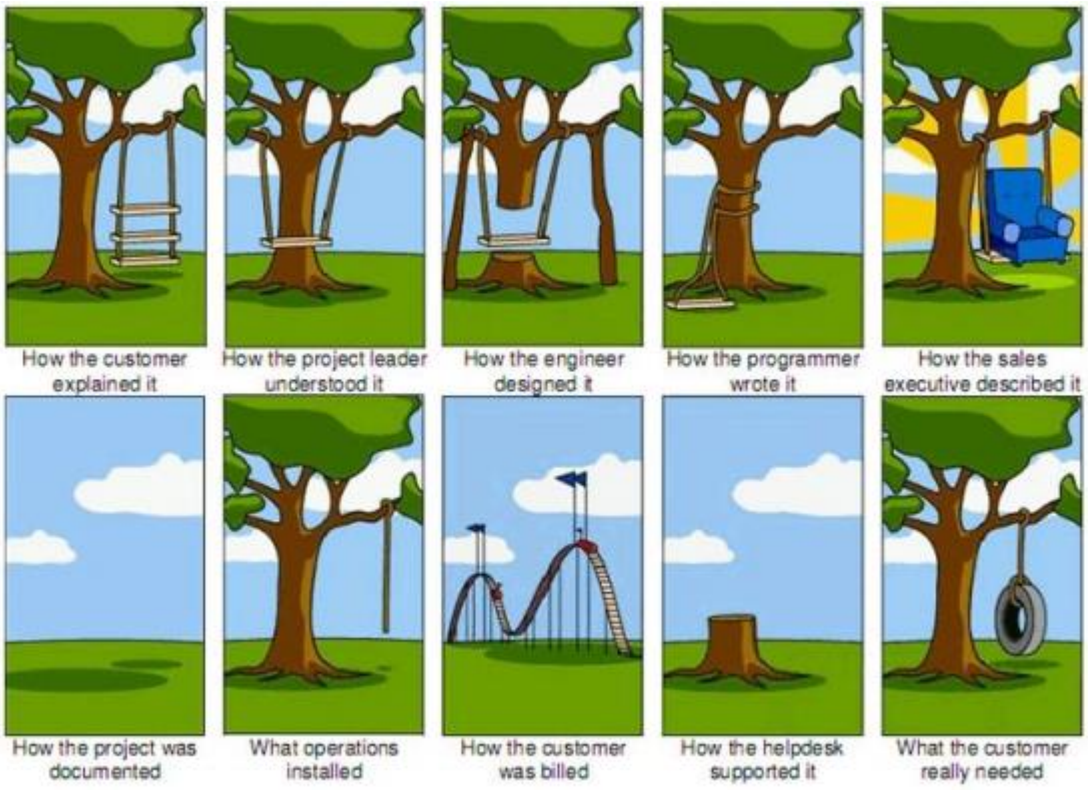
Requirements in Agile Software Development

Sam Huffer 6 October 2018



How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Figure 1: Screenshot from the DP1 lectures of what software development is. This image can used to outline what can result from poor requirements or miscommunication. Especially relevant to good requirements are the first four panels and the last.

What are requirements?

Requirements outline all aspects of a product, specifying the set of features that the client expects it to have, what it should be capable of doing, and the constraints it will have to work within (von Baggo 2018d). The purpose of requirements is to serve as a check list for software developers of what the client requires the solution to be, and to be able to do.

There are several types of requirements that developers need to work with when creating a product:

- Functional requirements, which dictate what the product should be able to do. They list the services or functionality the system should provide, how it should react to input, and how it should behave in specific situations (von Baggo 2018d), and implementation-agnostic, not caring about how they are met, only that they are met (Baruwal Chhetri 2018).
- Non-functional requirements, which outline aspects of the product that are visible to the user, but that are not directly related to the provision of specific features or functionality (Baruwal Chhetri 2018). Essentially, they dictate what the solution should be, or what characteristics it should have. These characteristics may be related to performance, and can cover areas such as organisation, safety, reliability, usability, and standards the product should meet (von Baggo 2018d).

- Constraints (or “pseudo-requirements”) that are imposed by the client or the environment in which the system will operate (Baruwal Chhetri 2018). They may include business or technical requirements and constraints (von Baggo 2018d).

