2

June, 2008

testing experience

The Magazine for Professional Testers

Test Management & Requirements
Experience & Tools

printed in Germany

free exemplar

www.testingexperience.com

ISSN 1866-5705

The Value of Software Testing to Business: The Dead Moose on the Table

by Gerald D. Everett, Ph.D.

Article Goal – to initiate discussion in the testing profession community that culminates in a concise, measurable definition of the value of software testing to business.

What A Dead Moose Looks Like

Have you ever had this experience as a software test manager or leader? The software development project is down to the last few weeks before the software is due. The testteam has identified some defects in the software that the testers have suggested are "go live" risks that could be mitigated by further testing, further corrections, further …. The business sponsor for the software says in effect, "yes, that is a risk, but it is one I am willing to take in order to achieve our (fill in the blank with "revenue", "sales", "customer", "product", "service", etc.) goals *this quarter by delivering the software when scheduled*. Later in the middle of the disaster caused by inaction on these risks, the typical witch hunt proceeds to place blame on everybody in the project … except the person who agreed to take the risk in the first place. This witch hunt in the presence of the guilty party while ignoring the guilty party is what the Canadians humorously call a "dead moose on the table".

The Real Dead Moose Dilemmas

I really see two dilemmas in the above scenario that the software testing profession needs to address. I believe that these two dilemmas are the testers' moose on the table. Both dilemmas revolve around what I consider a test professional's current inability to articulate the business value of software testing to business professionals.

The first dilemma is a question of how to gain sufficient credibility with a business sponsor so that the sponsor doesn't ignore real, identifiable risks as the development project approaches the due date. With a strong statement of testing business value in front of them, business sponsors would be much more hesitant to assume high testing-identified risks and much more disposed to finding more time, resources, budget, or whatever is needed to substantially lower those risks.

The second dilemma is the first dilemma driven up to the top of the food chain. When a new software development project is started, a budget and schedule for testing is included more often than not. That is a significant victory for testing over the 1990's when testing was done "only if we have the time". The second dilemma concerns how the testing budget and schedule are treated when the development team gets into scheduling trouble. When the developers run out the clock on the project, the project managers consistently use the testing schedule as the on time/on budget parachute. As with the risk-taking decision, as long as software testing professionals cannot articulate the value of testing to the business, then testing and its results will always be viewed as expendable.

I think it is time to solve these two dilemmas with an open discussion by testing professionals about the business value of software testing. I would like to start the discussion by some thoughts for your consideration. Hopefully you will reply to the article with your experience and insights.

Considerations for a Software Testing Value Proposition

There are several pieces to the software testing value puzzle in hand already. I'll offer what I know. I challenge others reading this article to contribute their knowledge and experience to complete the puzzle.

Software Testing Costs

There are a number of legitimate software testing costs during software development. These costs are frequently cited by business folks as a reason not to test. These software testing costs include:

- Test team salaries
- Tester training - basics
- A separate test computing environment
- Automated testing tools
- Tester training - tools

It is true that doing software testing (correctly or incorrectly) is expensive. I don't think this fact detracts from the probability of software testing having a greater value than cost to the business. It is just an incomplete story at the moment.

Software Failure Development Costs

There are a number of recurring software development costs associated with poor or nonexistent testing that I would expect to be part of a testing value statement and could offset the perceived testing costs listed above. All too often, a combination of organizational structure and internal accounting practices completely mask these costs from reconciliation with their effect on testing completeness. These software development costs include:

- The cost of code so badly written that redesign/refactoring is required
- The cost of correcting code before the software is released to end-users
- The cost of correcting code after the software has been released to end-users

Part of the reason these costs are masked from

view is a popular basis of developer incentives. All too often, bonuses, raises, or promotions are based mostly on "how many lines of code did you write?" In my 40+ years of computer professional experience, I have never encountered a company that took the "carrot" approach by incenting developers to write better code by saying, "your code for application X had the fewest defects of any other developer; therefore, you get a bonus for this year's work and a raise to continue good work next year." Similarly, I have never encountered a company that took the "stick" approach by incenting developers to improve their code by saying, "your code for application X had the most defects of any other developer; therefore, you do not get a bonus for the year or a raise for next year. On the other hand, we will help you improve toward a possible bonus next year by paying for additional programming skills training."

