Improving Quality and Reducing Technical Debt in Agile Projects

By Noopur Davis

Background

Several studies have shown that on average, a developer injects one defect for every ten lines of production code written in a 3GL programming language such as C, C++, and Java [1] [2]. This is understandable when one thinks of the fact that lines of code are handcrafted by human beings (auto-generated code just moves the abstraction level higher). Most of these defects have to be removed before the software can be used by end-users. Software engineering economics have shown that cost of removing these defects grows the longer they stay in the system [3][4]. When the software is targeted for life-critical or safety-critical systems, the necessity of removing these defects becomes not just a matter of economics, but literally a matter of life-or-death.

As Agile development methods have become more and more popular [5], the question arises: how do these methods address early defect removal? Consider Figure 1: it represents traditional software development. Developers work for months until the *Feature Complete* milestone is reached. Testers start testing somewhere around the *Feature Complete* milestone. A mountain of open defects soon exists, and the organization frantically works for weeks and weeks to fix the defects (the find-rate vs. the fix-rate chart becomes the focus of every meeting). Sometime around the *Release Date* milestone, the organization holds its breath and decides to release the software. My friend and mentor Watts Humphrey used to say "software is not released, it escapes"!. A whole bunch of defects are "deferred" to later releases, contributing to the technical debt carried forward.

One promise of Agile development has been in its iterative and incremental nature: if work is done in small increments *and* each increment is of high quality, then we have already reduced the length of time defects stay in the system before they are fixed. In other words, just moving from traditional development (Figure 1) to Agile development (Figure 2) helps reduce the costs of fixing defect, because as noted earlier, we know that the shorter the time between defect injection and removal, the lower the cost.

But we want more than a reduction in cost due to shifting defect removal closer to defect injection: we want to also reduce the number of defects injected in the first place. We want the cumulative area under the curves in Figure 2 to be less than the area under the curve in Figure 1. In addition, we want software to conform to its non-functional requirements (performance, safety, for example), and we want software development processes to be compliant with any applicable internal or external standards, such as FDA compliance.

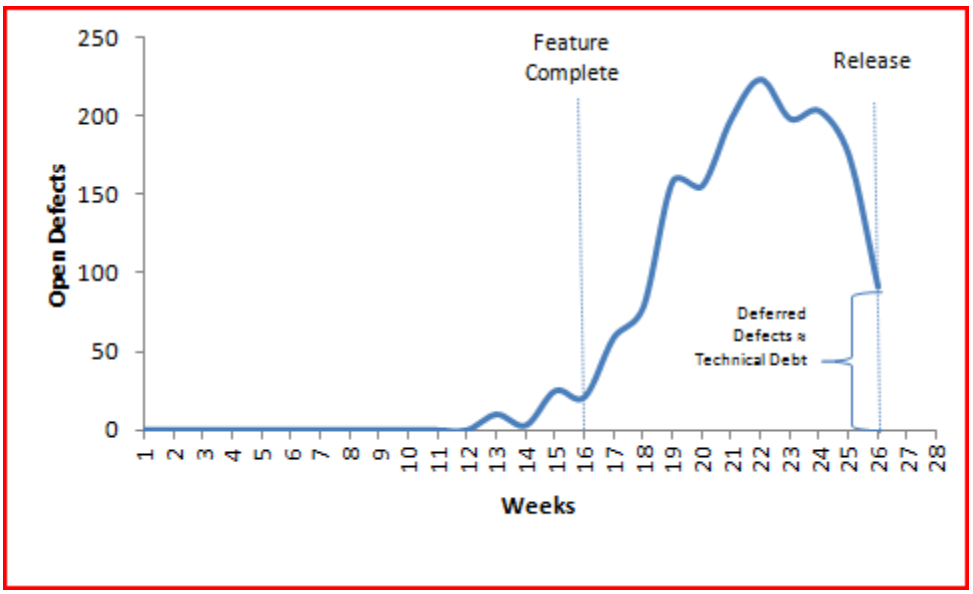

Figure 1: This is not Agile[1]