What are correct, unambiguous requirements?

According to Baruwal Chhetri (2018), correct, unambiguous requirements possess several attributes distinguishing them from poor requirements:

- They are externally consistent with the client’s needs, in terms of being complete and accurate or correct representations of what the client requires of the product.
- They are clear, concise, and unambiguous.

Besides being correct and unambiguous, good requirements also possess the following additional traits:

- They are externally consistent with reality and are realistic and doable.
- They are internally consistent, and do not contradict each other.

What are User Stories?

User stories are a way of expressing requirements from the perspective of a user’s goal. User stories are written in the format “As a (role), I need (requirement or feature) so that (goal or value).” As such, they are a popular way to express requirements, the reasons for which include (Agile Business Consortium 2014):

As a player, I need to be able to shoot my opponent’s ships, so that I can win the game.

Figure 2: An example of a user story that could have been produced for Development Project 1*’s Battle Ship project.*

- They focus on the viewpoint of a user or stakeholder who will use or be impacted by the product.
- They define the requirement in language that has meaning for the user or stakeholder it is relevant to.
- They help to clarify the true reason for the requirement.
- They help to define and outline high level requirements, and avoid going into low level detail prematurely.
- They ensure each requirement is captured in a feature- and value-oriented way that is meaningful to stakeholders.

User stories are often considered to be comprised of three elements, which are collectively called “the 3C’s” (Agile Business Consortium 2014):

- The card that represents the user story. The front of the user story’s card features a unique identifier (such as a number or reference), a clear, explicit title of what the requirement is, and the requirement itself, expressed in the format described above. Meanwhile, the back of the card lists acceptance and testing criteria for the requirement, divided into functional criteria, and then groups of non-functional criteria (for example, availability and security).
- The conversation that high-level user stories represent. High-level user stories express large, broad requirements, and serve as a placeholder for later conversations that provide more detail as to what the client requires of the product. In those conversations, which occur just

before the development of a user story or group or stories, these high-level stories are broken down into more detailed user stories and that are small and clear enough for the development team to work from. When breaking these new stories down further during an iteration of development, their full detail may not even need to be recorded at all, but just incorporated directly into the product's code and design.

- Confirmation by the product's stakeholders, such as users, the owner, etc. that the stories, as expressed, match what they require of the product.

How are Functional Requirements and User Stories used in Agile Software Development?

User stories can be employed at each stage of the planning and development of a project in Agile Software development. During the stage of the project where the development team is assessing the project's feasibility, high level user stories can outline the objective(s) of the project. The user story format can clarify who needs the product, what they need it for and why, and what they expect of the product. At this stage, stories should constitute a small number of clear statements that are sufficient to scope the project, identify if it's worth proceeding, and to establish likely costs and benefits (Agile Business Consortium 2014).

When conducting foundational planning of the project, once it is established that it is feasible, more understanding of the stakeholders' requirements is needed, with enough detail to clarify the project's scope, prioritise tasks and schedule iterations, provide estimates of how long each task and iteration will take, and formulate a realistic plan for delivering the finished product. At this stage, the broad user stories established during the feasibility stage are broken down into simple, small functional and non-functional user stories. They need to be specific enough that the time they will take to complete can be estimated, and small enough to fit into a two to four week-long iteration (Agile Business Consortium 2014).

Once the development team begins iterations of actually developing and coding the product, small, more specific user stories and requirements become particularly important. At the start of each iteration, the user stories allocated to it are further broken down into even more detailed stories that are small and clear enough for the team to use as a check list of sections of code to be produced during that iteration. Only the stories for that iteration are elaborated on at once, allowing developers to manage the complexity of the project's requirements. The fine detail of the user stories that are elaborated on is only elicited immediately before that element of the solution is created, which allows the team to avoid wasting time hammering out the details for the entire project up front (Agile Business Consortium 2014).

