Driving Quality Improvement and Reducing Technical Debt with the Definition of Done

Noopur Davis
Principal, Davis Systems
Pittsburgh, PA
NDavis@DavisSys.com

***Abstract*—This paper describes our experiences in using the Scrum concept of Definition of Done to drive quality improvements and reduce technical debt. We also describe how the Definition of Done can be a vehicle to implement standards, use checklists, and introduce compliance measures in the Agile development process.**

Keywords: Definition of Done, Agile Quality, Measuring Agile Quality, Reducing Technical Debt

I. INTRODUCTION

Do your projects have to meet corporate quality goals? Does your organization operate in a regulatory environment, such as FDA regulations? Does your project have non-functional requirements such as performance, scalability, security, or safety? Do you want to improve product quality? Or are you just tired of mounting technical debt? This experience report describes how we have used the Definition of Done to address these problems. We faced these issues with many teams we have coached and mentored: through trial-and-error, we have developed a multi-level application of the Definition of Done to address these issues: story-level, iteration level, and release level. We will describe what we did, how we did it, and the qualitative and quantitative results achieved.

II. 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.

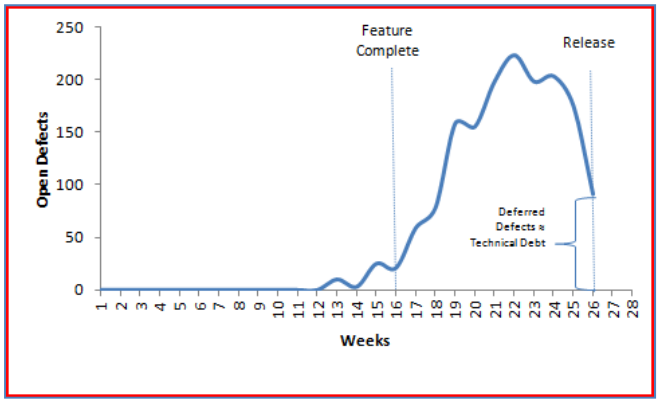

Figure 1: This is not Agile

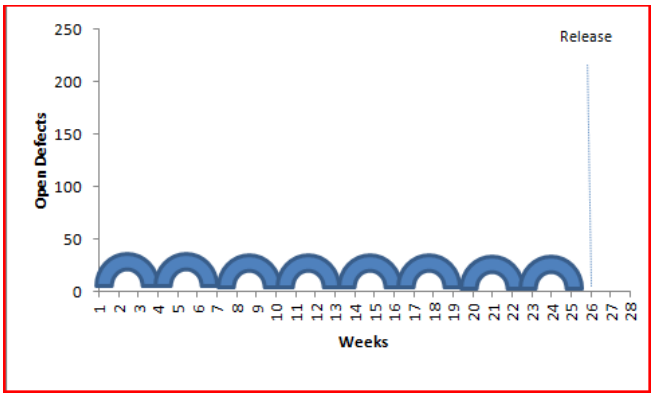

Figure 2: The Promise of Agile

In the past few years, as we trained and coached Agile teams, we often encountered situations where the Agile teams had to meet stringent quality goals (unit test statement coverage >= X%, customer support calls reduced by Y%, deferred defects reduced by X%, etc.), compliance standards (FDA compliance, corporate compliance such as Blackduck assured, Common Criteria assured, statically assured, etc.), non-functional attributes (performance baselines, reliability requirements, etc..)

Our challenge was: how can we build quality in? And how can we do it in a manner consistent with Agile principles of iterative and incremental development, with multiple opportunities to inspect and adapt? We decided to take advantage of these opportunities of inspection and adaption.

III. 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
- Potentially Shippable Increment (PSI) completion (in Scaled Agile Framework, or SAFe[7])
- Release completion

Each successive event provides additional 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, and to integrate compliance and governance.

The Definition of Done (DoD) for each of these events provides a mechanism for doing this.

IV. 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 operationalized it for application at 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.

