Refined, Ready, Accepted

Backlog Management for the Successful Agile Development Team

Vickie Hearne, Principal
Pierce/Wharton Research, LLC

Summary

Lack of "Ready" backlog items imposes a significant risk to every aspect of an Agile development project. Ready backlog effects the velocity of the team, the quality of the work, and the timeline of the effort. By ensuring that teams refine and solution together and adhere to standards of Ready and Accepted for development work, Agile teams are more likely to produce a quality product that is both on-time and in-budget.

Refining Backlog to Ready

Waterfall projects have an analysis and design phase that end before development begins. Once the development effort begins, modifications to the approved design are trailed into the "enhancement" or "maintenance" phase of the Software Development LifeCycle (SDLC).

Agile's incremental, iterative approach allows change to be more easily incorporated throughout a development project, but that does not mean change is continuous. Like Waterfall, Agile analysis ends before development begins; however, that end is for the individual development story ***only***. Rather than trail or discourage change, Agile "re-stories" change into the development backlog where it can be prioritized and groomed as business needs determine.

At the start of a project, an Agile team reviews the business requirements and ***use cases*** (beginning-to-end journeys), and then parses those needs into Epic and Feature-level objects. For larger projects, applications such as Microsoft's TFS, Jira, or VersionOne help manage stories, tasks, and bugs. Smaller teams rely upon spreadsheets and wikis. During backlog grooming and refining (analysis and design), analysts and product owners confer with the development team and other stakeholders to break down Epic and Feature-level requirements into "child" ***user stories.*** User stories are written specifically to support the development effort; requirements capture the needs of the business. "The Backlog" is ***not*** the same as "The Requirements."

Business requirements are rarely specific enough for the development effort

For the Agile novice, the level of granularity needed for a development story can be bewildering. While a requirement may be complete or written in an "As a [user persona], I want to [take an action] so that [Result or Reason]" statement, business requirements are rarely specific enough for the development effort. In most cases, it takes many Agile user stories to complete one business requirement fully. In all cases, Agile backlogs are significantly larger and more complex than their initial business requirements.

Best practices dictate that Agile stories be "full-stack," meaning any data model development or integration be demonstrated via a UI/UX interaction at sprint complete. While full-stack

is ideal, often dependencies and timelines necessitate that back-end tasks be parsed into different stories. This is where the concepts of Refined, Ready, and Accepted become essential to a successful application development project.

Because the size of development stories is small compared with an Epic-level business requirement, some stories can be Ready for development (ex: a portion of the UI/UX), while others are not Ready (ex: Active Directory integration). If the development team has unanswered questions or there are unknowns, the story is not Ready for development, and the story remains in the Backlog where it undergoes review, prioritization, and continued refinement until such time as it can be made Ready. For the duration of the project, high-level requirement gathering, backlog additions, and story refinement continue within the context of the project's Epics and Features.

In situations where the unknowns are too significant for a story to reach the Ready state, Agile dictates a research "spike" be created. Spikes allow research of critical unknowns ***before*** committing to development work. Spikes are different from stories because spikes do not involve coding or integration work. Whether spikes count against capacity is an individual team determination; nevertheless, under no circumstances, should development ever commence on stories that have unanswered questions or built-in spikes. If there are unanswered questions or missing information, the story is not Ready and should remain in the backlog.

When all questions have been asked/answered and the TEAM has clarity on the work being requested of it, the story is Ready.

Regardless of the size of the story, it is ***the team*** who determines if stories should be full-stack or split, and the team determines when there is sufficient detail in a story for it to be Ready for development. Ready is not a unilateral decision made by the product owner, architect, analyst or any one team member, nor is Ready a Definition of Done (DOD) checklist. Ready can differ from team-to-team and project-to-project. Ready is when all questions have been asked/answered and ***the team*** has clarity on the work being requested and delivered.

Lastly, a story can be Ready, but that does not mean that the development team has capacity to ***Accept*** the story into Sprint for development. In most cases, a Ready story is tagged as Ready, but Ready stories can be changed and refined up until the moment of Acceptance. However, once a story is Accepted, the refinement (analysis) of that story ends, and the story and its acceptance criteria does not change for the duration of the sprint.

