Agile Backlogs

Sam Huffer

13 October 2018

Agile Backlogs

Agile backlogs are a prioritised (Radigan) list of features and technical tasks that the team maintains, with its contents being necessary and sufficient to complete a project (Agile Alliance). The items that comprise a backlog include requirements, and can include items like change requests, bug reports, and even planned team member absences (Wikistrand 2010). They are organised by the product owner, who determines the priority of particular features or tasks, and may reprioritise them at any time as they see fit, placing the most important items at the top so that the developers know what to deliver first (Radigan).

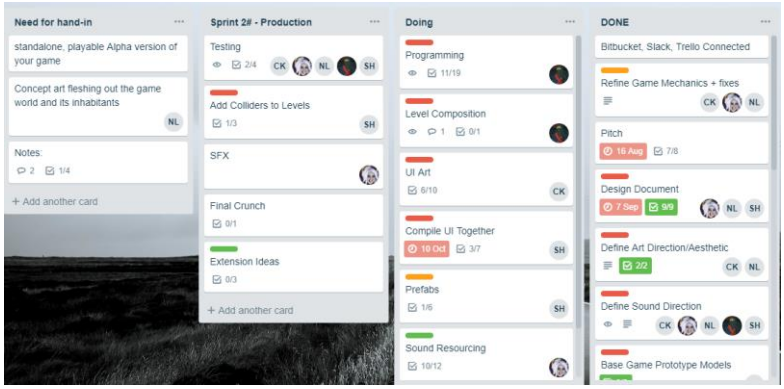

Figure 1: The Trello board my group in Digital Game Prototyping Lab *is using to track our work, with the (rough) backlog for sprint 2 featured on the centre-left. Earlier in semester it also featured a similar backlog for sprint 1, but that column has since been deleted.*

A key characteristic of a backlog is that it outlines all items of work your team intends to spend time on, thereby helping to set expectations with stakeholders and other teams, especially when they bring forward further work (Radigan). It should be noted that its contents should be expected to change throughout the project's lifecycle as the developers gain further knowledge of the features and tasks actually involved from the client. Further tasks that are identified as being necessary will be added to the backlog, and similarly, any items that are identified as not contributing will be removed (Agile Alliance), thereby keeping the backlog and the developers up-to-date with the product's requirements and other tasks required of the developers.

Projects won't necessarily have just one backlog; they may have multiple different backlogs. They could conceivably have backlogs for different categories of items to work on (e.g. bugs to fix, programming, animation or sound features to be implemented, etc.). Backlogs can also be used to group items by when they are to be completed, with items perhaps being organised by sprints or iterations (see fig. 1). Projects that would have larger backlogs should have items grouped into near-term items and long-term items (Radigan). Near-term items need to be fully fleshed out before being labelled as such (e.g. complete, detailed user stories), while long-term items can remain vague while they remain long-term, although it is a good idea to get a rough estimate from the developers to help prioritise them (these estimates will change once the team get a better understanding of and begin work on these longer-term items).

Agile Backlogs vs Software Requirements

A common pitfall in using backlogs is to confuse them with requirements documents; backlogs are not requirements documents (Agile Alliance). Backlogs are prioritised (Radigan) lists of features and technical tasks, with their contents being necessary and sufficient to complete a project (Agile Alliance). In comparison, a requirements document documents all the requirements of a project, which them-

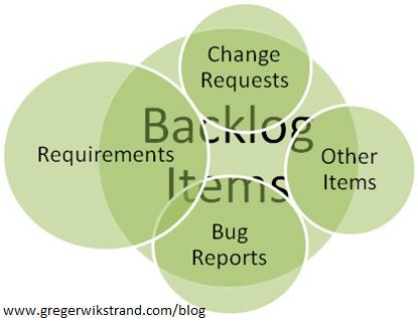

Figure 2: Wikistrand's outline of what can constitute backlog items, noting that while not all items in each category will necessarily be able to become backlog items, there is substantial overlap.

selves outline all aspects of the 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 2018).

That's not to say that requirements and backlogs cannot work together. Much of the contents of a backlog is derived from the requirements of the product; many backlog items are requirements or requirements-based tasks (Radigan). But that is not to say that all requirements can be listed as backlog items, as Wikistrand (2010) contends.

In addition, other tasks that don't directly address requirements but that are still tasks to be completed, such as bugs, design changes, technical debt (Radigan), change requests, bug reports and even planned team member absences, will constitute other backlog items (Wikistrand 2010). Synchronicity between backlogs and requirements could arguably be further enhanced by expressing requirements as user stories, which are managed in a manner that highly resembles how backlog items are managed (see below).

