A SHORT HISTORY OF THE COST PER DEFECT METRIC

Version 4.0. December 29, 2012

Abstract

The oldest metric for software quality economic study is that of "cost per defect." While there may be earlier uses, the metric was certainly used within IBM by the late 1960's for software; and probably as early as 1950's for hardware.

As commonly calculated the cost-per-defect metric measures the hours associated with defect repairs and the numbers of defects repaired and then multiplies the results by burdened costs per hour.

The cost-per-defect-metric has developed into an urban legend, with hundreds of assertions in the literature that early defect detection and removal is cheaper than late defect detection and removal by more than 100 to 1. This is true mathematically, but there is an economic problem with the cost per defect calculations that will be discussed in the article. As will be shown, cost per defect is always cheapest where the greatest numbers of defects are found. As quality improves, cost per defect gets higher until zero defects are encountered, where the cost per defect metric goes to infinity.

More importantly the cost-per-defect metric tends to ignore the major economic value of improved quality: shorter development schedules and reduced development costs outside of explicit defect repairs.

Capers Jones, Vice President and CTO, Namcook Analytics LLC

Web: www.Namcook.com

Email: Capers.Jones3@Gmail.com

INTRODUCTION

The cost-per-defect metric has been in continuous use since the 1960's for examining the economic value of software quality. Hundreds of journal articles and scores of books include stock phrases, such as "it costs 100 times as much to fix a defect after release as during early development."

Typical data for cost per defect varies from study to study but resembles the following pattern circa 2013:

Defects found during requirements =	$250
Defects found during design =	$500
Defects found during coding and testing =	$1,250
Defects found after release =	$5,000

While such claims are often true mathematically, there are three hidden problems with cost per defect that are usually not discussed in the software literature:

1. Cost per defect penalizes quality and is always cheapest where the greatest numbers of bugs are found.

2. Because more bugs are found at the beginning of development than at the end, the increase in cost per defect is artificial. Actual time and motion studies of defect repairs show little variance from end to end.

3. Even if calculated correctly, cost per defect does not measure the true economic value of improved software quality. Over and above the costs of finding and fixing bugs, high quality leads to shorter development schedules and overall reductions in development costs. These savings are not included in cost per defect calculations, so the metric understates the true value of quality by several hundred percent.

The cost per defect metric has such serious shortcomings for economic studies of software quality that a case might be made for considering this metric to be a form of professional malpractice for economic analysis of software quality.

Let us consider the cost per defect problem areas using examples that illustrate the main points..

Why Cost per Defect Penalizes Quality

The well-known and widely cited "cost per defect" measure unfortunately violates the canons of standard economics. Although this metric is often used to make quality economic claims, its main failing is that it penalizes quality and achieves the best results for the buggiest applications!

Surprisingly for a metric used widely for more than 40 years, there are no standard counting rules for calculating cost per defect. Some companies include test preparation, test execution, and defect repairs. Other companies exclude testing and only count actual defect repairs.

Worse, much of the literature that uses cost per defect does not define the specific counting rules that were used; merely the results are shown so there is no way to corroborate or challenge the values shown.

Furthermore, when zero-defect applications are reached there are still substantial appraisal and testing activities that need to be accounted for. Obviously the "cost per defect" metric is useless for zero-defect applications.

As with KLOC metrics discussed in another paper, the main source of error for the cost per defect metric is that of ignoring fixed costs. Three examples will illustrate how "cost per defect" behaves as quality improves.

In all three cases, A, B, and C, we can assume that test personnel work 40 hours per week and are compensated at a rate of $2,500 per week or $75.75 per hour using fully burdened costs. Assume that all three software features that are being tested are 100 function points in size and 5000 lines of code in size (5 KLOC).

Case A: Poor Quality

Assume that a software engineer spent 15 hours writing test cases, 10 hours running them, and 15 hours fixing 10 bugs. The total hours spent was 40 and the total cost was $2,500. Since 10 bugs were found, the cost per defect was $250. The cost per function point for the week of testing would be $25.00. The cost per KLOC for the week of testing would be $500.

Case B: Good Quality

In this second case assume that a software engineer spent 15 hours writing test cases, 10 hours running them, and 5 hours fixing one bug, which was the only bug discovered.

However since no other assignments were waiting and the tester worked a full week 40 hours were charged to the project. The total cost for the week was still $2,500 so the cost per defect has jumped to $2,500.

If the 10 hours of slack time are backed out, leaving 30 hours for actual testing and bug repairs, the cost per defect would be $2,273.50 for the single bug. This is equal to $22.74 per function point or $454.70 per KLOC.

