What Are You Doing?

By Steve Berczuk

At the center of any project management approach—agile, or otherwise—is the plan: the list of tasks that your team is working on during an iteration. In addition to providing the team a focus, the plan also can help developers identify the goal of a coding episode [1], prioritize their work, and inform design and implementation decisions.

As a team, it is useful to be able identify how well an iteration tracked to the plan, both for retrospection and future planning. Being able to answer questions like "How much coding did we need to do to implement a feature?" and "Did we do much work beyond what we planned?" is useful in a retrospective to improve your process. Having an easy way to identify what work went into implementing a feature helps future planning by make it easier to estimate similar features.

Once the team is coding, version-management systems are at the center of a development team's collaboration process. You can use the version-management system in collaboration with your task-tracking process to help both developers and teams focus on high-value tasks by associating each commit to the version-control system with the work items for an iteration. If done correctly, this can help you work more productively. If done incorrectly, it can slow teams down and even distract from goals.

The Plan
A prerequisite to having a tracking and planning process is to have a plan. A plan need not be heavyweight, and the planning process need not take an undue amount of time. Some developers—and even managers—consider planning a roadblock to actually "getting things done." But planning is an essential part of getting the *right things* done. Planning is simply identifying possible work, prioritizing it, and committing to the group of items to work during a development iteration.

The list of items for an iteration (the backlog) gives the team a focus. While having a plan does not preclude changes, it makes it clearer when there are changes, and enables all of the stakeholders to understand what to expect. The plan, then, is a list of backlog items that identify items of interest to stakeholders.

The plan provides a framework for the development team’s day—from deciding what to work on, to deciding what is worth discussing in the team's daily scrum (if you use Scrum). A useful way to help team members focus on the goal of delivering items off the backlog is to have a rule that, when committing code, you need to associate the commit as being part of an item off of the backlog.

To associate the backlog item with the change, you need a concise way to identify it. This is easy to do when you use an issue tracking system, which automatically assigns identifiers to backlog items. If you use a paper-based tracking system, you might want to have a mnemonic for each work item. The "Print a Report" index card could have the mnemonic PRINTRPT, or if your project has the codename "ROME" and printing was the third task, you can give it the mnemonic ROME-3. Any system works, as long as there is a short code that is easy to identify as an issue ID. Since issue-tracking systems are common, I'll focus on teams that use issue systems that integrate with version-management systems. Issue-tracking systems also can become impediments if used incorrectly (for example, when the tool goal becomes "entering issues" rather than "tracking progress"), so it's important to evaluate the overhead of your planning process.

Tracking Change The simplest way of associating commits with work items is to have every commit have a message that identifies which issue the change implements. For example, "TY-157: Added a new API to allow us to implement the tracking feature." Some tools, such as Mylyn [2], will even add the issue identifier to your default commit message when you commit changes in your Eclipse IDE—but you still have to tell it what issue you are working on.

Adding the issue identifier to your commit messages consistently gives some advantages:

The issue system is an easy way to give more context to the reason for a change, particularly when you are doing many small commits per change (a practice I recommend).

It's easy for anyone trying to do similar work to figure out what work to do.

It encourages people to stay focused and not get off track on other work. (If accomplishing a task leads to an opportunistic refactoring, for example, say that you are doing the change to support the issue.)

If your work environment leads to a lot of work that is off task, you can use the information from the commit logs to make it easier to identify whether additional work is a reason for not meeting sprint commitments.

Adding the identifier to your commit message should not replace other, contextual information you might have added. "Implemented PROJ-123" is less informative than a message like "PROJ-123: added caching." Your team needs to agree on the conventions to follow to balance efficiency on the commit and the review sides of a project.

Adding the issue identifier to the message is still a manual process, and it is useful to make it difficult to commit changes without identifying an issue (or explicitly identifying no issue). If your tools enforce this practice, you can re-enforce the team's awareness of the end goal of its work.

Reporting
Adding issue identifiers to your commit messages can help with maintaining focus, but does not provide much opportunity for feedback and improvement unless you also provide a mechanism for getting the information out.

Many issue-tracking systems, such as Jira [3], have ways of associating commits with issues if you add the issue number to the message. Some CI systems also show which issues were addressed in a given build.

If you don't use a tracking system—or use one that does not make it easy to gather data from the issue system—you can generate some information from tools that allow you to view commit logs. For example, you can write a script that searches through your subversion log history looking for lines that match the pattern of your issue mnemonic, allowing you to generate a list of commits that were associated with an issue.

Risks
People have mixed feelings about a practice like requiring developers to add tracking information into commit messages by hand. Traceability is nice, but having to assign every item to a work item can add overhead and lead to micromanaging. This is certainly a risk, but you can introduce the practice by making clear that the goals are about focus, not supervision.

To that end, you need to agree on a specific granularity of what to track. A Scrum team might start with a product backlog that consists of a small number of coarse-grained stories [4]. At the start of the sprint, the team may divide the stories into tasks. For the purposes of keeping focus, tracking at the story level is appropriate.

Likewise, people may assert that side issues may come up and requiring issues for every commit sounds bureaucratic. You can capture this with placeholder issues, and provide feedback for improving your issue creation process. You may also say that not every change, such as fixing a typo, is relevant to the story. If you did the work in service of the story, it is relevant, so say so.

Implementation Tips
To get the most value and traceability while minimizing overhead at coding time, you want to have an approach that is simple to implement. It should be easy to add this level of traceability to changes, and easy to overcome problems caused by vague story definitions and other planning missteps. You also want to provide an easy way to add data on how the task-level tracking worked so that you can analyze this during the retrospective. This can be accomplished in several ways:

For the purposes of assigning issues to commits, focus on high-level tasks (stories). Try to group the work for an iteration into a small number of stories.

Give each high-level feature in an iteration (story) a mnemonic. If you are using an issue-tracking system, there usually is an issue ID. If you are using a paper-based system (index cards), add a mnemonic to the card.

Add a hook to your version-management system to require that commits have an associated issue. This will make it harder to forget to add the issue number. The hook can be as simple as checking for a string that matches the issue number pattern, or as complex as checking for an actual issue. Simple is adequate.

Have a fallback issue number for developers to use in case a commit does not fit a story cleanly. This will allow work to continue while providing data to analyze about cases where team members wanted to not enter an issue.

Conclusion
Traceability can be onerous, especially when tracking requires extra work on the part of developers. It's possible to automate many things, but it's far easier to tell the system what you are doing. Doing so can help you understand if you are working on the right thing. You can use tools to minimize overhead, but the most important thing is to establish the habit of associating work with something—the plan. Knowing why you are making a code change, and sharing that knowledge with the team, can only be a good thing. The enhanced focus on sprint goals is the main benefit. Traceability for retrospection is a happy side effect.

References
[1] *A Pattern Language of Competitive Development, Part 1*
[2] JIRA
[3] Mylyn
[4] Cohn, Mike (2004). *User Stories Applied : For Agile Software Development*. Boston, MA, Addison-Wesley.

About the Author
Steve Berczuk is an engineer and ScrumMaster at Humedica where he's helping to build next-generation SaaS-based clinical informatics applications. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a certified, practicing ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at steveberczuk.blogspot.com.

StickyMinds.com Weekly Column From 3/14/2011

Close This Window