How Agile Teams work with Backlogs

Backlogs are used by agile teams to list tasks to be completed as prioritised by the product owner, and then organised by the developers into iteration or sprint backlogs, or into near-term and long-term backlogs according to when feel they will be able to address each item (Radigan).

Items will initially be described at a more macroscopic level, to be filled in later as appropriate (Agile Alliance). Once development begins, developers will break down the tasks and features to be delivered soonest, filling in their details with the assistance of the product owner (Agile Alliance). This process of progressing from vague backlog items to a detailed array of items is reminiscent of how user stories are initially broad placeholders for future discussions, and are broken down and the details filled in in collaboration with stakeholders in a series of brief model storming sessions (Ambler) as the iteration they are to be implemented in arrives (Agile Business Consortium 2014). As such, the use of user stories to express requirements would go hand in hand with how product backlogs are managed, as user stories are expanded upon and managed very similarly and their ideal characteristics (i.e. INVEST criteria) ease such management (Agile Business Consortium 2014).

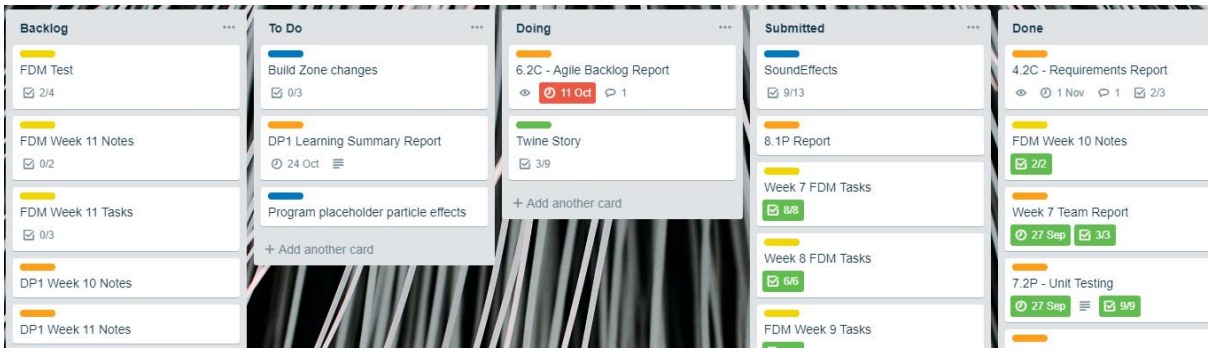

Figure 3: My Trello board that I am using to informally keep track of what tasks I will need to do (Backlog), what tasks I want to get done this week (To Do), what I am working on at the moment (Doing), what I have submitted that might require resubmission (Submitted), and what I have completed that does not require resubmission (Done). Once this report is completed, I will move it from Doing to Submitted pending it being marked off, after which it will be moved to Done.

From there, developers pull work from backlogs as there is capacity for it, either continually or by iteration (Radigan), moving items they are about to work now on from a backlog to a "to-do" column (Koren 2016). It can be inferred that from there developers would pull work items from the to-do section to the "doing" section as they start working on them, and move completed items to a "done" section, before pulling the next backlog item from the immediate to-do list (as illustrated by fig. 3). Once an iteration is over, the backlog items for the next iteration are expanded and moved from a backlog to "To Do", and the process repeats until the project is completed.

Impact of Good and Bad Backlogs on Work conducted and Project Outcomes

Good project planning is one of the key foundations of a successful project. Backlogs, as a task management tool, have great impact on the quality of the project's planning. Like user stories, they encourage collaboration with stakeholders to find out what they actually need (Ambler) so that the development team can deliver the right product. They streamline the breaking down of tasks into manageable chunks and the management of task complexity, and, by design, mandate the prioritisation by the product owner of what work should be done first (Radigan), as illustrated by my earlier examples (fig. 1, fig. 3) where high-priority or immediately due tasks being placed at the top of their respective lists and checklists breaking tasks down tasks into smaller, more manageable pieces. These benefits of backlogs help reduce developers' barriers to producing the right product in a timely manner, and therefore facilitate the project's success.