As quality improves, "cost per defect" rises sharply. The reason for this is that writing test cases and running them act like fixed costs. It is a well-known law of manufacturing economics that:

"If a manufacturing cycle includes a high proportion of fixed costs and there is a reduction in the number of units produced, the cost per unit will go up."

As an application moves through a full test cycle that includes unit test, function test, regression test, performance test, system test, and acceptance test the time required to write test cases and the time required to run test cases stays almost constant; but the number of defects found steadily decreases..

Table 1 shows the approximate costs for the three cost elements of preparation, execution, and repair for the test cycles just cited using the same rate of $75.75 per hour for all activities:

Table 1: Cost per Defect for Six Forms of Testing
(Assumes $75.75 per staff hour for costs)

	Writing Test Cases	Running Test Cases	Repairing Defects	TOTAL COSTS	Number of Defects	$ per Defect
Unit test	$1,250.00	$750.00	$18,937.50	$20,937.50	50	$418.75
Function test	$1,250.00	$750.00	$7,575.00	$9,575.00	20	$478.75
Regression test	$1,250.00	$750.00	$3,787.50	$5,787.50	10	$578.75
Performance test	$1,250.00	$750.00	$1,893.75	$3,893.75	5	$778.75
System test	$1,250.00	$750.00	$1,136.25	$3,136.25	3	$1,045.42
Acceptance test	$1,250.00	$750.00	$378.75	$2,378.75	1	$2,378.75

What is most interesting about table 1 is that cost per defect rises steadily as defect volumes come down, even though table 1 uses a constant value of 5 hours to repair defects for every single test stage! In other words every defect identified throughout table 1 had a constant cost of $378.25 when only repairs are considered.

In fact all three columns use constant values and the only true variable in the example is the number of defects found.

In real life, of course, preparation, execution, and repairs would all be variables. But by making them constant, it is easier to illustrate the main point: *cost per defect rises as numbers of defects decline*.

Since the main reason that cost per defect goes up as defects decline is due to the fixed costs associated with preparation and execution, it might be thought that those costs could be backed out and leave only defect repairs. Doing this would change the apparent results and minimize the errors, but it would introduce three new problems:

1. Removing quality cost elements that may total more than 50% of total quality costs would make it impossible to study quality economics with precision and accuracy.

2. Removing preparation and execution costs would make it impossible to calculate cost of quality (COQ) because the calculations for COQ demand all quality cost elements.

3. Removing preparation and execution costs would make it impossible to compare testing against formal inspections, because inspections do record preparation and execution as well as defect repairs.

Backing out or removing preparation and execution costs would be like going on a low-carb diet and not counting the carbs in pasta and bread, but only counting the carbs in meats and vegetables. The numbers might look good, but the results in real life would not be good.

Let us now consider cost per function point as an alternative metric for measuring the costs of defect removal. With the slack removed the cost per function point would be $18.75. As can easily be seen cost per defect goes up as quality improves, thus violating the assumptions of standard economic measures.

However, as can also be seen, testing cost per function point declines as quality improves. This matches the assumptions of standard economics. The 10 hours of slack time illustrate another issue: when quality improves defects can decline faster than personnel can be reassigned.

Case C: Zero Defects

In this third case assume that a tester spent 15 hours writing test cases and 10 hours running them. No bugs or defects were discovered.

Because no defects were found, the "cost per defect" metric cannot be used at all. But 25 hours of actual effort were expended writing and running test cases. If the tester had no

other assignments, he or she would still have worked a 40 hour week and the costs would have been $2,500.

If the 15 hours of slack time are backed out, leaving 25 hours for actual testing, the costs would have been $1,893.75. With slack time removed, the cost per function point would be $18.38. As can be seen again, testing cost per function point declines as quality improves. Here too, the decline in cost per function point matches the assumptions of standard economics.

Time and motion studies of defect repairs do not support the aphorism that "it costs 100 times as much to fix a bug after release as before." Bugs typically require between 15 minutes and 6 hours to repair regardless of where they are found.

(There are some bugs that are expensive and may takes several days to repair, or even longer. These are called "abeyant defects" by IBM. Abeyant defects are customer-reported defects which the repair center cannot recreate, due to some special combination of hardware and software at the client site. Abeyant defects comprise less than 5% of customer-reported defects.)

Considering that cost per defect has been among the most widely used quality metrics for more than 50 years, the literature is surprisingly ambiguous about what activities go into "cost per defect."

More than 75% of the articles and books that use cost per defect metrics do not state explicitly whether preparation and executions costs are included or excluded. In fact a majority of articles do not explain anything at all, but merely show numbers without discussing what activities are included.