Why all the fuss about the cost of correcting code anyway? If you subscribe to the industry research by Basili and Boehm, the fuss is enormous. [1] For example, if your development programmers caused 1,000 defects that had to be corrected during the system testing phase, then according to Basili and Boehm your programming staff just cost your company more than $1M USD in correction effort. I have seldom seen a development project budget with $1M USD allocated in advance for corrections. The story gets worse. If the same development programmers caused 1,000 defects that had to be corrected *after* the software was released to customers/end-users, then according to Basili and Boehm your programming staff cost your company more than $14M USD in correction effort. The inescapable conclusion is that sloppy programming can cost a project more money to correct defects than the original budget to develop the software, whether those developers are in-house employees, contractors, or off-shore workers.

So the cost of correction must somehow become one of our software testing value puzzle pieces. This puzzle piece presents a particularly difficult challenge to business because the cost of correction during development is seldom reported; furthermore, the cost of correction after development is usually billed to a separate maintenance organization. Seldom does a company attempt to reconcile these support costs back to the responsible development project team.

Software Failure Business Costs

There are at least two business costs associated with software failure after its release. These costs are in addition to the costs for correction just discussed. These business costs include:

- Primary business failure costs - Internal business failure in daily operation
- Secondary business failure costs - Business failure that causes customer business failure

Primary business failure costs are those costs of lost business because the software stopped daily business activities. This cost is usually expressed as revenue loss or customers loss, very measurable quantities. A not too recent but relevant example is Toys R Us. This company decided to hop onto the internet as quickly as possible to capitalize on the vast market the internet promised instantly. The venture was a business disaster. Within 12 months, Toys R Us appeared on the web, lost the company $9M USD in expected on-line sales, and went off the air. The Toys R Us website contributed directly to a $14M USD loss ($9M USD in lost sales and $5M USD in band-aid effort to salvage the hastily constructed website). [2]

I was one of the sales that Toys R Us lost during that period. So I experienced first hand the shopper-unacceptable slowness of response time and deadly software failure that contributed to the website's demise. Performance issues notwithstanding, there was a defect in the purchase screen, detectable by ISTQB testing methods, that I suggest killed the Toys R Us internet business. Let me describe the defect and you can judge for yourself.

I was able to collect about $250 USD in Christmas toys for my grandchildren in my Toys R Us shopping cart. When I attempted to check out (purchase the toys), I encountered the following error on my billing screen, "city field required – please enter a city name." I had entered my city name, San Antonio. I was baffled. So I put on my tester hat and tried for a half hour to enter something in the billing screen city field that would be acceptable. I failed.

I cancelled my order because I couldn't get past the billing screen address field defect. Knowing the challenges in deploying a new website, I decided to be a good Samaritan and tell Toys R Us about their billing screen problem. I clicked on the Help button. In the space provided, I objectively documented the billing screen defect. Then I clicked on the submit button. Another screen popped up asking me my name and address. When I completed the screen and hit the send button, I received the same error message, "city field required – please enter a city name." By reusing the same, untested city field validation code, the Toys R Us programmer had managed to block ALL purchases and ALL customer comments that might have alerted Toys R Us to a source of their lost business. The business loss in this case was the entire on-line business, the whole enchilada. Just to tell the rest of the story, in 2002 Toys R Us reappeared on the internet via Amazon.com which is a consummate internet retailer.

Secondary business failure costs are those costs of lost business incurred because the software stopped daily business activities of that business' customers. This cost is much more difficult to assess than those of the primary business failure because the loss varies with the customers. Here are two examples of secondary business failures.

Recently, Blackberry experienced a massive, preventable computer system failure that took it off the air for two workdays. [3] Blackberry is an internet messaging service that currently has 8M customers worldwide. The outage affected only the North American customers (Canada, the USA, Mexico), maybe 4M customers. The burning question is, "how much business did 4M customers lose while they were unable to transact business with their Blackberrys?" The answers could range from "not much impact, I was on vacation and just checking in with the office" to "I lost a $10M USD contract bid because I was unable to respond by the deadline." Somewhere in between lies the truth … and a piece of the software testing value puzzle.

Another example is last year's preventable crash of TurboTax that absolutely stopped the submission of over one million Internal Revenue Service (IRS) Tax Returns in the USA just before the tax filing deadline. [4] There is an unlikely white knight in this story, the IRS. For the first time since it was founded in 1935, the IRS extended the tax filing deadline to allow a private business, the TurboTax system, to recover and submit all the returns halted by the crash. The age of miracles has not passed!

From the testing value perspective, I think it would be interesting to calculate the potential IRS penalties TurboTax customers would have been fined if the IRS had not intervened with a grace period. This is one time the secondary impact could be calculated to the penny. The IRS has a standard schedule of penalties for submission after April 15 based on the amount owed on the late tax filing and the lateness of the filing past the deadline. TurboTax had captured every late tax filing before their crash.