However, not every backlog enhances every project. Poorly managed backlogs that fail to give direction often result in the following negative symptoms (Kothari 2018):

- The developers struggle to define a solid plan during iteration planning sessions, or the product owner is unable to provide clear direction. This might be acceptable for a couple of iterations if the team is in the early stages of its agile movement, but it is still important to recognise this symptom and discuss it with the product owner to fix it.
- During iteration planning, the developers are unable to elicit the right level of detail for backlog items, risking being unable to deliver the required product.
- The developers find they are too often unable to complete at least 20% of the tasks they planned to do in each iteration. Some churn is expected, but if there is too much, you need to provide feedback to the product owner so that they can solidify the backlog grooming process (see below).
- Your customers did not like a recently released feature and provide negative feedback, indicating that their concerns were not considered or they were not consulted when when the product owner was grooming the backlog.
- The developers are always running out of funds for delivering must-have requirements, which results from the product owner prioritising nice-to-have requirements or insufficiently differentiating between the two.

Ultimately, a bad project backlog will result in development teams lacking understanding of the value of given tasks and struggling to deliver objectives, reducing profits and lending towards project failure (Kothari 2018).

Guidelines for Good Agile Backlogs

To avoid the aforementioned negative impacts of poor backlogs, agile teams might like to follow the following guidelines:

- Backlogs should be represented in physical form, such as by a collection of index cards or sticky notes. While there isn't any mandatory format for backlogs, having a physical backlog mitigates the risk of creating multiple conflicting versions, which runs counter to backlogs' function as a single source of the work that needs to be done (Agile Alliance). At the very least, the backlog should not be a document that is stored locally and shared infrequently, as this increases the risk of conflicting versions and also hinders interested parties from getting updates (Radigan).
- Backlog items should be "atomic", each expressing singular requirements or tasks, as opposed to one item describing several requirements or tasks, or several backlog items expressing what should be one singular item. This atomicity is also encouraged by physical backlogs (Agile Alliance).
- Backlogs should be regularly reviewed before each iteration planning meeting to ensure prioritisation is correct and feedback from the last iteration has been incorporated. This grooming or refinement of the backlog ensures it's a reliable and sharable outline of the work items for the project (Radigan).
- When reprioritising backlog items, the product owner should seek input and feedback from customers, designers, and developers to optimise everyone's workload and the product delivery (Radigan).
- While the product owner dictates the prioritisation of work items, the developers should dictate the rate at which they work through the backlog, addressing items only as they have the capacity to do so (Radigan).
- Once work is in progress, changes to the backlog (or the items in the iteration backlog) should be kept to a minimum so as not to disrupt the development team and affect focus, flow, and morale (Radigan).

Given the amount of overlap in how backlogs and user stories are managed, agile teams may also find it useful to keep in mind INVEST criteria, especially if they are using user stories to express requirements (Agile Business Consortium 2014):

- Backlog items should be as **independent** as possible from each other so they can be rearranged, reprioritised, and potentially implemented independently with minimal impact on other work items.
- Longer-term backlog items should be **negotiable**. They aren't a contract, but placeholders for features and tasks which the team will discuss and clarify with stakeholders near their implementation.
- Backlog items should represent features or task) that provide clear business **value** to the product's stake-holders.
- Backlog items need to be clear enough for developers to be able to **estimate** for the appropriate timeframe, without items becoming too detailed.
- Backlog items should be **small** enough to be estimated. Larger long-term items should be broken down into smaller items as the project progresses.
- Backlog items should be worded clearly and specifically enough to, where appropriate be **testable.**

There will likely be backlog items that don't fit neatly with INVEST criteria, which is not unexpected, as said criteria are originally intended for requirements expressed as user stories, not backlog items, many of which are not requirements. That said, it would be a good idea to keep the criteria in mind for requirements-based tasks and other tasks that they could be applicable and appropriate for.

References

Agile Alliance, *Backlog*, Agile Alliance, viewed 12 October 2018, <https://www.agilealliance.org/glossary/backlog>.

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>.

Koren, Y 2016, *The Critical Difference Between Backlog and To Do (Kanban, Scrum)*, AgileSparks, viewed 12 October 2018, <https://www.agilesparks.com/blog/backlog-vs-to-do/>.

Kothari, P 2018, *Recognizing the Five Symptoms of a Poor Backlog*, Scrum Alliance, viewed 12 October 2018, <https://www.scrumalliance.org/community/articles/2018/january/recognizing-the-five-symptoms-of-a-poor-backlog>.

Radigan, D, *The product backlog: your ultimate to-do list*, Atlassian, viewed 12 October 2018, <https://www.atlassian.com/agile/scrum/backlogs>.

Von Baggo, K 2018, '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.

Wikistrand, G 2010, *Agile Requirements and the Agile Backlog*, WordPress Foundation, viewed 12 October 2018, <http://www.gregerwikstrand.com/requirements-and-backlogs/>.