Another major gap is that the literature is silent on variations in cost per defect by severity level. A study done by the author at IBM showed these variations in defect repair intervals associated with severity levels.

Table 2 shows the results of the study. Since these are customer-reported defects, "preparation and execution" would have been carried out by customers and the amounts were not reported to IBM. Peak effort for each severity level is highlighted in blue.

Table 2: Defect Repair Hours by Severity Levels for Field Defects

	Severity 1	Severity 2	Severity 3	Severity 4	Invalid	Average
> 40 hours	1.00%	3.00%	0.00%	0.00%	0.00%	0.80%
30 - 39 hours	3.00%	12.00%	1.00%	0.00%	1.00%	3.40%
20 - 29 hours	12.00%	20.00%	8.00%	0.00%	4.00%	8.80%
10 - 19 hours	22.00%	32.00%	10.00%	0.00%	12.00%	15.20%
1 - 9 hours	48.00%	22.00%	56.00%	40.00%	25.00%	38.20%
> 1 hour	14.00%	11.00%	25.00%	60.00%	58.00%	33.60%
TOTAL	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

As can be seen, the overall average would be close to perhaps 5 hours, although the range is quite wide.

In table 2, severity 1 defects mean that the software has stopped working. Severity 2 means that major features are disabled. Severity 3 refers to minor defects. Severity 4 defects are cosmetic in nature and do not affect operations. Invalid defects are hardware problems or customer errors inadvertently reported as software defects. A surprisingly large amount of time and effort goes into dealing with invalid defects although this topic is seldom discussed in the quality literature.

Using Function Point Metrics for Defect Removal Economics

Because of the fixed or inelastic costs associated with defect removal operations, cost per defect always increases as numbers of defects decline. Because more defects are found at the beginning of a testing cycle than after release, this explains why cost per defect always goes up later in the cycle.

An alternate way of showing the economics of defect removal is to switch from "cost per defect" and use "defect removal cost per function point". Table 3 uses the same basic information as Table 1, but expresses all costs in terms of cost per function point:

Table 2 Cost per Function Point for Six Forms of Testing
(Assumes $75.75 per staff hour for costs)
(Assumes 100 function points in the application)

	Writing Test Cases	Running Test Cases	Repairing Defects	TOTAL $ PER F.P.	Number of Defects
Unit test	$12.50	$7.50	$189.38	$209.38	50
Function test	$12.50	$7.50	$75.75	$95.75	20
Regression test	$12.50	$7.50	$37.88	$57.88	10
Performance test	$12.50	$7.50	$18.94	$38.94	5
System test	$12.50	$7.50	$11.36	$31.36	3
Acceptance test	$12.50	$7.50	$3.79	$23.79	1

The advantage of "defect removal cost per function point" over "cost per defect" is that it actually matches the assumptions of standard economics. In other words, as quality improves and defect volumes decline, cost per function point tracks these benefits and also declines. High quality is shown to be cheaper than poor quality, while with cost per defect high quality is incorrectly shown as being more expensive.

However, quality has more benefits to software applications than just those associated with defect removal activities. The most significant benefit of high quality is that it leads to shorter development schedules and cheaper overall costs for both development and maintenance. The total savings from high quality are much greater than the improvements in defect removal expenses.

Why Cost per Defect Understates the Economic Value of Quality

Assume you have a staff of 100 people working on a large software project in a large company. At a burdened cost of $10,000 per month the monthly burn rate is $1,000,000 for the project. The cost per hour for each staff member is $75.75.

Assume you find 1000 defects via static analysis and fix them at a rate of 3 hours per defect, which amounts to $227.25 per defect or $227,250 in total. (Static analysis has a low cost per defect because preparation costs are minimal and execution costs are low.)

You assume that if these same 1000 defects are found later during system test the effort will be 8 hours per defect or $606.00 each. The total cost would be $606,000.

Your savings from early removal will be equal to:

$606,000 - $227,250 = $378,750.

By dividing the cost of early defect removal into the savings from early defect removal, the nominal return on investment (ROI) would be:

$378,750 / $227,250 = $1.66

While this calculation does show some value associated with higher quality, there is another factor that needs to be addressed with even greater value.

If the 1000 defects had not been found early, the whole project would have slipped by a month during testing. This is because a major source of schedule delays is that of excessive defects when testing starts. (In order to know the amount of schedule slippage, it is necessary to have historical data from both low-quality and high-quality applications of the same size and type.)

Therefore with early defect removal via static analysis, the project would be delivered one month earlier than by using testing alone. This means that the true value of early defect removal is not just the cost of removal per se, but the savings to the entire project by finishing one month earlier and reducing total costs by $1,000,000.