During an iteration (as well as, more broadly, the whole project), the user stories and requirements that are outlined can be organised into a stack of prioritised requirements to be implemented now, or expanded upon later. They can be added, rearranged or removed as deemed necessary by the project's stakeholders at any time. That said, once the details of the current iteration's user stories and requirements are outlined, it might be advisable to freeze the iteration's requirements, unless changing or

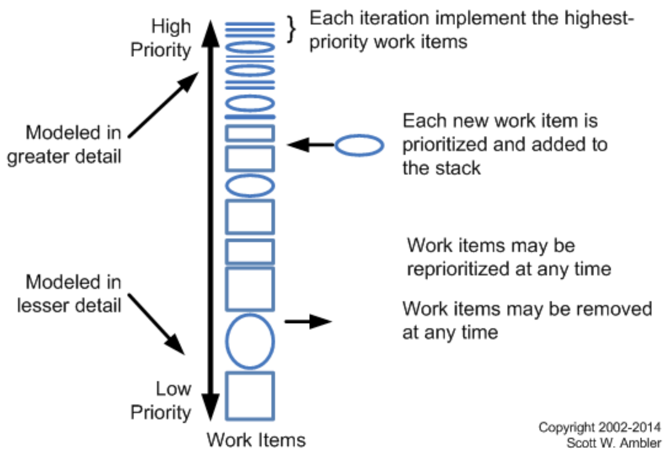

High Priority
Each iteration implement the highest-priority work items
Modeled in greater detail
Each new work item is prioritized and added to the stack
Work items may be reprioritized at any time
Modeled in lesser detail
Work items may be removed at any time
Low Priority
Work Items
Copyright 2002-2014 Scott W. Ambler

Figure 3: A visual representation of how user stories can be used as a stack of prioritised tasks that can be reorganised as the project's stakeholders see fit. Source: Ambler.

rearranging them is absolutely necessary, so as to provide some stability for the developers (Ambler).

What are some types of outcomes that can be affected?

Impact of Good and Bad Requirements in Traditional Software Development

Good requirements are the foundation of good outcomes for a product and are crucial to its success. If requirements are of a poor quality, are inaccurate, are incomplete, etc., then problems will arise later in the software development process, compromising the product's outcomes, sometimes to the point of it failing completely. In traditional software development, if bad requirements are identified later than is preferable, but with some time remaining to address them, they can lead to increased development costs, in terms of time spent by developers on avoidable reworking of the product – the particular requirements identified as bad and the number of such requirements would dictate the number and scope of the features needing reworking – and in terms of the financial cost of changes (to requirements or features) late in the development process (von Baggo 2018a). If they are identified too late, they could result in a product that is substandard and fails to meet a number of the client's actual needs properly, or that is a total failure due to not meeting the client's needs at all.

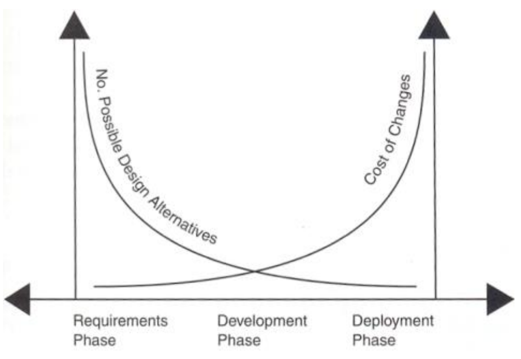


Figure 4: Screenshot taken from Topic 1: User-Centred Design *(von Baggo 2018a). If the "*No. Possible Design Alternatives*" line is reframed as an "*Ease of Change to Product*" line, the graph becomes more directly relevant to this report, and remains an accurate depiction of the cost of identifying a bad requirement at various stages of a traditional software development cycle.*

Impact of expressing Requirements as User Stories

Fortunately, the agile approach of using user stories to manage and express requirements mitigates consequences stemming from poor requirements that are identified as such late in the software development process. Because a) detailed user stories are only hammered out in direct collaboration with stakeholders just before their implementation, b) user stories express the requirements in a clear manner that is meaningful for and easily understandable by both the development team and stakeholders, and c) developers only implement a requirement expressed in a user story once stakeholders have confirmed that the user story expresses an actual need that they have (Ambler), the chances of an incorrect requirement becoming the basis of a portion of the finished product, and therefore the probable impact of poor requirements, are arguably greatly reduced.

