    defined, 125
    testing practice for, 135
  removing technical considerations from scenarios, 89-91
  requirements. See also product backlog
    describing with user stories, 35-38
    discovering with roles and benefits, 38-40
    distinguishing from solutions, 4
    establishing ubiquitous language, 40-41
    feedback loops on, 29-30
    communicating stakeholder desirments, 31-33
    prioritizing, 31
  guardrails
    “can-exist” assumption, 22
    common goal, 20
    high-level feature set, 21-22
    list of, 14
    shared vision, 17-20
stakeholder involvement, 16-17
team creation, 14-16
prioritizing in product backlog, 41-43, 49-52
trial-and-error process, 25-29
uncertainty 
  handling, 7-9
  impact of, 5-7
Research & Development (R&D),
  handling uncertainty in, 7-9
restrictions
  linking with scenarios, 129-131
  setting measurable quality objectives, 131-135
  testing with proven practices, 135-136
  translating nonfunctional requirements to, 127-136
robustness
  defined, 125
  testing practice for, 135
roles in user stories, 38-40
S
scalability
  defined, 125
  testing practice for, 135
scenarios, 74
  confirming collaboratively, 85-89
  converting to acceptance tests, 98-101
CI (continuous integration) 
  versus, 118-119
  connecting with interface, 110-114
  enhancing with test results, 119-120
implementing the interface,
  115-117
internal DSL (domain-specific language), 104-109
red-green-refactor cycle,
  101-103
refactor stage, 117
research
testing practice for, 135
Scrum, origin of term, 1
security
  defined, 125
  testing practice for, 135
shared vision, 17-20
simple/complicated zone
  (uncertainty diagram)
  described, 6
  handling uncertainty in, 7-9
Index

simplicity
  defined, 137
  proven practices for, 138
simplification pattern (splitting user stories), 62
sizing user stories, 56-60
SMART principle, 127
software features, usage statistics, 2
solutions
  distinguishing from requirements, 4
guardrails
  *can-exist* assumption, 22
  common goal, 20
  high-level feature set, 21-22
  list of, 14
  shared vision, 17-20
  stakeholder involvement, 16-17
  team creation, 14-16
uncertainty
  handling, 7-9
  impact of, 5-7
sorting. See prioritizing
SpecFlow automation framework, 108
Specification by Example (Adzic), 77
specification workshops, 87-89
specifications
  described, 4
  need for, 2
splitting
  scenarios, 83-84
  user stories, 60-62
sprints, 29-30
  communicating stakeholder desirements, 31-33
  organizing scenarios for, 91-95
planning with story mapping, 68-70
  prioritizing, 31
stable foundation. See guardrails
stakeholders
  common goal, 20
  confirming scenarios with, 85-89
desirements
  describing with user stories, 35-38
  discovering with roles and benefits, 38-40
  establishing ubiquitous language, 40-41
  prioritizing in product backlog, 41-43, 49-52
feedback loops with, 29-30
  communicating stakeholder desirements, 31-33
  prioritizing, 31
involvement of, 16-17
  shared vision, 17-20
  splitting user stories, 60-62
  in trial-and-error process for requirements, 25-29
state machines, 75-76
  states in scenarios, 81-82
stories. See user stories
story mapping, 68-70
storyboards, 52-56, 74
StoryQ automation framework, 109
Sutherland, Jeff, 1
T

task boards, 64
TDD (Test-Driven Development)
context-specification testing, 116
red-green-refactor cycle, 101-103
unit testing with, 115
team
confirming scenarios with stakeholders, 85-89
creating, 14-16
members of, 149-150
technical considerations, removing from scenarios, 89-91
testability
defined, 137
proven practices for, 138
testers, defined, 149
tests
acceptance tests
CI (continuous integration) versus, 118-119
connecting with interface, 110-114
converting scenarios to, 98-101
enhancing scenarios with test results, 119-120
implementing the interface, 115-117
internal DSL (domain-specific language), 104-109
red-green-refactor cycle, 101-103
refactor stage, 117
calibrating, 114
on restrictions, 135-136
scenarios versus, 77
time, estimating based on, 57-58
tracking user stories, 62-68
traditional zone (uncertainty diagram)
described, 6
handling uncertainty in, 7-9
transitions, 81
trial-and-error process, 9
feedback loops, 29-30
communicating stakeholder desiresments, 31-33
prioritizing, 31
for software requirements, 25-29

U

ubiquitous language
establishing, 40-41
formalizing, 81-41

uncertainty

guardrails
“can-exist” assumption, 22
common goal, 20
high-level feature set, 21-22
list of, 14
shared vision, 17-20
stakeholder involvement, 16-17
team creation, 14-16
handling, 7-9
impact of, 5-7
unit testing
replacing with context-specification testing, 116
with TDD, 115

usability
defined, 125
testing practice for, 135
usage statistics for software features, 2
Index

user stories, 35-38. See also
  product backlog
confirming scenarios
  collaboratively, 85-89
illustrating with storyboards, 52-56
nonfunctional requirements
  as, 127
organizing scenarios, 91-95
planning with, 71
prioritizing in product backlog,
  41-43, 49-52
removing technical considerations from scenarios, 89-91
roles and benefits in, 38-40
scripting with scenarios, 74-84
  FIT tabular format, 76-78
  formalism in, 76
  Given-When-Then syntax,
    79-80
  language formalization, 81-83
  splitting scenarios into
    commands/queries, 83-84
sizing, 56-60
splitting, 60-62
story mapping, 68-70
tracking, 62-68
ubiquitous language with, 40-41
User Stories Applied (Cohn), 35

V
verifying assumptions, 98-101
vision, clarifying, 17-20

W
Walt Disney Studios, 52
Weinberg, Gerald, 31, 123
“What,” See requirements

X
XP (Extreme Programming), 60
Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to informit.com/register to sign in or create an account. You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:
- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you’re looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.