The value of early defect removal or the economic value of high quality can now be calculated with these additional savings, as follows:

$1,378,750 / $227,250 = $6.07

When the value of early delivery is added to the value from cheaper defect removal, the true economic value of high quality can now be seen and the nominal ROI rises from $1.66 to $6.07.

What this means is that "cost per defect" ignores the most significant value topic associated with high quality levels and therefore understates the value of quality by several hundred percent.

When economic analysis switches from "cost per defect" to total value including reduced schedules, the results show another dimension that is also missing from cost per defect.

With cost per defect there is very little difference between a small project of 100 function points and a large system of 10,000 function points. Costs per defect will vary somewhat, but not very much.

However, when economic analysis includes the savings associated with shorter schedules, it will be seen that the economic value of quality is directly proportional to the

size of the application measured with function points. The larger the application, the more valuable high quality becomes. This phenomenon cannot be measured using cost per defect, but it can be measured using economic analysis based on total application schedules and costs.

Why the Economic Value of Quality Goes Up with Application Size

Because "cost per defect" is not suitable for showing either the total economic value of quality or the relationship between quality and application size, this section illustrates a method of measuring economic quality value based on total development plus one year of maintenance.

To reduce the number of variables all of the examples are assumed to be coded in the C programming language, and have a ratio of about 125 code statements per function point.

Because all of the examples are assumed to be written in the same programming language, productivity and quality can be expressed using the "lines of code" metric without distortion. The "lines of code" metric is invalid for comparisons between unlike programming languages.

For each size plateau two cases will be illustrated: average quality and excellent quality. The average quality case assumes waterfall development, CMMI level 1, normal testing, and nothing special in terms of defect prevention.

The excellent quality case assumes at least CMMI level 3, formal inspections, static analysis, rigorous development such as the team software process (TSP), and the use of prototypes and joint application design (JAD) for requirements gathering.

Although all of the case studies are derived from actual applications, to make the calculations consistent there are a number of simplifying assumptions used. These assumptions include the following key points:

- All cost data is based on a fully-burdened cost of $10,000 per staff month. A staff month is considered to have 132 working hours. This is equivalent to $75.75 per hour.

- For each of the six examples, staffing for the "excellent quality" and "average quality" are shown as being equal. In real life excellent quality can be accomplished with smaller staffs than poor quality but holding staffing constant simplifies the calculations.

- Unpaid overtime is not shown nor is paid overtime. Slack time and normal lunch periods and coffee breaks are not shown.

- Defect potentials are the total numbers of defects found in five categories: requirements defects, design defects, code defects, documentation defects, and

"bad fixes" or secondary defects accidentally included in defect repairs themselves.

- Creeping requirements are not shown. The sizes of the six case studies reflect application size as delivered to clients.

- Software reuse is not shown. All cases can be assumed to reuse about 15% of legacy code. But to simplify assumptions, the defect potentials in the reused code and other materials are assumed to equal defect potentials of new material. Larger volumes of certified reusable material would significantly improve both the quality and productivity of all six case studies and especially so for the larger systems above 10,000 function points in size..

- Bad-fix injections are not shown. About 7% of attempts to repair bugs accidentally introduce a new bug, but the mathematics of bad-fix injection is complicated since the bugs are not found in the activity where they originate.

- The first year of maintenance is assumed to find 100% of latent bugs delivered with the software. In reality many bugs fester for years, but the examples only show the first year of maintenance.

- The maintenance data only shows defect repairs. Enhancements and adding new features are excluded in order to highlight quality value.

- Maintenance defect repair rates are based on average values of 12 bugs fixed per staff month. In real life there are wide ranges that can run from less than 4 to more than 20 bugs repaired each month.

- Application staff size is based on U.S. average assignment scopes for all classes of software personnel, which is approximately 150 function points. That is, if you divide application size in function points by the total staffing complement of technical workers plus project managers, the result will be close to 150 function points. This value includes software engineers and also specialists such as quality assurance, technical writers, and test personnel.

- Schedules for the "average" cases are based on raising function point size to the 0.4 power. This rule of thumb provides a fairly good approximation of schedules from start of requirements to delivery in terms of calendar months.

- Schedules for the "excellent" cases are based on raising function point size to the 0.36 power. This exponent works well with object-oriented software and rigorous development practices. It is a fairly good fit for Agile applications too, although the lack of really large Agile projects leaves the upper range uncertain.