Accepted: QA to the Analysis

The final step in the Agile story development process is Accepted. Accepted, also called Committed, is the QA to the Agile backlog grooming, refinement, and analysis effort.

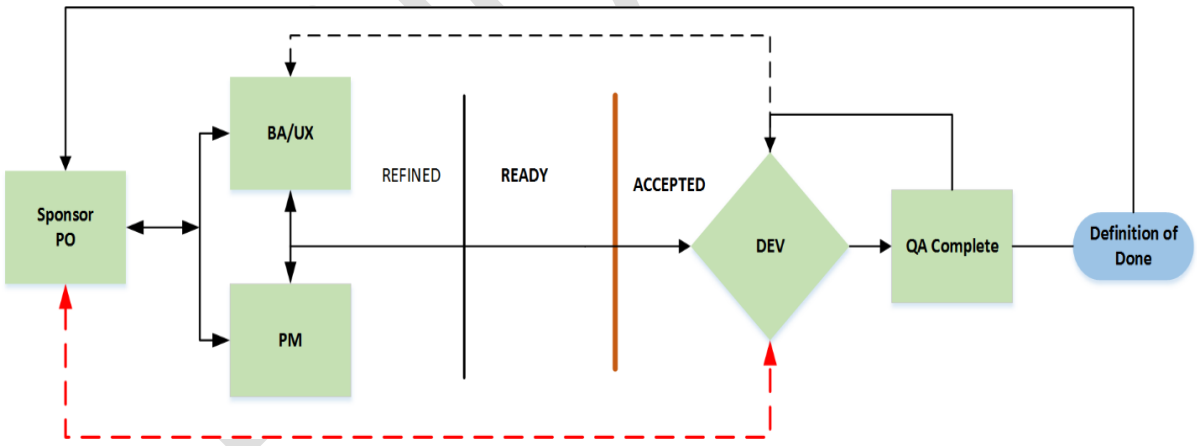
When a development team accepts a story into sprint, that acceptance confirms that the team understands the story, understands the acceptance criteria, understands what needs to be built, and understands how it will be tested. The team has discussed blocks, dependencies and data needs with their lead, affected groups, and each other. They have defined an approach, and then documented that approach with the appropriate time-boxed tasks and activities. Acceptance confirms that the developer(s) have read and understand the Epic, Feature, and story artifacts, and know what they need to do to demonstrate their work at the end of the sprint.

While many stories can be in a Ready state, that does not mean that the team has capacity to work on the story, and changes to the story can – and do – take place up until the time of Acceptance. Once a story is Accepted, however, the development team "owns" the story. The handoff from Ready to Accepted signals that the development team is able to "sprint" to finish the work. This is why it is essential that the team solutions together. If a developer is unclear about his/her tasks, the story is not Ready and should not be Accepted.

Backlog Grooming

The following diagram depicts the flow of information among team roles for the creation of backlog items. The flow connotes a linear relationship with the business analyst (BA), User Experience designer (UX) and Project Manager (PM) "feeding" the development team. It also implies the development team is separate from the business sponsors and product owners (PO). In reality, grooming and solutioning is a collaborative endeavor involving all members of the project team. [1]

At the start of the backlog development process, the entire team works together to develop and prioritize high-level business requirements into Epics and Stories that accommodate dependencies and constraints. For large projects, it might not be practical to include all developers in a grooming sessions. Leads may be involved in stakeholder sessions reserving individual team members for meetings to discuss specific stories as they become closer to Ready. At the time of acceptance, however, the developer who is doing the work should be familiar with the story, its criteria, dependencies, and how the work fits into the sprint and project. No one should ever be surprised by a story at the start of a sprint.



Feedback Loops

An additional risk of not Ready is the circumvention of legitimate feedback loops. If a story cannot be made Ready before coding begins, the development or QA team may turn to the PO or Sponsor for direction rather than the acceptance criteria provided by the user story. These clarifications may or may not align with other project requirements and may or may not conflict with the acceptance criteria of other backlog items.

