An Economic Analysis of Software Defect Removal Methods

By Gary A. Gack

Synopsis

This paper, based on my forthcoming book ***Managing the "Black Hole": The Executive's Guide to Software Project Risk***, explores the economic consequences of alternative strategies for software defect detection and correction during the software development life cycle. Most published analyses have relied on "cost per defect" as an index for comparison of alternatives. This metric has been shown to be fundamentally flawed as it fails to differentiate between fixed and variable costs and because it effectively penalizes high quality software. Cost per defect necessarily increases as incoming quality improves, regardless of defect detection strategies employed.

Most published papers on this topic include many unstated assumptions. This paper employs a set of parametric models that explicitly state the assumptions incorporated in the models. I invite readers to substitute alternative parameter values to explore consequences of different assumptions or experience. A hypothetical "case study" is presented to illustrate the impact of different assumptions. These models are available from the author on request, subject only to a good faith agreement to share assumptions and conclusions.

Objectives

This paper and the models it uses are intended to help software organizations determine an optimal combination of defect containment methods and resource allocation. These models provide a mechanism to forecast the consequences of alternative assumptions and strategies in terms of both cost (in person months) and delivered quality (measured by "Total Containment Effectiveness" – TCE – i.e., the estimated percentage of defects removed before software release to customers).

Caution

As George Box said, "All models are wrong – some are useful." *This paper uses model parameters taken from a variety of public sources, but makes no claim that these parameter values are valid or correct in any particular situation.* It is hoped the reader will take away a thought process and perhaps make use of this or similar models using parameter values appropriate and realistic in the intended context. It is widely recognized that all benchmark values are subject to wide (and typically unstated) variation. Many parameter values will change significantly as a function of project size, application domain and other factors.

Overview

The complete model includes 5 tables – the first 4 include user-supplied parameters and calculate certain fields based on those parameters. The fifth table summarizes the results of the other four. We will look at the summary first, and then get into the details upon which the summary is based.

The summary charts, displayed graphically below, include five scenarios all based on an assumed size of 1000 function points. The first three scenarios assume defects are "inserted" at US average rates according to Capers Jones ***Software Engineering Best Practices*** (2009, p.69) – a total of 5.00 defects per function point, including bad fixes and documentation errors. Scenarios four and five respectively reflect results reported by higher maturity groups in which defects inserted are reduce to 4 per function point in scenario 4 and 3 per function point in scenario 5.

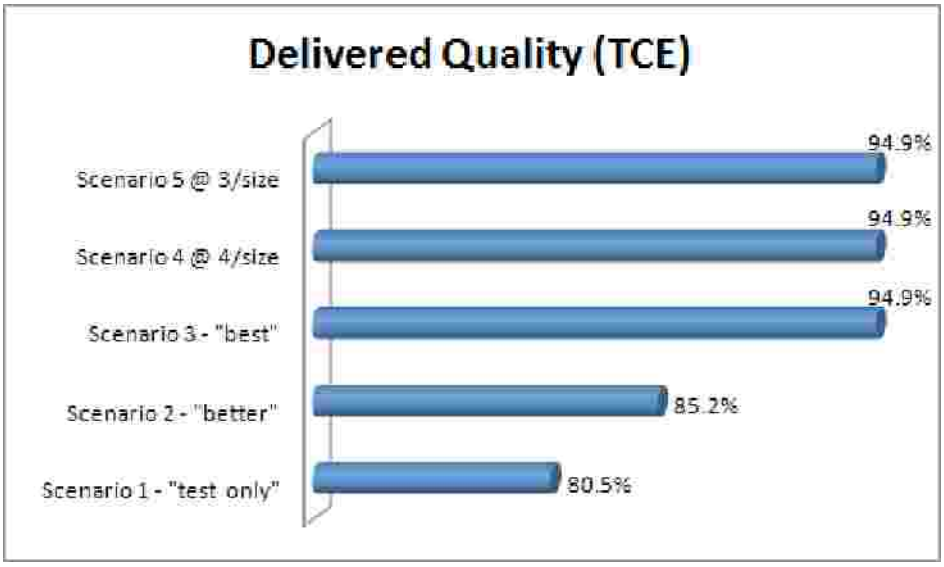
Scenario 1 represents a "test only" scenario in which no "pre-test" appraisals, such as formal inspections, are used. Scenarios two and three introduce certain pre-test appraisals, including inspections and static analysis, and in scenarios 3-5 discontinue some test activities. Other model parameters, discussed later, remain constant across all scenarios. Inspection percentages indicate the portion of the work product inspected.