- Data in this section is expressed using the function point metric defined by the International Function Point Users' Group (IFPUG) version 4.2 of the counting

rules. Other functional metrics such as COSMIC function points or engineering function points or Mark II function points would yield different results from the values shown here.

- Data on source code in this section is expressed using counts of logical statements rather than counts of physical lines. There can be as much as 500% difference in apparent code size based on whether counts are physical or logical lines. The counting rules are those of the author's book Applied Software Measurement.

The reason for these simplifying assumptions is to minimize extraneous variations among the case studies, so that the data is presented in a consistent fashion for each. Because all of these assumptions vary in real life, readers are urged to try out alternate values based on their own local data or on benchmarks from organizations such as the International Software Benchmark Standards Group (ISBSG).

The simplifying assumptions serve to make the results consistent, but each of the assumptions can change in either direction by fairly large amounts.

To clarify how various economic metrics work, the following tables include data based on function points, on lines of code (LOC), and on cost per defect.

The Value of Quality for Very Small Applications of 100 Function Points

Small applications in this range usually have low defect potentials and fairly high defect removal efficiency levels. This is because such small applications can be developed by a single person, so there are no interface problems between features developed by different individuals or different teams.

Table 4: Quality Value for 100 Function Point Applications
(Note: 100 function points = 12,500 C statements)

	Average Quality	Excellent Quality	Difference
Defects per Function Point	3.50	1.50	-2.00
Defect Potential	350	150	-200.00
Defect Removal Efficiency	94.00%	99.00%	5.00%
Defects Removed	329	149	-181
Defects Delivered	21	2	-20
Cost per Defect Pre-Release	$379	$455	$76
Cost per Defect Post Release	$1,061	$1,288	$227
Development Schedule (Calendar Months)	6	5	-1
Development Staffing	1	1	0
Development Effort (Staff Months)	6	5	-1
Development Costs	$63,096	$52,481	-$10,615
Function Points per Staff Month	15.85	19.05	3.21
LOC per Staff Month	1,981	2,382	401
Maintenance Staff	1	1	0
Maintenance Effort (Staff Months)	2	0	-1.63

Maintenance Costs (Year 1)	$17,500	$1,250	-$16,250
TOTAL EFFORT	8	5	-3
TOTAL COST	$80,596	$53,731	-$26,865
TOTAL COST PER STAFF MEMBER	$40,298	$26,865	-$13,432
TOTAL COST PER FUNCTION POINT	$805.96	$537.31	-$269
TOTAL COST PER LOC	$6.45	$4.30	-$2.15
AVERAGE COST PER DEFECT	$720	$871	$152

Note that cost per defect goes up as quality improves; not down. This phenomenon distorts economic analysis. As will be shown in the later examples, cost per defect tends to decline as applications grow larger. This is because large applications have many more defects than small ones.

Prototypes or applications in this size range are very sensitive to individual skill levels, primarily because one person does almost all of the work. The measured variations for this size range are about 5 to 1 in how much code gets written for a given specification and about 6 to 1 in terms of productivity and quality levels. Therefore "average" values need to be used with caution. Averages are particularly unreliable for applications where one person performs the bulk of the entire application.

The Value of Quality for Small Applications of 1,000 Function Points

For small applications of 1,000 function points quality is of course important, but it is also somewhat easier to achieve than it is for large systems. At this size range teams are small and methods such as Agile development tend to be dominant, other than for systems and embedded software where more rigorous methods such as the team software process (TSP) and the rational unified process (RUP) are more common. Table 5 shows the value of quality for small applications in the 1,000 function point size range:

Table 5: Quality Value for 1,000 Function Point Applications
(Note: 1000 function points = 125,000 C statements)

	Average Quality	Excellent Quality	Difference
Defects per Function Point	4.50	2.50	-2.00
Defect Potential	4,500	2,500	-2,000
Defect Removal Efficiency	93.00%	97.00%	4.00%
Defects Removed	4,185	2,425	-1,760
Defects Delivered	315	75	-240.00
Cost per Defect Pre-Release	$341	$417	$76
Cost per Defect Post Release	$909	$1,136	$227
Development Schedule (Calendar Months)	16	12	-4
Development Staffing	7	7	0.00
Development Effort (Staff Months)	106	80	-26
Development Costs	$1,056,595	$801,510	-$255,086
Function Points per Staff Month	9.46	12.48	3.01
LOC per Staff Month	1,183	1,560	376.51
Maintenance Staff	2	2	0
Maintenance Effort (Staff Months)	26	6	-20.00

Maintenance Costs (Year 1)	$262,500	$62,500	-$200,000
TOTAL EFFORT	132	86	-46
TOTAL COST	$1,319,095	$864,010	-$455,086
TOTAL COST PER STAFF MEMBER	$158,291	$103,681	-$54,610
TOTAL COST PER FUNCTION POINT	$1,319.10	$864.01	-$455
TOTAL COST PER LOC	$10.55	$6.91	-$3.64
AVERAGE COST PER DEFECT	$625	$776	$152

The bulk of the savings for the "excellent" column shown in table 4 would come from shorter testing schedules due to the use of requirements, design, and code inspections. Other changes that added value include the use of team software process (TSP), static analysis prior to testing, and the achievement of higher CMMI levels.