[1] System stories are similar in process; however, architect and development roles are more forward than in the model shown

Changes made to in-sprint development are particularly problematic because the acceptance criteria of the current development become the assumptions for future development.

If Ready is not enforced, business sponsors also may seek to change approved stories and acceptance criteria by appealing to a developer directly, or may not be timely in providing input or approval because history shows that changes to work can be made at any time. As a result, deadlines and sprint commits are ignored.

Changing acceptance and assumptions mid-sprint complicates the ability to plan for work, budget for work, predict velocity, manage scope, meet deadlines, and execute QA test plans. Moreover, in-sprint changes made by business sponsors create risks and dependencies that need to be mitigated after they occur. The circumvention of feedback loops is a project risk that can be easily avoided by adhering to standards of Ready and Accepted.

Unplanned Work

The ability to "plan the work and work the plan" is essential for any successful project. Because Agile development work is executed in smaller increments, a common misconception is that Agile teams do not have to tightly plan work or that change is acceptable at any stage of the development effort. Agile does not eliminate the need to plan or control change; it only provides a more nimble mechanism to track and refine changes that do occur. In many ways, Agile teams require a much higher level of discipline and planning rigor than Waterfall efforts do.

If stories are Accepted into Sprint before they are Ready, there is a high-risk of ***unplanned*** work. Unplanned work -- burn-up -- is not the same as Agile grooming and refinement. Burn-up is a symptom of not Ready. Unfortunately, teams are often unwilling to track burn-up (or don't fully understand what burn-up is) or mistake burn-up caused by the lack of Ready for constant Agile change.

Burn-up is a symptom of not Ready.

The impact of not Ready effects more than just the development team. Project management, business analysts, UI/UX designers, and other project resources are diverted from their pre-sprint roles of defining new backlog, vetting solutions, and journey refinement for ***upcoming*** development to in-sprint activities of clarifying and approving ***current*** development. The diversion of resources from planning and analysis to development support affects the capacity of the entire team to create new, ready backlog for upcoming sprints.

Not Ready affects the ability of the entire team to groom and refine new backlog to Ready.

Even when stories are Ready and Accepted, there can be requests for changes to the development effort. Agile dictates these requests be added to the backlog as a new story rather than changing Accepted work. While in-sprint changes may not affect the developer's level-of-effort, even a minor change could require hours of meetings, approvals, and revisions from stakeholders or cross-functional

resources. Time spent managing these changes adversely impacts project management, analysis, and UIUX resources, as well as QA and UAT testing. If that work were part of the analysis and vetting for ***upcoming*** sprints, that time would be considered part of the backlog grooming and refinement process – a ***planned*** activity. However, when these activities occur for stories that are already ***in-sprint*** and under development, they are ***unplanned*** work and considered "burn-up."

Burn-up can block development, but a more serious repercussion is that burn-up affects the entire team's ability to groom backlog to Ready. The disruption to the planned activities of creating and refining stories doesn't just jeopardize the current sprint, it also imposes a significant risk that future sprints will also be incomplete or disrupted. Without time to groom backlog properly to Ready, there is often little choice but to force not-Ready work into the new sprint when the current sprint ends. Again, the result is that clarifications and questions arise and work is blocked or requires change. Again, project resources are diverted from pre-sprint planning and analysis to in-sprint development support (burn-up). The cascade/domino effort of work diversion compounds sprint-after-sprint jeopardizing deadlines and budgets. Worse, because coding commenced with incomplete analysis and erroneous assumptions, ultimately, the team has no choice but to take a "hardening" sprint to address technical debt, refactoring, or some dependency not considered before the story was Accepted. The root cause of this vicious cycle is development commencing on stories that were not Ready.

Why Backlog Isn't Ready

The following are the most common reasons project teams lack Ready backlog:

Business Requirements Mistaken for User Stories

While business process and product owners set priorities and approve requirements, these owners are not usually experts in the complexities of system integration and application development. As a result, the level of complexity and the number of stories needed to fulfill what is viewed as one simple requirement is commonly underestimated.

