User Stories

User Story Notes

- Elicitation Techniques (SWEBOK Ch. 1 section 3.2)
- Prototyping (SWEBOK Ch. 15 section 5.3)
- Define Stakeholder's needs (29148-2019 6.3.3.3)
- 5 Essential Agile Techniques to Improve Your Requirements Documentation (Source)
- User Stories (Source)
- Elicitation Table (Source)

Elicitation Techniques (SWEBOK Ch. 1 section 3.2)

Click here to expand...

Elicitation techniques concentrate on techniques for getting stakeholders to articulate requirements relevant information. Since this task is very difficult and engineers need to understand that most stakeholders will not understand technical jargon there are some good methods to collect information without having to worry about unwilling or uneasy stakeholders when it comes time to understand their requirements for the development phase. Below is a list of curated techniques considered very effective for this very goal to be achieved.

- **Interviews** - Interviewing stakeholders is a “traditional” way of eliciting requirements. It is important to understand the advantages and limitations of interviews and how they should be conducted.
- **Scenarios** - Scenarios provide a valuable means for providing context to the elicitation of user requirements. They allow the software engineer to provide a framework for questions about user tasks by permitting “what if” and “how is this done” questions to be asked. The most common type of scenario is the use case description.
- **Prototypes** - This technique is a valuable tool for clarifying ambiguous requirements. They can act in a similar way to scenarios, by providing users with a context within which they can better understand what information they need to provide.
- **Facilitated Meetings** - The purpose of these meetings is to try to achieve a summative effect, whereby a group of people can bring more insight into their software requirements than by working individually. They can brainstorm and refine ideas that may be difficult to bring to the surface using interviews.
- **Observations** - The importance of software context within the organizational environment has led to the adaptation of observational techniques such as ethnography for requirements elicitation. Software engineers learn about user tasks by immersing themselves in the environment and observing how users perform their tasks by interacting with each other and with software tools and other resources.
- **User Stories** - and refers to short, high-level descriptions of required functionality expressed in customer terms. A typical user story has the form: “As a ___, I want ___so that ___ .” A user story is intended to contain just enough information so that the developers can produce a reasonable estimate of the effort to implement it.
- **Extra Techniques** - A range of other techniques for supporting the elicitation of requirements information exist and range from analyzing competitors’ products to applying data mining techniques to using sources of domain knowledge or customer request databases.

Prototyping (SWEBOK Ch. 15 section 5.3)

Click here to expand...

The notes above speak about how prototyping is a valuable tool for clarifying ambiguous requirements. In this section, we will take a deeper dive into prototypes and what makes one effective in achieving our goal of clarification. Below is a breakdown of some important prototyping concepts from their construction, use, and role.

Construction - Constructing a prototype of a system is another abstraction process. In this case, an initial version of the system is constructed, often while the system is being designed. This helps the designers determine the feasibility of their design.

Uses - There are many uses for a prototype, including the elicitation of requirements, the design and refinement of a user interface to the system, validation of functional requirements, and so on. The objectives and purposes for building the prototype will determine its construction and the level of abstraction used.

Role - The role of prototyping is somewhat different between physical systems and software. With physical systems, the prototype may actually be the first fully functional version of a system or it may be a model of the system. In software engineering, prototypes are also an abstract model of part of the software but are usually not constructed with all of the architectural, performance, and other quality characteristics expected in the finished product.

In conclusion, modeling, simulation, and prototyping are powerful techniques for studying the behavior of a system from a given perspective. All can be used to perform designed experiments to study various aspects of the system. However, these are abstractions and, as such, may not model all attributes of interest.

Define Stakeholder's needs (29148-2019 6.3.3.3)

Click here to expand...

