Software Defect Removal Efficiency

By

Capers Jones, President
Capers Jones & Associates LLC
Email: CJonesiii@cs.com

Abstract

The most important contributor to the quality of software-intensive systems is the quality of the software components. The most important single metric for software quality is that of defect removal efficiency (DRE). The DRE metric measures the percentage of bugs or defects found and removed prior to delivery of the software. The current U.S. average in 2011 is only about 85% of total defects removed. However, best in class projects can top 99% in defect removal efficiency. High levels of DRE cannot be achieved using testing alone. Pre-test inspections and static analysis are necessary to top 95% in defect removal efficiency.

Introduction

In the 1970's the author worked for IBM. Software applications were growing larger and more complex so quality was becoming a serious issue. IBM began a careful analysis of software quality. Measurements were taken of defects found in software requirements, design documents, source code, user manuals, and also "bad fixes" or secondary defects accidentally included in defect repairs.

At the same time IBM developed the function point metric, because it was necessary to analyze non-coding defects and non-coding development activities as well. After several years of data collection, it was possible to determine the relative contribution of various defect origins on total software defects. The total number of defects from all five sources was termed the "defect potential" of a software application.

Table 1 shows approximate U.S. averages from more than 13,000 projects. Table 1 shows the average volumes of defects found on software projects, and the average percentage of defects removed prior to delivery to customers:

Table 1: Defect Removal Efficiency by Origin of Defects Circa 2011 (Data Expressed in Terms of Defects per Function Point)

Defect Origins	Defect Potentials	Removal Efficiency	Delivered Defects
Requirements	1.00	77%	0.23

Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Document	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
Total	*5.00*	*85%*	*0.75*

Table 1 is an excerpt from the author's book The Economics of Software Quality, Addison Wesley, 2011.

There are of course fairly wide ranges. The maximum defect potential observed for large applications of 10,000 function points is about 7.0 defects per function point. The minimum number of defects observed for small projects below 1000 function points is about 2.00 per function point. The maximum defect removal efficiency observed is about 99% and the lowest is below 80%.

Both defect prevention and defect removal are important, but this article concentrates on defect removal efficiency because it is a critical metric and fairly easy to measure.

Measuring Defect Removal Efficiency (DRE)

Serious software quality control involves measurement of defect removal efficiency (DRE). Defect removal efficiency is the percentage of defects found and repaired prior to release.

In principle the measurement of DRE is simple. Keep records of all defects found during development. After a fixed period of 90 days, add customer-reported defects to internal defects and calculate the efficiency of internal removal. If the development team found 90 defects and customers reported 10 defects, then DRE is of course 90%.

(Note that the International Software Benchmark Standards Group (ISBSG) uses release plus 30 days for DRE measures. This means that ISBSG DRE measures are higher than the author's due to the 30-day versus 90-day intervals.)

In real life DRE measures are tricky because of bad-fix injections, defects found internally after release; defects inherited from prior releases; invalid defects; and other complicating factors.

Raising Defect Removal Efficiency (DRE) Levels

Most forms of testing are less than 50% efficient in finding bugs or defects. However, formal design and code inspections are more than 65% efficient in finding bugs or defects and often top 85%.

Static analysis is also high in efficiency against many kinds of coding defects. Therefore all leading projects in leading companies utilize formal inspections, static analysis, and formal testing. This combination is the only known way of achieving cumulative defect removal levels higher than 95% and approaching or exceeding 99%.

Table 2 illustrates the measured ranges of defect removal efficiency levels for a variety of reviews, inspections, static analysis, and several kinds of test stages.