> *"As a cashier, I want to accept all credit cards, so that my customers can pay with all credit cards."*

Agile Sprint 0 encompasses planning, discovery, and "road mapping." This lays the foundation to create the backlog for the project. Backlog owners are wise to use this time to indoctrinate the uninitiated in backlog management and the level of granularity needed for development stories.

> *The Backlog is not the same as The Requirements.*

Sprint 0 should also focus on outputs such as defining Epic and Feature objects, and creating shared artifacts such as data diagrams, database dictionaries, As-is/To-be process flows, business rules, persona journeys, and other documents that support the ***global*** development effort. Shared artifacts are critical to Ready and Accepted. Too often, these data are mistakenly embedded in an individual story or completely absent from the backlog, which impinges both development and QA from finding needed information.

Busy Coding…

Once funded, sponsors are eager to get started, and the team is eager to show progress. Often, the planning activities in Sprint 0 can be incomplete (or skipped entirely), and the team begins coding with verbal direction from a lead or sponsor with a false assumption that being "Agile" eliminates the need to plan. Inevitably, the team realizes the initial level of effort was underestimated, or development runs into questions or dependencies that were not considered before work started. Now, development is blocked, changed, or needs to be refactored, which negatively impacts all project metrics. To get the team back on track, project resources are diverted from backlog grooming to development support; QA test plans are changed or scrapped, and the team's velocity slows or stops.

In some cases, a hardening sprint can allow time to refactor incompatible architecture or enable proper analysis and refinement of backlog to Ready. However, if Ready cannot be enforced, and coding begins again with unclear requirements and unanswered questions, the effort quickly returns to ad-hoc as the ability to plan work, predict velocity, and schedule timelines becomes impossible.

Refactoring is more easily accommodated with an in-house development team; hardening sprints are an unacceptable alternative for high-end solution consultants whose clients expect them to get it right the first time. This is why Ready is essential for the successful implementation or app dev team.

No Approved UI/UX

From a development standpoint, UI/UX often has fewer dependencies and contingencies than back-end integration efforts; however, that does not mean that UI stories are easier or less complicated.

No one sees great code; everyone sees the color of the button.

Without UI/UX approval, the development team cannot Accept full-stack user stories. Integration efforts are parsed into new stories, which is less efficient for all. Another complications of non-full-stack development is that QA may be unable to completely test functionality and business rules without a UI/UX, and progress cannot be easily demonstrated to non-technical stakeholders for approval and feedback.

Any customer-facing experience is likely to include a significant review cycle. While minor changes to the UI/UX can be made easily at the end of the project, UI/UX approval is essential for Ready for all sprints as even the most minor changes made in-sprint can ripple through the team impacting all roles and planned activities.

While projects differ by complexity and capacity, effective backlog management dictates that UI/UX ideation be at least a half to one sprint ahead of the user story refinement. All Agile projects should endeavor to have one to two sprints of Ready backlog at all times.

No Future State

Similar to the misconception that business requirements are the same as development stories, product owners may not be sufficiently engaged in envisioning future state. "Just make it like the one we have now…." (or just like the mock-up) isn't adequate acceptance criteria for an Agile story. Moreover, it forces the development team to assume the role of the product owner by both mastering current state and then articulating some new-and-improved vision for future state.

Analysis is required in all application development projects regardless of whether or not an analyst or product owner is contributing to the project team. Without skilled resources to refine future-state, developers are certain to mis-assume business needs creating churn, burn-up, and re-work.

Dev-Driven Development

Agile consulting teams involved in custom application development projects seek to fulfill business requirements. Consultants also help to mold and refine the client's vision based on their consulting expertise and industry best practices. If a consultant cannot fulfill the client's original vision, the team offers alternatives. In all cases, fulfilling business needs is the project's top priority. This is not always the case with in-house development efforts.