Alternative Appraisal Strategies

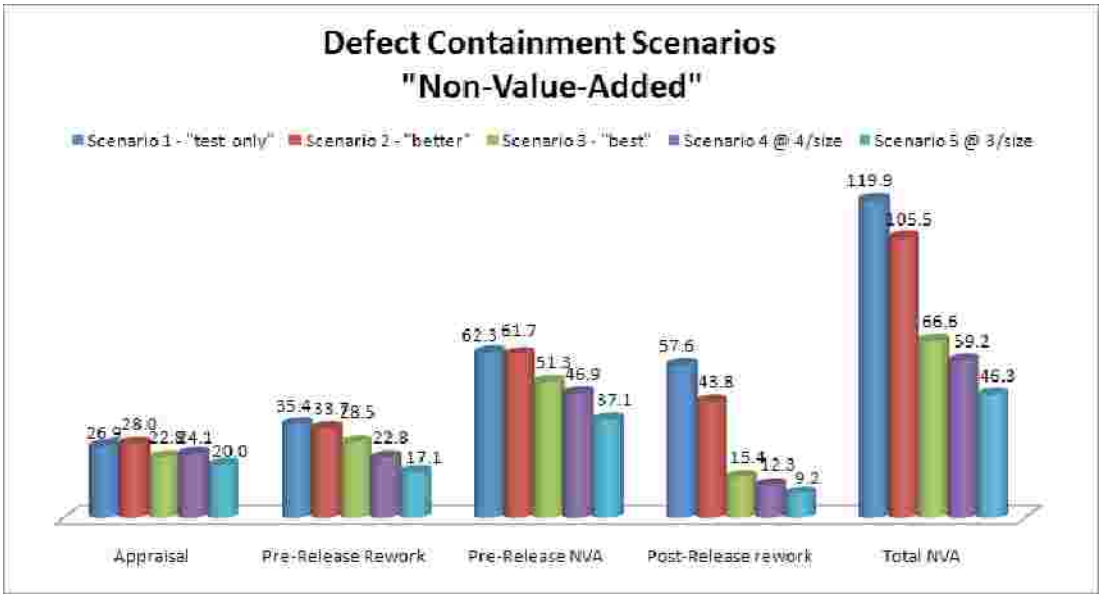
Appraisal Type	Scenario				
	1	2	3	4	5
Requirements Inspection		50%	75%	75%	75%
Design Inspection		30%	50%	50%	50%
Code Inspection		20%	20%	20%	20%
Static Analysis			x	x	x
Unit Test	x	x			
Function Test	x	x			
Integration Test	x	x	x	x	x
System Test	x	x	x	x	x
Acceptance Test	x	x	x	x	x

Note that static analysis can only be used for certain procedural languages such as C and Java.

Two summaries are developed – the first shows the impact of alternative appraisal strategies on delivered quality as measured by "Total Containment Effectiveness", i.e., the percentage of defects removed prior to delivery of the software. In this illustration a "best" mix of appraisal activities (scenarios 3-5) reduces delivered defects by about 75% compared to a test-only approach (scenario 1) typically used by average groups.



The second summary shows the impact of alternative appraisal strategies on "non-value-added" effort as defined in the Cost of Quality framework – i.e., all appraisal and rework effort is by definition "non-value-added" (NVA). Although certainly a "necessary evil" our goal will always be to minimize these costs.



In this illustration a "best" (scenario 3) mix of appraisal activities reduces total NVA effort (including both pre- and post-release effort) by 44% compared to a test-only approach typically used by average groups (66.6 person months in scenario 3 vs. 119.9 in scenario 1). More mature organizations, as a result of lower defect insertion, can reduce NVA by an additional 30% (to 46.3 in scenario 5 vs. 66.6 in scenario 3).

Supporting Details

As indicated earlier, 4 tables containing various parameters are used to develop the summary forecast.

- Table 1: The Defect Containment Model – identifies appraisal steps to be used in a given scenario, a forecast of the number of major defects likely to be present at the start of each appraisal step, and the "containment rate" (expected percentage of defects likely to be found) for each step. The model forecasts total major defects found and remaining at each appraisal step based on the parameters provided.
- Table 2: Inspection Effort Model – parameters define the percent of the total work product to be inspected (often less than 100%), the number of major defects expected to be found by each inspection type, and the estimated rework hours per defect per inspection type. Forecasts of number of inspections, inspection effort, and rework effort for each step are derived from the parameters provided and from data derived in table 1.
- Table 3: Static Analysis Effort Model – required parameters include a forecast of effort hours per size and rework hours per major defect. Total analysis effort and rework effort are calculated.
- Table 4: Test Effort Model – parameters include effort per size for each test step, a factor that reflects the impact of pre-test appraisals (if any), and rework hours per major defect for each test step. Total test execution and rework effort are calculated for each test step.

