100 to 1 Ratio for Agile Defect Prevention Over Traditional Methods

© Nancy Van Schooenderwoert, @vanschoo, nancyv@leanagilepartners.com

"There is no hard data that Agile software practices are better – just anecdotes from developers who simply prefer it." This kind of comment is often heard, but hard data ***is*** *actually available. In fact, there is reliable data showing more than a 100-to-1 difference in how well software teams perform defect prevention.*

Capers Jones has reported that an average traditional (that is, non-Agile) software codebase has a defect density of 4.5 defects per function point. For software written in Java, that means over 4 bugs in every 53 lines of code![1] He went on to add that the best traditional teams achieve a defect density of 2.0 defects per function point.

What about Agile teams? When I heard Capers Jones give a presention on this topic, it was immediately clear that I could directly compare my own Agile team's defect density with his software data – just what I had been looking for! His data was based on over 12,000 projects and I had been looking for a way to compare our performance against a legitimate broader measure.

There was just one problem – we hadn't actually been using function points as our measure of software size. All we had were simple static metrics such as lines of non-comment code. "Backfiring" tables help you to convert from lines of code to function points, but it's important to understand the variation inherent doing so. I found that 128 lines of C code were equivalent to 1 function point[2]. Next I calculated the defect density for our GMS (Grain Monitor System) project this way:

Effective software lines of Code (ESLOC) - 29,500 [total non-comment lines]

Total defects in code base = 51 [includes defects found post-delivery]

Lines of C code per Function Point = 128

GMS Function Points = 29500/128 = 230

GMS Defects per Function Point = 51/230 = 0.22

Therefore our project's average defect density was 0.22 defects per function point.

[1] Tables that associate lines of code with function points can be found at http://www.qsm.com/resources/function-point-languages-table and they give ranges of values, because function points is really a complexity measure.

[2] This figure came from a lines-of-code-to-function-points table that is no longer online. I referenced it in the paper "Embedded Agile Project by the Numbers With Newbies" and have seen the same figure of 128 lines of C code given in another source at http://www.cs.helsinki.fi/u/taina/ohtu/fp.html

That's nearly ten times better than what Capers Jones reported for the best of the traditional software teams (at 2.0 defects/fp)! Could it be right? Another lookup table gave 97 as the average number of lines of C in one function point. Plugging that into the calculations gives:

GMS Function Points = 29500/97 = 304
GMS Defects per Function Point = 51/304 = 0.168

This other calculation makes our project's defect density more than 11 times better than the best traditional teams.

Separately from the Capers Jones data, I found a description of two more Agile software teams that was complete enough for me to calculate their defect densities in the same way.

Team	Defects per Function Point	Approach
Follett Software[1]	0.0128	Agile
BMC Software[1]	0.048	Agile
GMS[2]	0.22	Agile
Industry Best[3]	2.0	Traditional
Industry average[3]	4.5	Traditional

1. Computed from data reported in "How Agile Projects Measure Up, and What This Means to You" by Michael Mah, Cutter IT Journal, Vol. 9, No. 9 (Sept 2008), page 10
2. Van Schooenderwoert, "Embedded Agile Project by the Numbers With Newbies" paper presented at Agile 2006. Available at http://www.leanagilepartners.com/publications.html
3. Capers Jones presentation for Boston SPIN, Oct., 2002. Available at http://www.boston-spin.org/talks.html#yr2002

Table 1. Defect densities for Agile and traditional teams

In the Cutter IT Journal, Michael Mah published an article where he gave lines-of-code and defect numbers for two Agile teams that he studied. One of these, the Follett team, achieved 0.0128 defects per function point – easily another order of magnitude beyond my GMS team! The other team he reported on, BMC, also exceeded our result though not by as much. Here is what those calculations look like:

ESLOC = 500,000
Total defects = 121
Lines of Java code per Function Point = 53
Follett Function Points = 500000/53 = 9434
Follett Defects per Function Point = 121/9434 = 0.1283

ESLOC = 700,000

Total defects = 635
Lines of Java code per Function Point = 53[3]
BMC Function Points = 700000/53 = 13208
BMC Defects per Function Point = 635/13208 = 0.048

The BMC team's defect density was 40 times smaller than the best traditional teams, and they were a large 92-person distributed team using Scrum. The Follett team was co-located, with a peak staff of 25 people using XP. Their defect density of 0.128 was 156 times smaller than the best traditional teams!

What do these numbers mean? They are the visible sign of problems *avoided*. Low defect densities mean the team is preventing defects, or catching most of them with less-than-usual effort expended. This is in stark contrast to teams that have high defect densities and must delay releases while they clean up the bugs. All that extra work is pure waste which Agile teams avoid by early and frequent testing built right into every stage of the work.

What's new is seeing this waste quantified: A 100-to-1 improvement can't be achieved by just pushing harder, or working late. No offshore team gets paid 1/100th of the on-shore labor rate. This is a sign of a systemic improvement that really works.

Before moving to Agile, software teams generally are devoting a third to half of their time to rework due to defects. Therefore, a team can approach doubling of their output through Agile defect prevention practices alone. The implications for safety-critical work are even more significant.

In safety-critical applications, hard-to-test scenarios that were checked by analysis in traditional teams are now tested in Agile teams by exercising the production software within a 'test runner' harness. Test runners are a tool used by developers to test the software as it's being written, and they also execute regression tests automatically – usually many times per day. The result is that many more defects are caught early, and hazard mitigations can also be repeatedly checked to ensure they have not been inadvertently weakened.

Agile design, development, and test practices are a real breakthrough that ends the need to choose between quality and speed in software development.

[3] The table at http://www.qsm.com/resources/function-point-languages-table gives 53 as both the average and the median lines of Java per function point.