Furthermore, user stories also provide a number of additional benefits to the project (Ambler):

- Because the amount of up-front work is reduced to the amount necessary to identify the project's scope, estimate task times, and develop a high-level schedule, developers don't need to waste time producing an SRS document that doesn't express requirements as well as the user stories.
- As noted earlier, because the detail is filled in just before implementation, user stories help with managing the project's complexity.
- Organising user stories in a rearrangeable stack (see above) provides much greater ability to organise and manage tasks, and much greater agility when responding to changes in requirements than working from an SRS would; if a requirement would change, the developers can

create one or more new user stories that express the updated requirement, and slot them into the stack according to the priority ascribed to the requirement and the fixing of related features by the stakeholders.
- Because the developers are using an agile, user story-centred approach to requirements, they demonstrate an intention to work collaboratively with users and stakeholders to discover what they actually require of the product (Agile Business Consortium 2014), and thus likely increasing the stakeholders' confidence that the developers will deliver the right product.

What are some guidelines for good requirements?

To create a successful product, we need requirements that outline what the product needs to do and be, and that can be used as the basis for producing a product that satisfies the needs of the client. As noted earlier, Baruwal Chhetri (2018) outlines several attributes that characterise good requirements:

- They are externally consistent with reality, being realistic and doable, and with the client's needs, in terms of being complete and accurate or correct representations of what the client requires of the product.
- They are clear, concise, unambiguous, and internally consistent with each other.
- They are verifiable and traceable, in that they can be traced back to a specific statement elicited during the requirements elicitation process or later feedback on the product where the client expressed a particular need.

In traditional software development, the following steps would be taken up-front to produce the requirements of the product:

1) Eliciting information from the client about what their needs are using an appropriate variety of data-gathering methods (von Baggo 2018b).
2) Preparing and synthesising the data obtained from the client, processing it into a more useful form for analysis (von Baggo 2018c).
3) Analysing the processed data using a number of methods to gain clear, precise insights into the client's needs and what requirements might be specified (von Baggo 2018e).
4) Extracting the product's requirements from the analysed data and specifying them formally in a Software Requirements Specification (SRS) document. The requirements themselves would be written in the following format (von Baggo 2018d):
    - **Name of major feature or category**: the overarching category that a number of requirements will fit into.
    - *Name of second level feature or category*: the particular feature or attribute that needs to be addressed.
    - **Requirement** statement: the details of what the what the required attribute or feature entails.
    - **Rationale** statement: an optional statement explaining why the feature or attribute is being specified in the requirements.
    - **Note**: optional commentary about the requirement that does not fit into either the requirement or rationale statements.

4.1 Utility Features
…
4.1.3 Calculator
Requirement: The patient shall have access to an in-app calculator.
Rationale: This feature will assist patients while shopping to calculate prices and change.
Note: The priority of this feature varies from patient to patient. On an average, this requirement gets low priority. This can be activated from the carer's interface as well.

Figure 5: An example requirement that was specified as part of the SRS produced during COS20001 User-Centred Design.

Agile software development, while still wanting requirements to possess the same attributes as traditional software development, employs different processes to produce them. Arguably, it should employ different processes, as traditional software development's inhibits agility, increases the up-front workload, and arguably increases the chances of encountering erroneous requirements.

Instead, when working with user stories in agile software development, which by their nature don't work very well with traditional requirements elicitation processes, one would be better served conducting brief model storming sessions to produce user stories as required (Ambler). They are aimed at discovering details behind a requirement by asking stakeholders (the best people to model the requirements, as they are the domain experts) to explain what they mean in impromptu meetings which typically last about five or ten minutes; it is rare to go for longer than thirty minutes, as requirements chunks explored in each session are so small. The modelling of the user stories is typically done by sketching on paper or a whiteboard, with the developers and stakeholders gathering together around the shared modelling tool to explore the issue until they're happy the developers understand the issue, and then they get back to work (which is often coding). The use of simple, inclusive modelling tools and techniques in model storming sessions enables maximum stakeholder participation, which is essential for producing good, accurate user stories.