Table 2: Pre-Test and Test Defect Removal Efficiency Ranges

Pre-Test Defect Removal	**Minimum**	**Average**	**Maximum**
Formal design inspections	65.00%	87.00%	97.00%
Formal code inspections	60.00%	85.00%	96.00%
Static analysis	65.00%	85.00%	95.00%
Formal requirement inspections	50.00%	78.00%	90.00%
Pair programming	40.00%	55.00%	65.00%
Informal peer reviews	35.00%	50.00%	60.00%
Desk checking	25.00%	45.00%	55.00%
Average	*48.57%*	*69.29%*	*79.71%*
Test Defect Removal	**Minimum**	**Average**	**Maximum**
Experiment-based testing	60.00%	75.00%	85.00%
Risk-based testing	55.00%	70.00%	80.00%
Security testing	50.00%	65.00%	80.00%
Subroutine testing	27.00%	45.00%	60.00%
System testing	27.00%	42.00%	55.00%
External Beta testing	30.00%	40.00%	50.00%
Performance testing	30.00%	40.00%	45.00%
Supply-chain testing	20.00%	40.00%	47.00%
Cloud testing	25.00%	40.00%	55.00%
Function testing	33.00%	40.00%	55.00%
Unit testing (automated)	20.00%	40.00%	50.00%
Unit testing (manual)	15.00%	38.00%	50.00%
Regression testing	35.00%	35.00%	45.00%
Independent verification	20.00%	35.00%	47.00%
Clean-room testing	20.00%	35.00%	50.00%
Acceptance testing	15.00%	35.00%	40.00%
Independent testing	15.00%	35.00%	42.00%
Average	*29.24%*	*44.12%*	*55.06%*

The low defect removal efficiency levels of most forms of testing explain why the best projects do not rely upon testing alone. The best projects utilize formal inspections first, static analysis, of code, code inspections for key features, and then a multi-stage testing sequence afterwards. This combination of inspections followed by static analysis and testing leads DRE in the range of 95% to 99%/ It also leads to the shortest overall development schedules, and lowers the probabilities of project failures.

Low Quality Defect Removal Efficiency (DRE) Case Study

Table 3 is a simple case study that illustrates the typical results of four common forms of testing: 1) Unit test; 2) Function test; 3) Regression test; 4) System test. Since testing is not very efficient, the results are not very good. We will also assume a traditional “waterfall” development method.

In this case study let us assume an application of 1,000 function points in size. Let us also assume a defect potential of 5.0 defects per function points. This means that total probable defects in the application will be 5,000. We will also assume that 7% of

defect repairs result in “bad fixes” or new defects. Table 3 illustrates a common pattern of fairly low defect removal efficiency:

Table 3: Low Quality Defect Removal Efficiency (DRE) Example

Size (function points) =		1,000
Defect potential per function point =		5.00
Defects in application =		5,000
Bad-fix injection =		7.00%
	Defect Removal Efficiency	**Defect Removal Pattern**
Unit test	38%	
Defects found		1,900
Bad fixes		133
Defects remaining		2,967
Function test	40%	
Defects found		1,187
Bad fixes		83
Defects remaining		1,780
Regression test	35%	
Defects found		623
Bad fixes		44
Defects remaining		1,114
System test	42%	468
Defects found		33
Bad fixes		613
Defects remaining		
TOTAL DEFECTS REMOVED		4,178
TOTAL BAD FIXES		292
TOTAL DEFECTS DELIVERED		613
HIGH-SEVERITY DEFECTS DELIVERED		110
DEFECT REMOVAL EFFICIENCY (DRE)		85.32%
DELIVERED DEFCTS PER FUNCTION POINT		0.61

The case study in table 3 achieved only 85.32% in cumulative defect removal efficiency prior to delivery. This is because testing with no prior inspections or prior static analysis is not usually sufficient to achieve high levels of defect removal efficiency.

Table 3 is something of a professional embarrassment. No true engineering discipline should deliver a product with only about 85% of known defects removed. But such results are the norm for software applications.

High Quality Defect Removal Efficiency (DRE) Case Study

Because the example in table 3 was professionally embarrassing, let us see what happens when formal inspections are used prior to testing. Let us also assume the use of one of the more effective software development methods, Watts Humphrey's Team Software Process (TSP). With both TSP and inspections in use, these advantages occur:

1. Defect potentials are reduced.
2. Defect removal efficiency levels are higher.
3. Bad fix injections are reduced.

In this second case study let us assume the same application size of 1,000 function points. However let us also assume a defect potential of 4.5 defects per function points due to TSP. This means that total probable defects in the application will be 4,500. We will also assume that only 3.5% of defect repairs result in "bad fixes" or new defects as opposed to 7% in the prior example.