There are four main tasks that are a part of defining the stakeholder's needs in your project, below is a list of them and the concepts that follow:

- **Define Context of use within operations concepts and lifestyle** - The ConOps describes an organization's assumptions or intent in regard to an operation or series of operations. It is often captured in long-range strategic plans and annual operational plans and can cover a series of connected operations that are carried out simultaneously or in succession. It provides the basis for bounding the operating space, system capabilities, interfaces, and operating environment. Sometimes, the context of use is captured using a Context of Use Description.
- **Identify stakeholder needs** - Identification of stakeholder needs includes elicitation of needs directly from the stakeholders, identification of implicit stakeholder needs based on domain knowledge and context understanding, and documented gaps from previous activities. Needs often include measures of effectiveness.
- **Define the stakeholder needs and rationale** - SEE BELOW SECTION

In most systems, there can be many sources of needs and ultimately, requirements, and it is essential that all potential sources are identified and evaluated for their impact on the system. Below is a list of common sources and issues, as well as good questions to direct you to the solutions:

- **Goals** - The term goal refers to the overall, high-level objectives of the system. Goals provide the motivation for a system but are often vaguely formulated. It is important to assess the value and cost of goals.
- **Mission profile** - There are important questions to ask when dealing with the mission profile like how will the system perform its mission? How will the system contribute to the business or organizational operations? These answers help build your mission profile.
- **Operational scenario** - Scenarios can be used to define operational concepts and to bound the range of anticipated uses of system products, the intended operational environment, and interfacing systems, platforms, or products. Scenarios help identify requirements that might otherwise be overlooked.
- **Operational Environment & Use Context** - Requirements are derived from the environment in which the system or software product will operate. Will it operate in hot or cold conditions, externally, or under other equally restrictive conditions? What are the characteristics, timing, and quantity of interactions with the system environment?
- **Operational Deployment** - some good questions to understand your operational deployment are when will the system be used? Will it be deployed during the initial, middle, or wrap-up phases of a need? these questions will start pointing you to your deployment concepts
- **Performance** - It is important to ask yourself and your team what the critical system parameters to accomplish the mission are.
- **Effectiveness** - How effective/efficient should the system be in performing its mission? What are the applicable measures of effectiveness? Does the system have to be available to perform its missions for a minimum amount of time, such as 90-percent of the time?
- **Operational Life Cycle** - How long will the system’s lifetime be? 20 years? 30 years? How many hours per year should the system operate?
- **Org Environment** - Many systems are required to support an organization's process and this may be conditioned by the structure, culture, and internal politics of the organization. In general, new systems should not force unplanned change in the business process.
- **User & Operational Characteristics** - – Who will be using or operating the system? How will they vary in the role, skill level, and expected workload? What are the expectations or constraints on their capability and availability? Should allowance be made for accessibility?

---

5 Essential Agile Techniques to Improve Your Requirements Documentation (Source)

Click here to expand...

Agile for requirements

When the requirements aren’t done the right way, there is a high chance of having an unstable foundation in the product lifecycle, which often leads to an unsatisfactory end product. In most traditional projects, the requirements team has a mental barrier between them and other teams. This mental barrier isn’t intentionally created, in fact, it is more of a cultural issue. Agile techniques play a key role in transforming that closed culture.

The Heart of Agile Requirements

User Stories, or stories as some might call them (or them), represent customer requirements in a simply written narrative rather than a tedious comprehensive document. It drives forward conversations within Agile teams for planning and estimation. They contain a number of criteria that can be used to determine when a User Story is considered to be complete. A good narrative for a User Story would be something that adds value for the customer, partner, consumer, or stakeholder.

Several stories make a Product Backlog and ideally the whole Agile team is responsible for the health of the Product Backlog. However, a Product Owner (PO) is appointed to serve as the single wring-able neck if something goes wrong with the end product. A dangerous yet common misconception in the Agile world is that the PO is the only person responsible for the Product Backlog which I’m afraid is the root cause of many failed products.

5 Agile Techniques for Documentation:

1. **Compliment User stories with supporting artifacts** - In cases where a User story does not have enough detail, you may attach use cases, traditional requirements, or decision tables. In some situations, a client may demand more documentation, especially while dealing with organizations that get audited for process compliance for e.g. CMMI. This approach may work out well in that scenario.
2. **Create requirements that slice the cake** -The best way to do this is by writing end-to-end User Stories which include smaller feature sets, rather than writing stories that are split across technical boundaries. Splitting stories along technical boundaries creates dependent stories which fail the INVEST principle, which is described below. When writing stories, think of slicing a multilayered cake where each layer represents a functional area of the product. For e.g. front end, middleware, backend, database, and so on.
3. **INVEST in your User Stories** - see below
4. **Groom your User Stories often** -Conduct User Story grooming workshops daily or weekly depending on how soon you need results. I remember being a part of these workshops during my career as a software developer and I can't tell you how valuable it was. It can be implemented by following these steps. First, brainstorm stories on a whiteboard, then organize them into their user themes, prioritize the stories into high, medium, and low and lastly improve high priority stories and make them follow INVEST principle.
5. **Don't be afraid to create prototypes** -It is better to spend a small amount of time building prototypes to just test out the waters. This really helps in bringing ideas to reality and encourages healthy discussions. However, make sure the prototype does not turn into a bulky project and build it so that it can be used in some useful way in the actual product.