In the size range of 1,000 function points numerous methods are fairly effective. For example both Agile development and Extreme programming report good results in this size range as do the Rational Unified Process (RUP) and the Team Software Process (TSP).

The Value of Quality for Large Applications of 10,000 Function Points

When software applications reach 10,000 function points in size, they are very significant systems that require close attention to quality control, change control, and corporate governance. In fact without careful quality and change control, the odds of failure or cancellation top 35% for this size range.

Note that as application size increases, defect potentials increase rapidly and defect removal efficiency levels decline, even with sophisticated quality control steps in place. This is due to the exponential increase in the volume of paperwork for requirements and design, which often leads to partial inspections rather than 100% inspections. For large systems, test coverage declines and the number of test cases mounts rapidly but cannot usually keep pace with complexity.

Table 6: Quality Value for 10,000 Function Point Applications
(Note: 10,000 function points = 1,250,000 C statements)

	Average Quality	Excellent Quality	Difference
Defects per Function Point	6.00	3.50	-2.50
Defect Potential	60,000	35,000	-25,000
Defect Removal Efficiency	84.00%	96.00%	12.00%
Defects Removed	50,400	33,600	-16,800
Defects Delivered	9,600	1,400	-8,200
Cost per Defect Pre-Release	$341	$417	$76
Cost per Defect Post Release	$833	$1,061	$227
Development Schedule (Calendar Months)	40	28	-12
Development Staffing	67	67	0.00
Development Effort (Staff Months)	2,654	1,836	-818
Development Costs	$26,540,478	$18,361,525	-$8,178,953
Function Points per Staff Month	3.77	5.45	1.68

LOC per Staff Month	471	681	209.79
Maintenance Staff	17	17	0
Maintenance Effort (Staff Months)	800	117	-683.33
Maintenance Costs (Year 1)	$8,000,000	$1,166,667	-$6,833,333
TOTAL EFFORT (STAFF MONTHS)	3,454	1,953	-1501
TOTAL COST	$34,540,478	$19,528,191	-$15,012,287
TOTAL COST PER STAFF MEMBER	$414,486	$234,338	-$180,147
TOTAL COST PER FUNCTION POINT	$3,454.05	$1,952.82	-$1,501.23
TOTAL COST PER LOC	$27.63	$15.62	-$12.01
AVERAGE COST PER DEFECT	$587	$739	$152

Cost savings from better quality increase as application sizes increase. The general rule is that the larger the software application the more valuable quality becomes. The same principle is true for change control, because the volume of creeping requirements goes up with application size.

Return on Investment for Achieving Software Quality Excellence

As already mentioned the value of software quality goes up as application size goes up. Table 7 calculates the approximate return on investment for the "excellent" case studies of 100 function points, 1,000 function points, and 10,000 function points..

Here too the assumptions are simplified to make calculations easy and understandable. The basic assumption is that every software team member needs 80 hours of training to get up to speed in software inspections, static analysis, and the team software process (TSP). These training hours are then multiplied by average hourly costs of $75.75 per employee.

(Note that the costs of ascending the CMMI levels from 1 to 3 are not shown since they are not related to specific projects. CMMI expenses are costs associated with business units. If CMMI training were included, about another 160 hours of training per staff member would be needed over a multi-year period.)

These training expenses are then divided into the total savings figure that includes both development and maintenance savings due to high quality. The final result is the approximate ROI based on dividing value by training expenses. Table 6 illustrates the ROI calculations:

Table 7: Return on Investment in Software Quality

Function Point Size	100	1,000	10,000
Education Hours	80	560	5,360
Education Costs	$6,060	$42,420	$406,020
Savings from High Quality	$26,865	$455,086	$15,012,287
Return on Investment (ROI)	$4.43	$10.73	$36.97

The ROI figure reflects the total savings divided by the total training expenses needed to bring team members up to speed in quality technologies. In real life these simple assumptions would vary widely, and other factors might also be considered. Even so, high levels of software quality have a very solid return on investment due to the reduction in development schedules, development costs, and maintenance costs.