Table 4 illustrates the results of the new scenario which combines both an effective development method with a more efficient defect removal pattern:

Table 4: High Quality Defect Removal Efficiency (DRE) Example

Size (function points) =		1,000
Defect potential per function point =		4.50
Defects in application =		4,500
Bad-fix injection =		3.50%
	Defect Removal Efficiency	**Defect Removal Pattern**
Formal Inspections (Design, Code)	85%	
Defects found		3,825
Bad fixes		134
Defects remaining		809
Unit test	42%	
Defects found		340
Bad fixes		12
Defects remaining		457
Function test	45%	
Defects found		206
Bad fixes		7
Defects remaining		251
Regression test	40%	
Defects found		101
Bad fixes		4
Defects remaining		147
System test	47%	
Defects found		69
Bad fixes		2

Defects remaining	76
TOTALDEFECTS REMOVED	4,540
TOTAL BAD FIXES	25
TOTAL DEFECTS DELIVERED	76
HIGH-SEVERITY DEFECTS DELIVERED	14
DEFECT REMOVAL EFFICIENCY (DRE)	98.33%
DELIVERED DEFCTS PER FUNCTION POINT	0.08

When the results of Table 3 are compared with the results of Table 4, we can see that defect removal efficiency levels have climbed from an embarrassing 85.32% up to a respectable 98.33%.

Not only were inspections very efficient in finding defects, but the combination of inspections plus the formal Team Software Process also raised the efficiency level of each test stage.

Removing 100% of software defects is almost impossible, but achieving defect removal efficiency levels that are higher than 95% should be a minimum professional requirement. In fact such levels of defect removal efficiency should probably be included in software outsource contracts.

Summary and Conclusions

This article illustrates only four test stages plus formal inspections of design and code. Some large systems use inspections of requirements, design, code, and test materials. They also use static analysis tools prior to testing. In addition they may use as many as a dozen test stages rather than the four shown here. This article is intended to explain the basic principles of defect removal efficiency (DRE) but it does not cover every possible combination and permutation.

Complete elimination of software defects is beyond the current state of the art. However elevating levels of defect removal efficiency from today's average of 85% up to more than 95% can easily be achieved. It is only necessary to use a synergistic combination of pre-test inspections, static analysis, and formal testing. But it is also necessary to measure defect removal efficiency (DRE).

Measuring defect removal efficiency (DRE) measurement and topping 95% in cumulative DRE are the signs of a top software production group. Companies that do not measure DRE are usually well below 85% when the author has been called in for an external quality benchmark study.

References and Readings

Gack, Gary; Managing the Black Hole: The Executives Guide to Software Project Risk; Business Expert Publishing, Thomson, GA; 2010; ISBN10: 1-935602-01-9.

Galorath, Dan; Software Sizing, Estimating, and Risk Management: When Performance is Measured Performance Improves; Auerbach Publishing, Philadelphia; 2006; ISBN 10: 0849335930; 576 pages.

Garmus, David and Herron, David; Function Point Analysis – Measurement Practices for Successful Software Projects; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3;363 pages.

Gilb, Tom and Graham, Dorothy; Software Inspections; Addison Wesley, Reading, MA; 1993; ISBN 10: 0201631814.

Humphrey, Watts, TSP – Leading a Development Team; Addison Wesley, Boston, MA; ISBN o-321-34962-8; 2006; 307 pages.

Jones, Capers and Bonsignour, Olivier; The Economics of Software Quality; Addison Wesley, Boston; 2011; ISBN 10: 0-13-258220-1;587 pages.

Jones, Capers; Software Engineering Best Practices; McGraw Hill, New York, 2010; ISBN 978-0-07-162161-8; 660 pages.

Jones, Capers; Applied Software Measurement; McGraw Hill, 3rd edition 2008; ISBN 978-0-07-150244-3; 668 pages; 3rd edition (March 2008).

Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 2007; ISBN 13-978-0-07-148300-1.

Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA; ISBN 0-201-48542-7; 2000; 657 pages.

Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.

Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

Radice, Ronald A.; High Quality Low Cost Software Inspections; Paradoxicon Publishing Andover, MA; ISBN 0-9645913-1-6; 2002; 479 pages.

Wiegers, Karl E.; Peer Reviews in Software – A Practical Guide; Addison Wesley Longman, Boston, MA; ISBN 0-201-73485-0; 2002; 232 pages.