The I.N.V.E.S.T. Strategy (In-Depth)

An ideal User Story would have the following characteristics: Independent, Negotiable, Valuable, Estimable, Small, and Testable. This is called the INVEST principle. Make stories as independent as possible. Avoid using stories as written contracts, they should be negotiable. Make User Stories valuable to the user and consumers. A story may not be estimable if the implementers lack domain knowledge or technical knowledge. POs and technical leads need to help the implementers gain the domain and technical knowledge by writing effective user stories which follow the INVEST principle and cause conversations focused on implementing the User Story. Predefined test criteria should be mentioned in the done criteria of an ideal User Story. POs should take testers' guidance for this and testers should be proactive in guiding the product owners.

User Stories (Source)

Click here to expand...

What are agile user stories?

A user story is the smallest unit of work in an agile framework. It's an end goal, not a feature, expressed from the software user's perspective. User Stories are an informal, general explanation of a software feature written from the perspective of the end-user or customer. The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

Why create user stories?

- **Stories keep the focus on the user.** A to-do list keeps the team focused on tasks that need to be checked off, but a collection of stories keeps the team focused on solving problems for real users.
- **Stories enable collaboration.** With the end goal defined, the team can work together to decide how best to serve the user and meet that goal.
- **Stories drive creative solutions.** Stories encourage the team to think critically and creatively about how to best solve for an end goal.
- **Stories create momentum.** With each passing story, the development team enjoys a small challenge and a small win, driving momentum.

Working with user stories?

Once a story has been written, it's time to integrate it into your workflow. Generally, a story is written by the product owner, product manager, or program manager and submitted for review. During a sprint or iteration planning meeting, the team decides what stories they'll tackle that sprint. Teams now discuss the requirements and functionality that each user story requires. This is an opportunity to get technical and creative in the team's implementation of the story. Once agreed upon, these requirements are added to the story.

How to write user stories:

- **Definition of "done"** — The story is generally "done" when the user can complete the outlined task, but make sure to define what that is.
- **Outline subtasks or tasks** — Decide which specific steps need to be completed and who is responsible for each of them.
- **User personas** — For whom? If there are multiple end-users, consider making multiple stories.
- **Ordered Steps** — Write a story for each step in a larger process.

- **Listen to feedback** — Talk to your users and capture the problem or need in their words. No need to guess at stories when you can source them from your customers.
- **Time** — Time is a touchy subject. Many development teams avoid discussions of time altogether, relying instead on their estimation frameworks. Since stories should be completable in one sprint, stories that might take weeks or months to complete should be broken up into smaller stories or should be considered their own epic.

The User Story Template:

“As a [persona], I [want to], [so that].”

Elicitation Table (Source)

Key	Summary	Description	T	P
CCM-31	As a Club Manager, I want access to monthly check in analytics and data so that scheduling can be made more accurately based on statisitcal analysis	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-30	As a Club Employee, I want to make reservations for members so that there are no conflictions of events	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-29	As a Club Manager, I want a statistical guess on availble parking during the day based on member checked in rates so that members can have an idea of the capacity	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-28	As a Club Manager, I want the daily log of all members checked in so there is a log for any neccessary reason to show proof of visitation	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-27	As a Club Manager, I want to see an estimation of how many employees should be on schedule based on statistical check ins so that there is enough staff during business operation	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-26	As a Club Employee, I want to charge members through the database so that members have access to goods and services that are tracked	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-25	As a Board Member I want access to quarterly statistics so that the board can see how many members are active	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-24	As a Club Employee, I want access to membership status so that members can know if their membership is active	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-23	As a Club Employee, I want to check-in members and guests so that I can verify their membership is active	“As a <role>, I want <goal /desire> so that <benefit>.”		
CCM-22	As a Club Employee, I want to search members from the database so that I can supply information on who is checked in	“As a <role>, I want <goal /desire> so that <benefit>.”		

10 issues