Each of these tables is described below and the sources of the parameter values used are identified. Values used in this example are for Scenario 3. *These parameter values may or may not be appropriate to a particular context.* I encourage readers to share any data and/or opinions they may have regarding these parameter values. A summary of feedback I receive will be available to participating parties.

Table 1: The Defect Containment Model

Appraisal Step		Forecast			
Tag	Description	DI_n - Potential Major Defects/Size "Inserted" (SEBP09 p.69)	ACE %	DF_n - Major Defects Found	DR_n - Major Defects Remaining
ACE_1	e.g., Requirements Inspections	1.150	60%	690	508
ACE_2	e.g., Design Inspections	1.438	60%	1168	860
ACE_3	e.g., Code Inspections	2.013	60%	1724	1270
ACE_4	e.g., Static Analysis (removes code defects only)		50%	635	679
ACE_5	e.g., Unit Test		0%	0	679
ACE_6	e.g., Function Test		0%	0	679
ACE_7	e.g., Integration Test		30%	204	490
ACE_8	e.g., System Test		35%	171	330
ACE_9	e.g., Acceptance Test		25%	83	254
			Total Containment Effectiveness		94.9%

Column 1 - Each row is "tagged" by an identifier in the form "ACE_n" where ACE stands for "Appraisal Containment Effectiveness" and the subscript uniquely identifies a specific appraisal step or type.
Column 2 - Appraisal steps to be included in any given scenario are named. I have included a default list of commonly used appraisal steps, but these may be changed as appropriate to any particular situation. Additional rows might be added if required.
Column 3 – Defects per size "inserted". The values used here are based on Jones2009[1], p.69. I have adjusted his values by allocating documentation defects proportionately to requirements, design, and code.
Column 4 – ACE % is an estimate of the percentage of incoming defects likely to be found by the indicated appraisal step. Jones (ibid, p.125) reports containment rates for inspections range from 65% to 85%. I have elected to use a more conservative 30% in scenario 2 and 60% in scenarios 3-5 for inspections. I have assumed a conservative 50% containment rate for static analysis – Jones (ibid, p.615) indicates 87% - note that static analysis applies only to code (it does not find requirements or design defects) and also does not generally find functional defects – most defects found by static analysis are "technical" in nature – memory leaks, null pointers, etc. Test step containment rates

are based on Jones2007[2] p.488 (Outsource). 0% indicates an appraisal type is not used.
Column 5 – defects found = Major defects present (cumulative) * column 4, ACE%
Column 6 - Defects remaining (present less found, cumulative) are increased by 7% to reflect bad fixes at each step.

This example (scenario 3 "best") inspects requirements, design, and code, employs static analysis, and does not use unit or function tests. As indicated this strategy leads to 94.9% TCE.

Table 2: Inspection Effort Model

Appraisal Step		Forecast						
Tag	Description	% Inspected	Majors per Inspection	# Inspections	Inspection person months	Rework hours per major defect	Rework person months	Total person months
ACE_1	e.g., Requirements Inspections	75%	18.4	21	2.8	0.500	2.6	5.4
ACE_2	e.g., Design Inspections	50%	18.4	23	3.2	0.750	6.6	9.8
ACE_3	e.g., Code Inspections	20%	3.75	68	9.2	1.000	13.1	22.3
			Totals	112	15.2		22.3	37.6

Columns 1 and 2 are the same as in table 1.
Column 3 specifies the percent of the total work product inspected. My experience suggests wide variation, and I do not believe any particular benchmark is relevant. This value is a local policy decision based on product risk characteristics and other factors.
Column 4, Major Defects per Inspection, indicates an expected average number of major defects likely to be found by a single inspection event conducted in accordance with IEEE Std. 1028-2008. The values used in this illustration are based on a collection of approximately 1500 individual inspection records complied by my colleagues and I at SoftwareInspection.org. These records indicate actual results achieved across several dozen different software groups in a range of industries. Where local data exist they should certainly be used.
Column 5, number of inspections, is calculated as follows:

> # Inspections = [# Major defects Found (from table 1) / column 4 (majors per inspection)] * column 3 (% inspected) – the logic of this calculation rests on the assumption that we can intelligently select those portions of the work product most likely to contain major defects and will not need to inspect 100% of the product to find the number of defects forecast. It is generally accepted that "defect clustering" occurs – defects are not evenly distributed across the work

product. *Caution – this logic can be misleading if an aggressive containment rate is used in table 1.*

Column 6, Inspection person months, is calculated as [number of inspections * 18 hours per inspection] (consistent with IEEE 1028-2008, excluding rework calculated separately), divided by 132 hours per month to arrive at person months of inspection effort.