Using the IRS penalty schedule and the TurboTax copy of the tax filings, an exact penalty could be calculated for each late submission and the penalties totaled for the true cost of the crash to TurboTax's customers. It is easy to speculate that TurboTax's greatest fear was a class action lawsuit by their customers to at least recover all penalties that they owed the IRS due to TurboTax's crash, at most a class action lawsuit over TurboTax's guarantee to submit tax returns in a timely way. Since jail time for delinquent income tax submissions is possible under the IRS law, this particular secondary business loss just keeps on giving.

A Draft Value Statement

We have identified a number of development and business costs that can offset testing costs in software development. I propose that we begin to build a business case for software testing value from these costs. Consider the following equation as a starting point.

```
TV =    50% S DCDP * SCCED
      + 50% S (DCDP * DCCP)
      + SBLR
      - TBL
```

where	TV	is the Testing Value
	DCDP	is the number of Defect Corrections per Development project Phase
	SCCED	is the project Support Correction Cost per End-user Defect
	DCCP	is the average Development Correction Cost by project Phase
	SBLR	is the Software Business Loss due to testing Risks this project's business sponsor assumed in lieu of correction and, in fact, the risk occurred.
	TBL	is the Testing Business Loss due to incomplete test planning or test execution that allowed business loss to occur.

What I want us to be able to say to the business sponsor of my software development project is, "the testing budget for this project is $500K USD; however, the expected value of this project's testing to the company is $23.6M USD.

Here is my rationale for the testing value equation. The additive expression "50% S DCDP * SCCED" represents the value that testing brings to the development team by reducing the number of defects ultimately delivered to the end-users. Clearly, it costs more to correct software in the end-user's hands than on the developer's workstation. Basili and Boehm underscore just how disproportionate those costs are. The only point of contention is the percentage of those development defects that would have reached the end-user without testing. Based on my experience, I suggest 50% to be a conservative but appropriate multiplier of the defect costs because many, but not all defects discovered during development will directly affect end-users if not corrected.

The additive expression "50% S (DCDP * DCCP)" represents the value that testing brings to the support team by reducing the number of end-user discovered defects that the support team must correct. Using the cost of correction by phase in which the defect is discovered, the value of saving support effort is graduated by how early in the development process the defect was discovered. I suggest a 50% multiplier because I know it isn't 0% and it certainly isn't 100%. I'll start the discussion by splitting the difference. Suggestions based on your experience are welcome.

The additive factor "SBLR" represents the consequences of a business sponsor "taking a risk" that has been identified by testing and recommending the wrong action against that risk. This factor probably represents a challenge for the organization, because it requires someone to make a connection between the software development project risk decisions and the HelpDesk calls and the customers who suffered the business loss because of a poor risk gamble by the business sponsor. Although this factor could be an open-ended loss, it will have more credibility with the business community if we limit the losses just to those within the first year after release.

The expressions and factors described so far accrue to the benefit of testing. They represent value that testing uniquely contributes to software development. In business, there is always some kind of offsetting entry on the books, because there is no such thing as a free lunch. In the case of testing value, consider the factor TBL to be that offset. "TBL" represents business loss due to poor or incomplete test planning or execution or both. As with the business loss due to business sponsor risk taking, this cost will be challenging to determine. It will require a comparison of end-user detected defects with the project's test plans and test execution results to identify candidate defects missed by testing. Then the business loss due to those defects must be determined.

Here is one last thought about the test value equation. The cost of the software development project is not included anywhere in the equation, because I believe that testing value needs to be predicated on what testing accomplishes for the project regardless of the project's budget.

A Test Drive of the Test Value Equations

Here is how I derived the example testing value I quoted in the previous section. I challenge you to collect and share real metrics from a recent software development project and see how large a value your data places on testing. Your mileage may vary.