There are many other topics where software engineers and managers need training, and their may be other cost elements such as the costs of ascending to the higher levels of the capability maturity model. While the savings from high-quality are frequently observed, the exact ROI will vary based on the way training and process improvement work is handled under local accounting rules.

If the reduced risks of cancelled projects or major overruns were included in the ROI calculations, the value would be even higher.

Other technologies such as high volumes of certified reusable material would also have a beneficial impact on both quality and productivity. However as this book is written in 2009 there are only limited sources available for certified reusable materials. Uncertified reuse is hazardous and may even be harmful rather than beneficial.

Since this paper deals with some of the shortcomings of the "cost per defect" metric, it is instructive to see what happens to this metric across the six examples. Unfortunately cost per defect moves in the opposite direction from true economic value, and achieves its lowest levels for the largest, buggiest example!

- The lowest average cost per defect in the six examples is $520 and that occurs for the largest, buggiest example in table 5.

- The highest average cost per defect in the six examples is $871 and that occurs for the smallest, highest quality example in table 3.

As already mentioned, cost per defect is cheapest where defect volumes are greatest. This phenomenon leads to some unpleasant surprises for those interested in software process improvement. One of these surprises is that cost per defect for CMMI level 1 applications is much lower than for CMMI level 5 applications. This is because at CMMI level 5 there are very few defects but a great deal of effort goes into testing and defect removal, much of it in the form of fixed costs for writing and running test cases.

Unfortunately because cost per defect simultaneously penalizes quality and also ignores the main economic value of quality, which centers on shorter schedules and lower costs for the entire application, it is not a suitable metric for economic studies.

To study the economic value of quality, comparing the total costs of ownership (TCO) of high-quality and low-quality projects during both development and maintenance gives the most accurate value data. Of course these are long-range studies that don't produce instant results.

For short-term studies function-point metrics at least match the assumptions of standard economics, since as quality improves cost per function point declines. However function point analysis is somewhat expensive to perform, which is why this method is not as widely deployed as it might be. The advent of high-speed, low-cost function point analysis methods will no doubt increase the usage of function points for quality and economic studies.

It is an interesting question as to why the cost per defect metric continues to be used even though it produces invalid results. There seem to be both technical and sociological reasons for cost per defect to remain so popular in the software engineering and quality literature.

The technical reason is that the one of the main problems with cost per defect is not clearly visible until quality starts to approach zero-defect status. For applications with hundreds or thousands of bugs, the costs of defect repairs are so much higher than the fixed costs of preparation and execution that the problem is hard to detect.

One sociological reason has to do with cognitive dissonance, or the psychology of opinion formation. This theory was developed by Dr. Leon Festinger, and turns out to be surprisingly important in situations where new ideas compete with older ideas.

Once an idea becomes firmly entrenched, people tend to cling to it strongly, and initially reject evidence that is counter to the belief. Only when the evidence becomes overwhelming is there a change in belief patterns.

Yet another sociological reason is that cost per defect is among the easiest and cheapest metrics to calculate. Not only that, but the resulting data seems to indicate that early defect removal is extremely valuable, which it is. Even if the math is suspect, the cost per defect metric tends to show benefits from early defect removal.

Because the cost per defect metric is so easy to use and seems to provide valuable results, many people stop at this point and fail to realize that over and above defect repairs, software quality adds value by shortening total schedules and lowering both development and maintenance costs. Neither of these value-added topics can be studied via cost per defect metrics.

To study the true economics of software quality, side-by-side comparisons are needed of both development and maintenance. To normalize the data, function point metrics are the most stable and accurate. (Lines of code vary widely by language, and hence cannot be used to compare applications written in different languages.)

In order to illustrate various economic topics, the author's Software Risk Master ™ tool shows "cost per defect", "cost per LOC," and "cost per function point" in side-by-side columns.

Having all three metrics available side-by-side makes it easy to show clients how these various metrics change and how real economic value can only be measured using function point metrics.

Summary and Conclusions on the Economic Value of Quality

In spite of the fact that the software industry spends more money on finding and fixing bugs than any other activity, software quality remains ambiguous and poorly covered in the software engineering literature.

There are dozens of books on software quality and testing, but hardly any of them contain quantitative data on defect volumes, numbers of test cases, test coverage, or the costs associated with defect removal activities.

Even worse, much of the literature on quality merely cites urban legends of how "cost per defect" rises throughout development and into the field", without realizing that such a trend is caused by ignoring fixed costs and may not reflect actual economic facts.