Column 7, Rework hours per major defect, an additional parameter, is used to calculate total rework in person months. I have been unable to find satisfactory benchmark values for rework of defects found by inspections. Hence I use my own "guesstimates" based on experience. Jones and others have shown the classic "1-10-100" ratio of increasing defect costs is clearly not correct. In my experience cost to fix a defect does increase through the life cycle, but only slowly. Again, these parameters are subject to wide variation and are best determined locally.

This scenario indicates the indicated sequence of three types of inspections will require 15.8 person months to conduct a total of 116 inspections and an additional 16.7 person months of effort to correct the 3,582 defects forecast to be found by inspections (from table 1, column 5, $ACE_1 – ACE_3$). Inspection effort may be regarded as a "fixed" cost in that once a decision has been made to conduct the inspections the necessary effort will be incurred regardless of the number of defects actually found. Rework effort to correct defects actually found is a variable cost.

Table 3: Static Analysis Effort Model

Appraisal Step		Forecast						
Tag	Description	Hours per Size			Analysis Person Months	Rework hours per major defect	Rework person months	Total person months
ACE_4	e.g., Static Analysis (removes code defects only)	0.1			0.8	0.250	1.2	2.0
				Totals	**0.8**		**1.2**	**2.0**

Columns 1 and 2 are as in table 1.

Column 3 is a parameter that defines effort required to perform static analysis in hours per size. The value indicated here, .1 hours per function point, is based on personal correspondence with Capers Jones. Some static analysis tools run in real-time as code is entered and will suggest corrections for certain types of defects – hence, time to "run" the analysis is minimal. I have used .1 hours (6 minutes per function point) as a very rough approximation of the overhead likely to occur.

Columns 4 and 5 are not used

Column 6 calculates person month required to run static analysis and analyze the results. = (column 3 * size) / 132 hours per month – .8 person months in this example

Column 7 is a parameter value for time per major defect to correct defects found. The value used is also based on correspondence with Capers – he indicates static analysis tools will suggest corrections for approximately 85% of defects identified so fix time for those is negligible. He also estimates the other 15% will on average require 1.5 hours each. Accordingly I have assumed an average fix effort of .25 hours per defect.
Column 8 calculates rework person months = [major defects found (from table 1) * column 7] / 132.

Table 4: Test Effort Model

Appraisal Step		Forecast						
Tag	Description	Test hours per Size	Pre-test Impact Factor		Test Execution Person Months	Rework hours per major defect	Rework person months	Total person months
ACE_5	e.g., Unit Test	0.880	1.000		0.0	1.000	0.0	0.0
ACE_6	e.g., Function Test	0.880	1.000		0.0	1.000	0.0	0.0
ACE_7	e.g., Integration Test	0.750	0.500		2.8	1.250	1.9	4.8
ACE_8	e.g., System Test	0.660	0.500		2.5	1.500	1.9	4.4
ACE_9	e.g., Acceptance Test	0.380	0.500		1.4	1.750	1.1	2.5
				Pre-Release Total	6.8		5.0	11.8
				Post-Release Rework		8.0	15.4	15.4
				Project Totals	6.8		20.3	27.1

Columns 1 and 2 are as in table 1.
Column 3 specifies test hours per size. Values included here are based on Jones2009 (ibid, p.264, mode values)
Column 4 is a “Pre-Test Impact Factor”. This parameter is used to estimate the impact that will result if pre-test appraisals are in fact used. When pre-test methods are used there are several important consequences that impact test effort:

- The number of defects coming into any given test step will necessarily be significantly fewer.
- Hence, fewer defects will be found, and less rework will be needed to correct those defects. “Variable” cost will go down.
- Fewer defects incoming means fewer tests will fail. When a test fails it will often prevent other tests from being executed – they are said to be “blocked”.
- Fewer defects incoming also means fewer tests will need to be re-executed to confirm the fix resolved the problem and did not cause unintended secondary effects.
- In total the length of the overall test cycle may be significantly shorter, resulting in a reduction in total labor cost required - “fixed” cost may also be less.