The "Architect as King," also known as "solution in search of problems," phenomenon is most often found in small to mid-size companies. Contributing to this are things such as the shortage of talent in technical marketplace, the complexity of integrations with the Cloud, third-party vendors, and other human factors. The result is a solution-first environment where the business needs are secondary to the desire, direction, and preferences of the IT group. In these organizations, users are recipients of software, not stakeholders in the development of it; the principal architect or development lead calls the shots: "They'll use what we give 'em…."

Agile methods dictate active involvement of business users in story development.

Because user-centric design and team-solutioning is not a priority in the dev-driven environment, ready backlog doesn't exist because the need to create "user stories" isn't a priority. Architects dictate functionality and approach directly to a developer who works on the task in whatever manner or duration is determined. The backlog functions as a to-do list; grooming and refinement is unnecessary. Final product is approved by the architect who requested the work. Often the code is only tested by the developer who did the work. QA is ad-hoc because there's nothing really to "test against." Work is released into production with little preparation or strategy. Because history has shown that user input is irrelevant to the technology team, users don't bother with complaints or feedback. They accept the bugs and poor design as SOP or create manual work-arounds to avoid using the system.

Agile methods dictate active involvement of business users in story development. In the dev-driven environment, the ability to manage and predict user needs is secondary to maintaining current systems and processes. Because business needs are never truly analyzed, backlog doesn't exist. If Agile's user-centric backlog grooming and refinement techniques were used at the outset, the project would produce a higher quality product for the end-user, and the need for ongoing enhancements and rework would be mitigated.

Real Agile Requires Refined, Ready, and Accepted

Agile's key benefit is the ability to plan and develop work in increments. By planning and developing in small chunks, the team retains the ability to seek insights, mitigate unknowns, and leverage metrics, usability, and other feedback into upcoming work ***before*** that work begins, not after the work is completed. Unfortunately, if the Agile team is not able to plan work thoroughly before it is accepted into sprint, the team's activities become devoted to clarifying and supporting current, in-sprint development. Analysis efforts are not spent planning new

development, features, or leveraging lessons learned.

Not planning work to Ready or permitting change to work that was Accepted results in projects de-volving from Agile to ad-hoc. In-sprint work is continually blocked pending some meeting, some decision, some artifact. When those decisions are made, there is little guarantee the change will fit into the previous timeline. Sprint planning becomes chaotic; eventually, the team completely ignores both acceptance criteria and deadlines because history shows they are neither correct nor enforced.

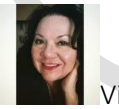
Without standards of Ready, analysts and project management resources are diverted from their intended pre-sprint planning tasks to in-sprint development ***support***. As a result, the overall quality of the product suffers because bona fide analysis, user feedback, and consistent vetting is substituted with ad-hoc, in-sprint "get 'er done" decisions. Stories are reduced to one-line, empty containers, backlogs become little more than to-do lists, and the key benefits of Agile methodology and incremental development are never achieved.

The benefits of Agile development are not achieved just because work is broken into two-week sprints.

Solution and technology consultants who utilize Agile methodologies are keen to enforce standards of Ready because time is the key driver for consulting teams. This is different from captive resources in an internal effort. For in-house teams, time is rarely the key driver. Many internal teams in loose matrices have little control over the flow of work into the team. These groups have a tacit understanding that story tasks are padded to include unplanned, unapproved work and stakeholder churn; sprint deadlines are merely a suggested duration. In these teams, unfinished stories roll into the next sprint. Burn-up or burn-down is untracked, and metrics are collected for gross hours and project duration only.

However, even in teams that struggle with enforcing Ready for backlog, the principles of Refined, Ready, and Accepted can assist the effort by helping to define a reasonable cadence, which assists in long-term duration planning. Moreover, by using the story refinement process and adhering to standards of Ready, teams are likely to see fewer bugs, fewer requests for change, and less technical debt.

About the Author



Vickie Hearne is a writer and technology analyst with more than 20 years of experience as a contractor, independent consultant, and direct employee. Her clients include Fortune 500/100 technology and biomedical companies as well as government organizations. Vickie holds a BA and MA in Political Science, and currently resides in southern California.