Software quality does have value, and the value increases as application sizes get bigger. In fact, without excellence in quality control even completing a large software application is highly unlikely. Completing it on time and within budget in the absence of excellent quality control is essentially impossible.

The economic value of software quality is due to two factors: 1) Reduction in defect repair costs; 2) Reduction in development and maintenance costs.

The first of these factors is handled in a distorted fashion by the cost per defect metric, and the second factor is not handled at all. Serious economic analysis of software quality needs additional metrics besides cost per defect, and better measurement methods as well.

READINGS ON SOFTWARE QUALITY AND SOFTWARE ECONOMICS

Beck, Kent; Test-Driven Development; Addison Wesley, Boston, MA; 2002; ISBN 10: 0321146530; 240 pages.

Boehm, Barry Dr.; Software Engineering Economics; Prentice Hall, Englewood Cliffs, NJ; 1981; 900 pages.

Bundschuh, Manfred and Deckers, Carol; The IT Measurement Compendium; Springer Verlag, 2008; ISBN 978-3-540-68187-8; 643 pages..

Chelf, Ben and Jetley, Raoul; "*Diagnosing Medical Device Software Defects Using Static Analysis*"; Coverity Technical Report, San Francisco, CA; 2008.

Chess, Brian and West, Jacob; Secure Programming with Static Analysis; Addison Wesley, Boston, MA; 20007; ISBN 13: 978-0321424778; 624 pages.

Cohen, Lou; Quality Function Deployment – How to Make QFD Work for You; Prentice Hall, Upper Saddle River, NJ; 1995; ISBN 10: 0201633302; 368 pages.

Crosby, Philip B.; Quality is Free; New American Library, Mentor Books, New York, NY; 1979; 270 pages.

Ebert, C. and Dumke, R.: Software Measurement – Establish, Extract, Evaluate, Execute; Springer Verlag, 2007; ISBN 978-3-540-71648-8.

Everett, Gerald D. And McLeod, Raymond; Software Testing; John Wiley & Sons, Hoboken, NJ; 2007; ISBN 978-0-471-79371-7; 261 pages.

Festinger, Leon: Theory of Cognitive Dissonance; Stanford University Press, 1957; ISBN-10 0804701318; 291 pages.

Gack, Gary; *Applying Six Sigma to Software Implementation Projects*; http://software.isixsigma.com/library/content/c040915b.asp.

Gack, Gary; Managing the Black Hole; The Software Executives Guide to Software Project Risk; Business Expert Publishing, 2010.

Galorath, Daniel D. & Evans, Michael W.; Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves; Auerbach, Philadelphia, AP; ISBN 10-0849335930; 2006; 576 pages.

Garmus, David and Herron, David; Function Point Analysis; Addison Wesley, 2001.

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

Hallowell, David L.; *Six Sigma Software Metrics, Part 1.*; http://software.isixsigma.com/library/content/03910a.asp.

International Organization for Standards; ISO 9000 / ISO 14000; http://www.iso.org/iso/en/iso9000-14000/index.html.

Jones, Capers: A Catalog of Software Benchmark Sources; version 11 with 15 benchmark groups and 68,100 benchmark projects; available from the author upon request.

Jones, Capers and Bonsignour, Olivier; The Economics of Software Quality; Addison Wesley, 2011.

Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.

Jones, Capers; Software Engineering Best Practices; McGraw Hill, NY; 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; 662 pages.

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: "*Sizing Up Software*;" Scientific American Magazine, Volume 279, No. 6, December 1998; pages 104-111.

Jones; Capers; "*A Short History of the Lines of Code Metric*"; Version 4.0; May 2008; Capers Jones & Associates LLC; Narragansett, RI; 15 pages (monograph).

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.

Land, Susan K; Smith, Douglas B; Walz, John Z; Practical Support for Lean Six Sigma Software Process Definition: Using IEEE Software Engineering Standards; WileyBlackwell; 2008; ISBN 10: 0470170808; 312 pages.

Mosley, Daniel J.; The Handbook of MIS Application Software Testing; Yourdon Press, Prentice Hall; Englewood Cliffs, NJ; 1993; ISBN 0-13-907007-9; 354 pages.

Myers, Glenford; The Art of Software Testing; John Wiley & Sons, New York; 1979; ISBN 0-471-04328-1; 177 pages.

Nandyal; Raghav; Making Sense of Software Quality Assurance; Tata McGraw Hill Publishing, New Delhi, India; 2007; ISBN 0-07-063378-9; 350 pages.

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

Tassey, Gregory J.; The Economic Impacts of Inadequate Infrastructure of Software Testing; National Institute of Standards and Technology (NIST); May 2002; RTI Project Number 7007.011.

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