Beyond the traits that define good requirements, user stories also should also conform to several criteria, known as the INVEST model, to maximise their effectiveness (Agile Business Consortium, 2014):

- Stories should be as **independent** as possible from other stories so they can be rearranged, and potentially implemented independently, with minimal impact on other user stories. Developers should consider combining tightly dependent stories into a single user story.
- Stories should be **negotiable**. They aren't a contract, but placeholders for features which the team will discuss and clarify with stakeholders near development time.
- Should represent features (not tasks) that provide clear business **value** to the product's stakeholders.
- User stories need to be clear enough for developers to be able to **estimate** for the appropriate timeframe, without the stories becoming too detailed.
- User stories should be **small** enough to be estimated. Larger "Epic" stories should be broken down into smaller stories as the project progresses. These smaller stories should also conform to INVEST criteria.
- User stories should be worded clearly and specifically enough to be **testable.**

Alternately, this series of simpler checks could be applied when evaluating user stories (Agile Business Consortium 2014):

- User stories should be clear, concise, and complete.
- Stories should not combine, overlap, or conflict with other stories.
- User stories should conform to organisational and project standards and policies where applicable.
- User stories should be traceable back to the business needs expressed by the stakeholder in the business case and project objectives.
- Where several stories relate to the same feature but for different users, they should cross-reference each other.

Regardless of the criteria applied to the user stories, and even the requirements, each requirement and user story should be approved by the project's stakeholders before it is developed from. It is their role to provide, clarify, specify, and prioritise requirements. The developer's role is to understand and implement them. Developers should as questions to get the stakeholders to specify their needs in greater detail as appropriate, and may even suggest requirements. However, they should only be developed from if the stakeholders adopt them, and it is their right to choose not to, or to modify your suggestions to match their actual needs; stakeholders are the only official source of requirements (Ambler). Project stakeholders that should be consulted, include the product owner, users, managers, operations staff members, support staff, testers, developers working on other systems that will interact with yours, and maintenance professionals.

References

Agile Alliance, *User Stories*, Agile Alliance, viewed 25 September 2018, <https://www.agilealliance.org/glossary/user-stories>.

Agile Business Consortium, 2014, 'Requirements and User Stories', in *The DSDM Agile Project Framework Handbook* (free web version), Agile Business Consortium, viewed 25 September 2018, <https://www.agilebusiness.org/content/requirements-and-user-stories>.

Ambler, S, *Agile Requirements Modelling*, Scott Ambler + Associates, viewed 25 September 2018, <http://agilemodeling.com/essays/agileRequirements.htm>.

Baruwal Chhetri, M 2018, *Lecture 2: Solving Problems and Version Control*, SWE20001: Development Project 1 – Tools and Practices, Learning materials on Blackboard, Swinburne University of Technology, 7 August, viewed 14 September 2018.

Von Baggo, K 2018a, 'Topic 1: User-Centred Design', in *Lecture 1: Introduction to Unit and User-Centred Design*, COS20001 – User-Centred Design, Learning materials on Blackboard, Swinburne University of Technology, 27 February, viewed 14 September 2018.

Von Baggo, K 2018b, 'Topic 2: Researching Context of Use', in *Lecture 2: Researching Context of Use, Qualitative Data Analysis, and Ethics of Human Research*, COS20001 – User-Centred Design, Learning materials on Blackboard, Swinburne University of Technology, 6 March, viewed 14 September 2018.

Von Baggo, K 2018c, 'Topic 3: Qualitative Data Analysis' in *Lecture 2: Researching Context of Use, Qualitative Data Analysis, and Ethics of Human Research*, COS20001 – User-Centred Design, Learning materials on Blackboard, Swinburne University of Technology, 6 March, viewed 14 September 2018.

Von Baggo, K 2018d, 'Topic 5: Requirements Specification' in *Lecture 3: Requirements Specification and Model Construction*, COS20001 – User-Centred Design, Learning materials on Blackboard, Swinburne University of Technology, 13 March, viewed 14 September 2018.

Von Baggo, K 2018e, 'Topic 6: Model Construction', in *Lecture 3: Requirements Specification and Model Construction*, COS20001 – User-Centred Design, Learning materials on Blackboard, Swinburne University of Technology, 13 March, viewed 14 September 2018.