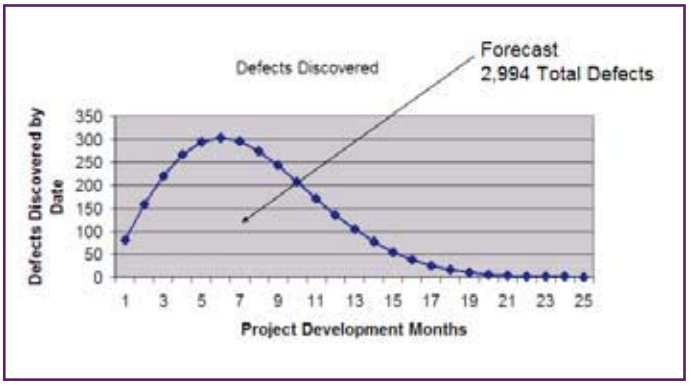
Several of the factors are inter-related and can be derived together. This derivation relies on several management practices that may or may not be implemented in your organization. Those practices are:

- Test Management that includes full test planning documents and test execution results reports (ISTQB recommended).
- Defect Management that includes a complete defect log for the project (ISTQB recommended).
- Project Management that includes the actual cost of defect correction by development phase.
- Project Management that includes project documentation of all risks identified during the project and their resolution at project end.
- Support Management that includes complete end-user/customer problem log for the software system delivered by the development project.
- Marketing Management that includes a survey of business impact on the company's customers when any of the company's software systems crash.

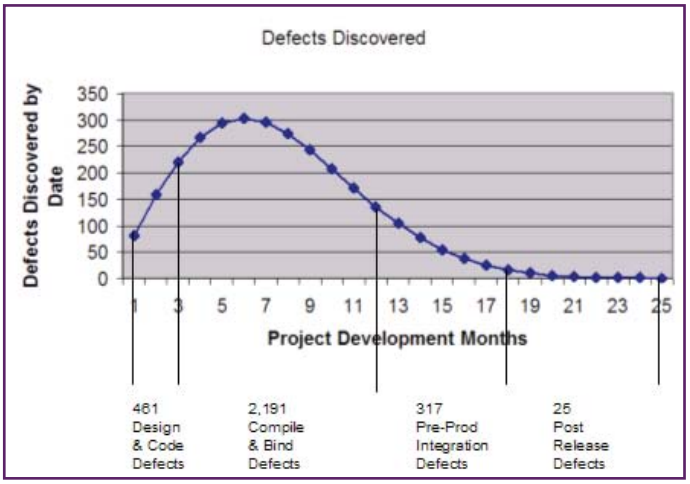
I will start with the defect log maintained by Test Management. The defect log will tell me how many defects were discovered and corrected during each phase of the development which is S DCDP. The Support Management average cost per defect correction is SCCED. I will then apply the Support Management average cost per end-user defect correction to the defect log numbers to evaluate S (DCDP * SCCED).

The Project Management average cost per defect correction per phase is DCCP. I will then apply the Project Management costs to the development defect log numbers by phase to evaluate S (DCDP * DCCP).
If the development project did not keep a defect log, then you are relegated to a SWAG (scientific wild ass guess). One reasonable SWAG is using a Rayleigh curve to estimate the number of defects that should have been found in the project's software. Group the total defects into the four general development phases identified by Basili and Boehm. Multiply each Rayleigh group total by Basili and Boehm's industry average cost to correct defects during that phase.

Here is an example Rayleigh curve for an 18-month project that developed a software application containing 3M lines of code with an average of 1 defect per 1K lines.



Here is the Rayleigh curve showing approximate grouping of defects by Basili and Boehm's development phases.



Here are the Basili and Boehm's cost factors per development phase.

- Design & Code 461 defects @ $139/correction
- Compile & Bind 2,191 defects @ $455/correction
- Pre-Prod Integration 317 defects @ $977/correction
- Post Release 25 defects @ $14,102/correction

If you have not used a Rayleigh curve to forecast defect discovery, please see Chapter 12 of Software Testing: Testing Across the Entire Software Development Life Cycle for a suggested non-statistical technique. [5]

Now we are in a position to calculate the test value expressions.

50% S DCDP * SCCED =
50% (461+2,191+317) * $14,102 = $20.9M

50% S (DCDP * DCCP) =
50% (461*$139 + 2,191*$455 + 317*$977) = $0.7M

To get the business losses due to ignored testing risks (SBLR), you will need to review the project teams closeout report to identify the disposition of testing risks raised duing development. List out the risks "taken" by the business sponsor. Next, review the Support team's list of customer problems by application or system. Match up the customer problems for the development project you are analyzing. Narrow the list of customer problems to those apparently caused by defects discovered by testing during development but not corrected because the business sponsor "took the risk."

You will need to contact Marketing to see how best to obtain the total cost of the customer problems for which the risk occurred within the first-year after development was done. Some business losses due to software problems could equate to $5,000/day while other business losses could equate to $5M/hour. Arbitrarily, I will use a number that I consider a moderate first year business loss for large software development projects that are mission critical or near-mission critical to the end-users.