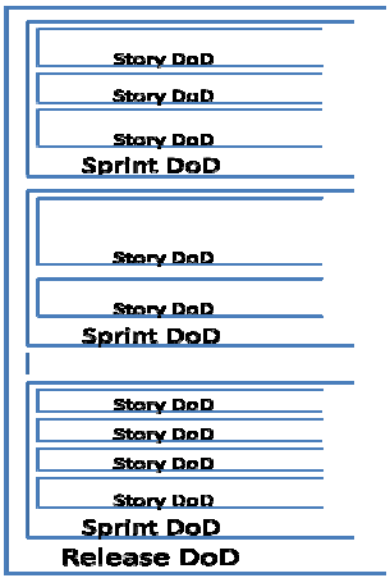

Figure 3: The Definition of Done Fractal

V. Examples of Definition of Done

During Sprint planning, we coach Agile teams to agree upon a Definition of Done. Here, we present the evolution of a DoD for a project as it matured through three releases, each 6-months duration. This team of 5 developers and 3 testers were working on the next release of an existing product in the enterprise software domain. The previous release had been so full of bugs that the team had spent two whole Sprints stabilizing the product before release. Customers were unhappy with product quality and support calls were increasing. The company had set strict quality goals (reduce deferred defects). The product had to conform

to performance baselines (industry benchmarks existed). And the product had to pass the highest level of security audits in the organization.

Table 1 shows the initial DoD the team defined during Sprint 1 planning. Each row represents an activity included in the DoD, while each column shows what completion boundry the activity applied to. An "X" at the intersection of the row and column signifies that the activity in that row applies to the boundary in that column. For example, an "X" in row1-column1 means that the team will not consider a story complete until the code was written according to team coding standards and unit test code coverage was at least at 25%. Note that the items in the DoD DO NOT imply Waterfall: they are criteria for fitness, not meant to be done in any particular order or follow any gates.

The team used this DoD for 6 4-weeks Sprints, and delivered Release 1.

	Story	Sprint	Release	Measure	Notes
Code and Unit Test	X			% statement coverage	Code completed according to project coding standards. Goal for unit test was 25% statement coverage
Code Peer Review	X			Code Review Yield	All defects found in code peer review are fixed
Acceptance Test	X			% tests automated	Goal is 100% automation
Performance Test		X	X	Compare to Performance Baseline	Goal is to not go decrease performance as product is enhanced

Table 1: Release 1 Definition of Done

During Sprint retrospectives, the team found that they had to redo a lot of work because they did not do enough design (UX design as well as engineering design). They also found that Acceptance tests were not enough: too many defects were still being found late and being deferred. So they added design, design peer review (amazing how many issues testers found during design walkthroughs and reviews), acceptance test peer review (developers would often say, "You didn't think about that in the tests", or even better, "I didn't think about that in the code) and exploratory testing to their DoD. Table 2 shows the DoD the team used at the start of Release 2. As you can see, the DoD became more rigorous. This is a pattern we often see: as teams mature, their DoD becomes more rigorous, resulting in higher and higher code quality.

	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
Code and Unit Test	X				Code completed according to project coding standards
Code Peer Review	X			Code Review Yield	All defects found in code peer review are fixed
Acceptance Test Written	X			% tests automated	Goal is 100% automation
Acceptance Test Peer Review	X			Test Review Yield	
Exploratory Test		X		Defect density	The whole team spends 1 day per Sprint in exploratory testing
Performance Test		X	X	Compare to Performance Baseline	

Table 2: Release 2 Definition of Done

During release 2, the team found that although they had improved quality a lot, localization testing revealed a lot defects right before the release, and security audits revealed a lot of defects as well. The team also found a lot of issues when the software was run on production systems. So they incrementally improved DoD during release 3 to address these issues. Table 3 shows the final DoD after Release 3.

	Story	Sprint	Release	Measure	Notes
Design					Design completed according to project design standards
Design Peer Review				Design Review Yield	Use design review checklists All defects found in design peer review are fixed
Code and Unit Test	X				Code completed according to project coding standards
Code Peer Review	X			Code Review Yield	Use code review checklists. All defects found in code peer review are fixed
Acceptance Test Written	X			% Tests automated	Goal is 100% automation
Acceptance Test Peer Review	X			Test Review Yield	Use test review checklists
L10N		X	X	Localization Defect Density	Product tested on non-english operating system. Psuedo-translation of resources each Sprint. Actual translation as each feature completed.
Performance Test		X	X	Compare to Performance Baseline	Baselines and goals set based on current performance and industry benchmarks
System Test		X	X	Defect Density	
Static Analysis		X	X	Defect Density	Using organization rule-set
Security Review		X	X	Vulnerability Density	

Table 3: Release 3 Definition of Done

Working with my friend and colleague Carl Wyrwa, a 38-year veteran of Medical Device software, we have developed a DoD based on FDA IEC 62304 and TIR45 for use my teams we are coaching in the Medical Device software industry [8]. This DoD is presented in Table 4. As you can see, this is much more rigorous than any of the DoDs presented so far.

