2012 Ninth International Conference on Information Technology - New Generations

Using Test Cases to Size Systems

A Case Study

Adam Schwartz
Trueleandevelopment.com
Holden, MA, USA
aschwartz@trueleandevelopment.com

Abstract— **Productivity, quality and speed are the three key areas that all technology organizations strive to understand. Yet despite the keen interest, the ability to quantitatively measure these aspects of performance often eludes us. Central to the issue is that software development varies significantly in size from project to project making comparisons challenging.**

Measurement of the size of software is a key component to comparing dissimilar projects. Without a means to normalize for size, drawing conclusions between projects is a nearly impossible task. While solutions have existed to size systems in both lines of code (KLOC) and function points (FP), this paper explores a successful alternative approach to sizing that exhibits the benefits of existing methods with less incentive to manipulate the measurement system and a significantly lower cost. This paper proposes that the number of test cases is a viable measure of system size, available early in the process.

Keywords-defects, metrics, software sizing, test cases, function points.

I. INTRODUCTION

The desire to size a system in order to make comparisons between projects has long been a topic of vigorous discussion. Some have proposed that due to challenges with our existing sizing measurements, it would be worthwhile to avoid the question altogether [1].

IBM started their measurement of system size using KLOC. To compensate for differences between programming languages, IBM developed a technique to normalize the size of the system by translating higher level languages to assembly equivalent lines of code [2, pp. 77]. Eventually, function points evolved to compensate for issues associated with KLOC measurements [3], although not without criticisms of their own, particularly surrounding the more subjective elements of the counting process [4] [5] [6]. Today, the International Function Point Users Group (IFPUG) offers a standard method for sizing systems via the use of function points.

Like many organizations, the large company in this study (hereafter, XCorp) was exploring methods to look at quality trends within their development organizations as well as between development organizations. Parameters for that exploration were set as follows:

- The measurement had to be self-contained.
- The measurement had to be self-sustaining.
- The measurement had to be language/technology neutral.

A measurement that required many organizations to calculate would lack sustainability. Having a single organization which is capable of the collection and reporting of quality data assured one point of responsibility existed to make the measurement a reality. If the measurement had to be negotiated across many silos, the odds of failing to achieve an agreement would have increased.

Although measurement is a critical component to proving a process improvement has occurred, XCorp recognized that measurement itself does not directly add value to their customers. Therefore, any measurement system requiring significant human intervention would lack sustainability as well. As the organization continued to grow, a measurement that required manual data collection would fall by the wayside in favor of "value-added" work.

Lastly, XCorp uses a wide range of languages and technologies throughout the organization. Any measurement created should result in the ability to compare across parts of the organization.

II. LIMITATIONS OF EXISTING MEASUREMENTS

KLOC has been much derided for the ease of which developers can artificially manipulate this measurement. When used to create metrics such as cost per KLOC delivered, a productivity metric, developers can improve the appearance of efficiency by actually being less effective with their code. They simply need to write more lines of code in order to increase the denominator.

In addition, KLOC measurements only allow productivity measures of the development phase. The use of KLOC as a productivity measure suffers when you consider the relatively fixed costs of requirements or design. Higher level languages reduce the proportion of lines of code produced compared to other non-coding work, generating the appearance of severely reduced productivity [2, pp. 57-58].

The significant advantage of KLOC is that the measurement is a free byproduct of development. Tools are readily available which can count the amount of source code delivered although some languages, such as SQL, may not be adequately covered. This has some appealing characteristics,

978-0-7695-4654-4/12 $26.00 © 2012 IEEE
DOI 10.1109/ITNG.2012.41

since having to put in place a separate measurement system to watch over the work adds clearly non-value-added work to the organization.

Proponents of function point counting argue that FP measures something fundamentally different from KLOC – the size of the problem as opposed to the size of the solution. It is this differentiation that gets at the heart of one of the key issues with sizing systems using KLOC. Natural variations in solution design and/or efforts to deliberately deceive the measurement system make KLOC undesirable. An independent appraisal of how much has been delivered is needed.

Function points appear to overcome many of the challenges of the KLOC measure, but not without costs. In order to implement a function point program, training and maintaining resources to effectively count the function points is necessary.

III. Related work

In addition to the familiar FP and KLOC options, related methods exist to compensate for shortcomings perceived. Options exist that are structurally similar to FP counting, such as Mark II FP [7], Demarco's BANG Metric [8], and SPQR/20 [9].

Other methods approach the sizing problem by seeking to appropriately classify complexity of software systems [10], leveraging techniques such as functional decomposition [11], and identifying alternative proxies for the system size, such as UML diagrams [12]. Lastly, methods exist specific to a problem domain, such as for web-based development [13], and object oriented development [14].

IV. An alternative method

Gack has proposed that much of the value of function points can be derived without much of the cost. His proposal for "FP Lite" realizes 80% of the accuracy for a fraction of the cost [15]. This greatly reduced cost is far more appealing than full function point counting costs, but still represents a non-value-added effort.

Recognizing the challenges with the existing measurement systems available, XCorp set out looking for a measurement which would achieve the low cost of KLOC with the improved independence of function points. The goal was to find a measurement mechanism which was a natural byproduct of their development process rather than an add-on measurement.

Not all companies employ an independent quality assurance organization, but the shift has been towards it. Gartner Group found that from 2002-2006 the investment in formal system testing increased from 10% of project spend to 25% of project spend [16].