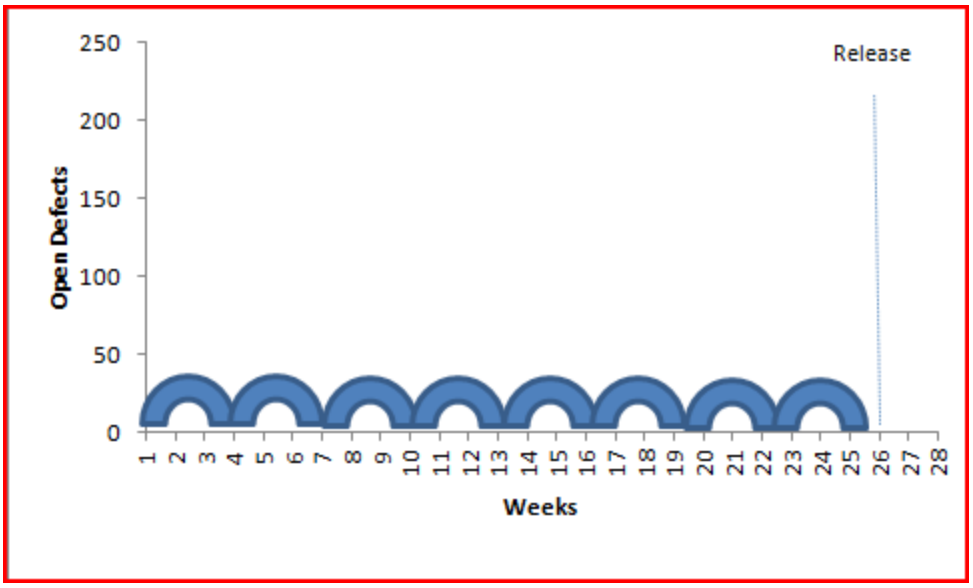

Figure 2: The Promise of Agile

Opportunities to Inspect and Adapt

In Agile development, there are several events used to show demonstrable product progress via working, tested outputs: story completion, sprint completion, and release completion. Each successive event provides additional product capabilities, and opportunities for early feedback. These "inspect and adapt" opportunities allow early "debugging" of plans/processes/products. These are also natural boundaries to inspect and adapt product quality and quality attributes.

Through trial and error, we have discovered that the concept of Definition of Done in Agile is a good way to build quality and compliance into Agile projects.

Definition of Done

According to the Scrum Guide [6], the Definition of Done(DoD) is defined as follows:

> *When the Product Backlog item or an Increment is described as "Done", everyone must understand what "Done" means. Although this varies significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the "Definition of Done" for the Scrum Team and is used to assess when work is complete on the product Increment.*
>
> *The same definition guides the Development Team in knowing how many Product Backlog items it can select during a Sprint Planning Meeting. The purpose of each Sprint is to deliver Increments of potentially releasable functionality that adhere to the Scrum Team's current Definition of "Done."*
>
> *Development Teams deliver an Increment of product functionality every Sprint. This Increment is useable, so a Product Owner may choose to immediately release it. Each Increment is additive to all prior Increments and thoroughly tested, ensuring that all Increments work together.*
>
> *As Scrum Teams mature, it is expected that their Definition of "Done" will expand to include more stringent criteria for higher quality.*

Since Scrum is a framework, no operational guidance is provided on how to implement this. We have taken this concept and applied it to multiple-levels of an Agile project: Story DoD, Sprint DoD, and Release DoD. In other words, we want to remove any ambiguity about what it means for a story to be done, a sprint to be done, or a release to be done: especially any ambiguity related to quality or compliance.

Sample Definition of Done

	Story	Sprint	Release	Measure	Notes
Design					Design completed according to project design standards
Design Peer Review				Design Review Yield	All defects found in design peer review are fixed
Implement	X				Code completed according to project coding standards
Code Peer Review	X			Code Review Yield	All defects found in code peer review are fixed
Unit Test	X			% statement	Goal is 80% statement

				coverage	coverage
Acceptance Test Written	X			% tests automated	Goal is 100% automation
Acceptance Test Peer Review	X			Test Review Yield	
L10N		X	X	Localization Defect Density	
Performance Test		X	X	Compare to Performance Baseline	
System Test		X	X	Defect Density	
Static Analysis		X	X	Defect Density	
Security Review		X	X	Vulnerability Density	

Table 1: Sample Definition of Done

Conclusion

Noopur Davis has used this multi-level implementation of the Definition of Done with dozens of Agile projects she has coached, including scaled agile projects where many teams work cooperatively to deliver large projects. The results have been very encouraging, and will be published in an upcoming article.

References

[1] Humphrey, W. "A Personal Commitment to Software Quality." Pittsburgh, PA: The Software Engineering Institute (SEI) <www.sei.cmu.edu>.

[2] Jones, C. and Bonsignour, O. The Economics of Software Quality. Pearson 2011.

[3] Boehm, B. Software Engineering Economics. Prentice Hall 1981

[4] Rothman, J. "What Does It Cost You To Fix a Defect? And Why Should You Care?" http://www.jrothman.com/2000/10/what-does-it-cost-you-to-fix-a-defect-and-why-should-you-care/

[5] Version One State of Agile Survey 2012 - http://www.versionone.com/state-of-agile-survey-results/

[6] http://www.scrum.org/Scrum-Guides