Activity	Story	Sprint	Release	Project	Notes
Project					
5.1 - SW Development Planning 5.1.7 - SW Risk Management Planning				Yes	*Planning the overall project. Project management, scoping, resourcing. Risk Management Planning.*
5.2 - SW Requirements Analysis 5.2.3 - SW Risk Control Measures				Yes	*High-level, Backlog Management. Translating Design Inputs into high-level definitions of software functionality.*
5.2.4 - Re-Evaluate Medical Device Risk Analysis				Yes	*System-level review when software requirements are established*
5.2.6 - Verify Software Requirements				Yes	*Requirements - Personal Review & Peer Inspection.*
5.3 - SW Architectural Design				Yes	*Major architectural design.*
5.3.5 - Identify segregation for Risk Control				Yes	*Identify segregation of software items essential to risk control*
5.3.6 - Verify Software Architecture				Yes	*Architecture - Personal Review & Peer Inspection.*
Release					
5.1 - SW Development Planning			Yes		*Planning the release. Mid-level planning, integration points of increments, organizing the release into increments.*
5.6 - SW Integration & Integration Testing			Yes		*Complete sets of functionality are integrated together.*
5.7 - SW System Testing & Regression Testing			Yes		*Releases are tested against detailed requirements specifications.*
5.8 - SW Release			Yes		*Formal release to users (internal and/or external).*
Increment/Sprint					
5.1 - SW Development Planning		Yes			*Planning the increment. Integration points of subsystems. Organizing an increment into stories.*
5.6 - SW Integration & Integration Testing		Yes			*Stories integrate with other stories to create a complete set of related functionality.*
5.7 - SW System Testing & Regression Testing		Yes			*Increments are tested against detailed requirements specifications.*
Story					
5.1 - SW Development Planning	Yes				*Individual planning, assigning tasks, execution details for daily tasks, tracking progress.*
5.2 - SW Requirements Analysis	Yes				*Detailed requirements specifications are developed.*
5.3 - SW Architectural Design	Yes				*Emergent – architecture emerges as it needs to in order to support new functionality added by a story.*
5.4 - SW Detailed Design	Yes				*Detailed design specifications are developed for each story.*
5.4.4 - Verify Detailed Design	Yes				*Detailed Design - Personal review & peer inspection.*
5.5 - SW Unit Implementation & Verification	Yes				*Coding. Unit testing.*
5.5.2 - SW Code Verification	Yes				*SW Code - Personal Review & Peer Inspection.*
5.6 - SW Integration & Integration Testing	Yes				*New functionality is integrated into the broader software product as code is developed. Integration is tested as pieces are put together.*
5.7 - SW System Testing	Yes				*Stories are tested against detailed requirements specifications.*

Table 4: Definition of Done Targeted for FDA IEC 62304 and TIR45

OPERATIONAL GUIDANCE

Some early struggles we faced resulted from the fact that teams would agree on a DoD, but then it would become really difficult to track if the team was really using the DoD. To mitigate this problem, we decided to operationalize the use of the multi-level DoD from several points of view.

- Training – we incorporated a DoD exercise during our two-day Scrum and Agile training class. This enabled all team members to ask questions and become socialized to the concept
- Templates – we created templates in Rally and Version One (tools used most frequently by teams we coach for
    - Story DoD templates (used to populate tasks for a story)
    - Sprint DoD templates (used to create one story and associated tasks per Sprint)
    - Release DoD templates (used to create one story and associated tasks per Release)
- Measurement – we started collecting core metrics to monitor the health of the system at Sprint and Release boundaries.

RESULTS

At the end of each Sprint and at the end of each Release, we collected some metrics, which are shown in **Error! Reference source not found.**. Let me explain each measure.