SBLP = $4M

To get the business losses due to incomplete test planning or execution (TBL), you will need to follow a similar path through the project results. Start with a review of the Support team's list of customer problems by application or system. Match up the customer problems for the development project you are analyzing. Subtract from the list of problems those that you assigned to the business risk loss because once the test team has identified a risk to management, the test team's cost responsibility has been fulfilled. Categorize the remaining list of customer problems by possible cause. Match those problem categories to the test plan from Test Management. Identify those problems that fall well within the test plan but were not discovered by subsequent test execution.

You will need to contact Marketing to see how best to obtain the total cost of these customer problems caused by missed testing within the first year after development was done. Arbitrarily, I will use a number that I consider a moderate first year business loss for large software development projects that are mission critical or near-mission critical to the end-users, and the test team has completed some form of professional skills certification like the ISTQB.

TBL = $2M

There is an interesting footnote to the process for deriving the TBL factor that I hope you already caught. When you reduce the total list of problems for this application by those problems that fall well within the test plan but were not discovered by testing, the remaining problems are those that occurred outside the test plan. Perhaps a review of this list of "outliers" could indicate ways you can improve your method and effectiveness of test coverage.

We now have all of the equation expressions evaluated and have determined all the remaining factors. Here is the resulting business value of testing for our example business.

TV = $20.9M + $0.7M + $4M - $2M = $23.6M

Conclusion

We come back to the dead moose on the table. By developing an equation for measuring the value of testing to business, we can carve up the carcass into pieces small enough to assign ownership, declare business benefit or liability, and be permanently dragged off the table.

Please send your observations and suggestions for the article's proposed test value equation to jerry.everett490@gmail.com.

References

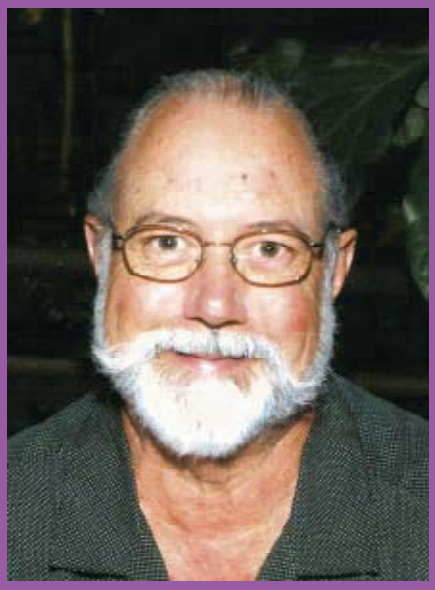
[1] Basili and Boehm, "Software Defect Reduction Top 10 List", IEEE Computer Society, vol. 34, (No.1), January 2001, pp/ 153-137.

[2] "Spending on Web Store Hurts Toys R Us Profit", Stephanie Stoughton, Washington Post Archives, Aug 17, 1999, p E1.

[3] http://www.origsoft.com/Reading_Room/nightmares.htm

[4] http://www.irs.gov/newsroom/article/0,,id=169583,00.html

[5] Everett and McLeod, *Software Testing: Testing Across the Entire Software Development Life Cycle*, Wiley-Interscience, 2007, Chapter 12, ISBN 978-0-471-79371-7.



Biography

Gerald D. Everett, Ph.D. has been designing, developing, and testing software systems for more than 40 years. Paralleling his corporate software development work, Jerry has spent 10 years as a university faculty member in Computer Sciences and Management Information Sciences departments.
For the last 8 years, his career has focused on developing and delivering courses in software testing to IBM testing practitioners and consultants. Jerry's efforts have resulted in 21 formal software testing courses ranging from 8-hour introductions to 40-hour workshops. His love of travel has been fulfilled by IBM assignments to deliver these testing courses in Amsterdam, Austin, Atlanta, Bangalore, Beijing, Boston, Brussels, Budapest, Hersley, Hong Kong, Lisbon, Montpellier, Rome, San Francisco, Stuttgart, Tokyo, and Toronto. Paralleling his corporate software testing education delivery, he continues to be invited to deliver an average of 6 testing lectures per year to US universities.

Dr. Everett published Software Testing; Testing Across the Entire Software Development Life Cycle in 2007 which is enjoying adoption primarily by graduate level university courses. He is on the Board of Directors of the ASTQB and recently achieved his ISTQB Foundation Level testing certification by using only the ISTQB syllabus and his textbook for study materials.

Jerry's newest professional interest is researching and piloting effective ways to teach software testing in virtual social worlds like Second Life.