The Pre-Test Impact Factor is used to quantify the overall impact of these consequences – in effect this value indicates the % reduction expected for a given test

step due to pre-test appraisals. The value may in some instances be 100% (1.0) if incoming quality is believed to be sufficiently good to simply not do certain types of tests (e.g., unit tests).

So far as I am aware no benchmark data on this parameter exist. Note that the number of defects incoming to testing in scenario 3 (from Table 1, column 6) is 679 vs. 4,601 in the test-only scenario – a reduction of approximately 85%. The values used in this example assume a 50% reduction in required test effort.

Column 5 is not used.
Column 6 calculates the effort (person months required to conduct each indicated type of testing = [(test hours per size * size) / 132] * column 4 (pre-test impact factor)
Column 7 indicates rework hours per major defect. Values indicated are rough estimates based on my experience. The post-release value (8 hours per major defect) is based on a rough "average" of several different data points provided by Jones[3].
Column 8, total rework hours, is the product of column 7 and the defects found count from table 1.

Sanity Check
Does all of this really make sense? How does this "simulation" compare to what we know about the real world? One way to evaluate the results of these models is to examine their conclusions in relation to total effort likely to be devoted to an actual project. *Sanity checking any model is always a good idea.* Experimenting with these models has shown that some published parameter values lead to totally implausible conclusions – e.g., pre-release NVA effort can exceed total project effort when some of these values are used. Obviously such a conclusion cannot be valid – at least one parameter value must be incorrect when the results do not make sense.

According to Jones2008 (ibid, p.295, table 3-29) an average 1,000 function point project will require about 97.5 person months of effort. The following table summarizes the results of the 5-scenario simulation, using the parameter values described above, to illustrate how those results relate to 97.5 person months of total effort.

1. Scenario	2. Total Project Effort	3. Pre-Release NVA	4. Pre-Release NVA % of Scenario 1	5. Post-Release NVA Effort	6. Total Effort (Pre + Post)	7. Total Effort % of Scenario 1
Scenario 1 - "test only"	97.5	62.3	100%	57.6	155.1	100%
Scenario 2 - "better"	96.9	61.7	99.0%	43.8	140.7	91%
Scenario 3 - "best"	86.5	51.3	82.3%	15.4	101.9	66%
Scenario 4 @ 4/size	82.1	46.9	75.3%	12.3	94.4	61%
Scenario 5 @ 3/size	72.3	37.1	59.5%	9.2	81.5	53%

Column 1 identifies the scenario.
Column 2 assumes a “scenario 1” project requires 97.5 person months in total. Other scenario effort requirements are reduced to reflect savings in pre-release NVA effort indicated in column 3 as a consequence of the appraisal strategies used in each.
Column 4 gives the percent of pre-release NVA effort for each scenario relative to scenario 1. As we see, scenario 3 (86.5 person months) results in an 11.3% reduction in total pre-release effort vs. scenario 1 (97.5 person months). Scenario 5 saves 25.8% pre-release vs. scenario 1.

Column 5 indicates post-release NVA effort. We see that scenario 1 post-release NVA effort (57.6 person months) is nearly as large as NVA effort pre-release. The real cost of this project is almost 50% greater than the cost measured at the time of release. Very few low maturity organizations ever measure this and quite likely are in denial, but in my experience this appears to be very realistic. Scenario 2, which introduces limited inspections in addition to testing included in scenario 1, reduces *pre-release* NVA by slightly less than 1% but also reduces post-release NVA by about 24%. Perhaps this dynamic is why so many organizations try inspections but do not sustain them – the largest savings are not evident at the time of release, and may in any case be in someone else’s budget!

Column 6 is total effort pre- plus post-release – i.e., the sum of column 2 (total pre-release effort) and column 5 (post-release effort, which is all rework and hence all NVA). As indicated in column 7, scenario 3 saves 34% relative to scenario 1.

Scenarios 3, 4, and 5 appear to be highly consistent with results reported by high maturity organizations.

Try this out with your own data or assumptions – I look forward to your feedback! For a copy of the model, please send me an email – ggack@process-fusion.net

[1] Jones, Capers, *Software Engineering Best Practices,* McGraw Hill 2009 ISBN 978-0-07-162161-8
[2] Jones, Capers, *Estimating Software Costs, 2nd Ed.,* McGraw Hill 2007 ISBN 978-0-07-148300-1
[3] Approximately 6.6 hours per post-release defect may be derived from data provided in tables 9-13, 9-14, and 9-15 in Jones2009 (ibid. p. 596-599). 10 hours is suggested in Jones2008 *Applied Software Measurement, 3rd Ed.*, p.485