- Defects deferred[1] – this is a measure of technical debt for future releases. Any defect deferred in release X is a technical debt for release X+n. A DoD that results in fewer deferred defects is one that demonstrates reduced contribution to future technical debt. This metric was collected from the Bugzilla data base the project used to track bugs.
- Defects reopened – this team found that initially, 1 in 5 defects that they "fixed" were kicked back. This is just waste. A reduction is percentage of defects re-opened is a measure of elimination of rework or waste. Note: As the team's unit test and acceptance test automation percentage increased, this measure decreased. However, testing the system the team was working on could never be fully automated because of latency and other issues. This metric was collected from the Bugzilla data base the project used to track bugs.
- Peer review yield – this measure captures the percentage of total defects found in peer reviews. This is an indication of the quality of the Peer reviews. The team used Code Collaborator from Smartbear, and this measure was collected from the tool directly.
- Unit test statement coverage – this measure can be used to indicate both code quality (as in defect-free code) and code maintainability (of course, maintainability is a quality attribute in itself). Because this team was starting from almost no unit tests, they agreed that for them, the statement coverage measure was more about maintainability than "free-from-defectsness".
- Customer beta defects – a measure of defects found by customers during customer betas. A reduction in this measure shows improved quality from the customer point of view. This measure was collected from Bugzilla.
- Finally, we wanted to measure the team's satisfaction with the process. At the end of each release (and at the end of random Sprints), we used a Net Promoter Score measure to gauge team satisfaction with their work processes.

Here are the results from the project described earlier as it matured its DoD through three consecutive releases, with each DoD becoming more stringent as the team inspected and adapted. One result that was surprising was how much the team embraced the DoD concept (as shown by the team Net Promoter Score): they were so sick and tired of dealing with poor quality that they were open to trying new things.

Measure	Release 1	Release 2	Release 3	Notes
Defects Deferred	278	136	12	% and count of defects deferred is one measure of technical debt
Defects Reopened	20%	13%	<1%	% and count of defects reopened is one measure of waste
Peer Review Yield	9%	45%	51%	Measure of effectiveness of peer reviews
Unit Test Statement Coverage	~31%	~43%	~75%	Measure of code maintainability
Customer Beta Test Defects				Measure of product quality
Team Net Promoter Score	63%	70%	71%	Measure of team satisfaction with process

Table 5: Results (6-month release cadence)

Based on these results, one can see how this project

- Reduced technical debt – by reducing deferred defects
- Reduced waste – by reducing percentage of defects re-opened
- Improve product quality – by reducing customer beta defects
- Reduced costs – by improving peer review yields
- Improved maintainability – by increasing unit test statement coverage

Although not measured, you can also see that the project used the multi-level DoD to

- Help implement non-functional requirements around performance and security
- Incorporate standards (coding standards) , guidelines(static analysis rule-sets), checklists (review checklists), and metrics (the measures shown above)
- Bridge gaps with other functions such as localization and documentation – localization became a Sprint-level activity instead of a Release-level activity.

Finally, via the FDA-project DoD we can see how the DoD can be used to comply with regulatory requirements.

One note of caution: there is a danger than teams start focusing too much on the DoD. The focus of the team

[1] We did not normalize the defect counts by code size, because this organization released on a 6-month cadence, and the same team worked on all three releases.

should always be on completing stories, on inspection and adaptation, on continuous improvement.

CONCLUSION

We have now used this concept of a multi-level implementation of the Definition of Done with about a dozen of Agile projects, including three Scaled Agile projects involving up to a dozen geographically distributed teams working cooperatively to deliver large projects. The quantitative results have been very encouraging. Just as encouraging has been the way teams have embraced this concept. Although we have been able to empirically show that this method reduces the number of defects and technical debt resulting from deferred defects, we do not yet have enough data to reach solid conclusions about improvements in non-functional requirements. We will continue to gather data from more projects, and will publish future changes to the DoD that result from these analyses.

REFERENCES

[1] Humphrey, W. "A Personal Commitment to Software Quality." Pittsburgh, PA: The Software Engineering Institute (SEI) ftp://ftp.sei.cmu.edu/public/documents/articles/pdf/psp.qual.pdf

[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

[7] http://ScaledAgileFramework.com

[8] Davis, Noopur and Wyrwa, Carl "Medical Device Software: Leveraging Agile and the Team Software Process", Medical Device Summit, May 22, 2013 http://medicaldevicesummit.com/Main/Features1/Medical-Device-Software-Leveraging-Agile-and-the-T-1365.aspx