XCorp's Quality Assurance team approaches testing by attempting to achieve functional coverage against the features delivered. As a result, the process begins with the user's requirements. These requirements are converted into a set of scenarios. Scenarios break down the requirements into component parts to test both the should-do and should-not-do parts of the requirement. From there, the scenarios are further divided into test cases to explore both nominal and edge cases.

In essence, the transformation of requirements into test cases breaks down and normalizes the work into uniformly sized chunks. The goal of achieving complete coverage over the requirements assures that every part of the system is being assessed.

Recently, XCorp has made a move towards risk-based testing, which assesses the business value, technical complexity and test complexity of each scenario. This process minimizes the effort of writing cases against low-value, low-risk code. However, it still requires that the necessary scenarios and cases be recognized before they can be scored. If low scoring, there is no need to flesh out the test case into a series of steps, but this process still provides an adequate method of assessing how much functionality was theoretically delivered, whether the functionality is formally tested or not.

Because of the process undertaken, we believed that the use of test cases as a proxy for the size of the system would be valid.

Examination of this hypothesis was done through looking at whether planned test cases would actually act as a measurement of opportunity. We approached the issue through consideration of what an increased opportunity ought to mean. The belief was that the more functionality that was delivered, all things being equal in the process, the more defects the process ought to experience.

Analysis used a sample of approximately 1200 projects during 2007, 2008 and 2009 for which XCorp had recorded both the number of test cases and the defects. Using the sample, we compared the relationship between test cases and defects. The relationship between test cases and defects was a moderately-strong relationship with a Pearson Correlation Coefficient of .690 ($p = 0.012$). Given that many human aspects enter into this counting system - such as differences between individuals writing tests, varying means to decompose requirements into scenarios and cases, and so on, these are satisfying results from the Pearson Correlation. In addition we have provided the results of the Spearman's rank correlation coefficient since the data exhibit some outliers that can adversely affect the Pearson Correlation calculation (Table I). Both measures show a strong relationship between the proposed independent and dependent variables.

One might be inclined to argue that the relationship between defects and test cases must necessarily exist, since if you do not execute a test case you cannot find a defect. It is true that without exercising the code one cannot find a defect, but the inverse need not be true. That is, running a test case does not necessitate that you find a defect. The existence of the correlation indicates that while you are likely to find more defects when running more test cases, the possibility exists that one may run thousands of test cases and find effectively nothing – as exhibited in a number of outliers between 1,000 and 10,000 test cases executed.

TABLE I. CORRELATION COEFFICIENT OF TEST DEFECTS VS. TEST CASES

Test	Pearson Correlation	p-value	Spearman Rank Correlation
Defects v. Test Cases	0.690	0.012	0.671

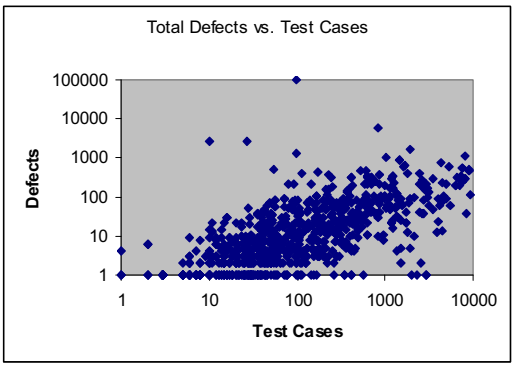

Figure 1. Defects v. Test Cases (log transformed)

In addition, although XCorp does not regularly tie production defects back to the projects that injected them, we were able to conduct a small study to ascertain whether the relationship between test cases and defects extended beyond the test phase (Table II) The risk inherent with a measurement system that uses the test cases is that inconsistencies in test capabilities could lead to an inconsistent measurement system – in effect, are we measuring the size of the system or simply the capability of the test organization? Evidence that the relationship between test cases and defects continues to hold when test capability is removed from the equation supports the assertion that test cases, when created in a consistent manner, represent a proxy for the size of the system.

We see in the scatter plot (Figure 1), that the relationship between test cases and defects appears heteroscedastic. However, one should note that both axes have been log transformed to expose the data, and the apparent diminishing variance as test cases increases is an illusion of the scale. In fact, this convergent pattern is what one would expect to see if homoscedastic data were displayed on a log scale. However, we also observe another interesting characteristic to the data – the relationship between test cases and defects appears to potentially be non-linear, as there appears to be some curvature in the data, likely indicating that there are diminishing returns from excessive testing.

This result would be the expectation for a measurement which acted as a good proxy for the opportunity for a defect – it must show that increasing the opportunity results in increases in defects. As in manufacturing, the opportunity for a defect is a proxy for how much has been delivered. If you delivered more vehicles off the assembly line, there are more opportunities for a defective vehicle. In software, this opportunity, and therefore the size of system would otherwise have been measured by KLOC or FP.

We recognize that all test cases are not equal. In XCorp's experience, the number of test case steps in a test is heavily right skewed, although the median number of steps is approximately 7. Furthermore, since XCorp is evaluating the number of defects or against a set of test cases, the law of large numbers allows the number of steps to regress to the mean and size variation case-to-case becomes a non-issue.

V. PREDICTIVE CAPABILITY OF THE MEASURE

In order to provide value to the organization, a measurement system like this must assist in activities such as predicting how many defects we are likely to detect. Although a strong positive correlation exists between the variables, there is significant variance around the mean which requires explanation in order to achieve value.

Based on this data set, we collected several additional factors which we hypothesized would explain the variance. These factors were:

- Use of unit testing prior to formal integration test
- Missing interim project deadlines
- Accepting change controls on the project
- Meeting entrance criteria to formal integration test

The first three of these factors showed evidence of statistical significance (Table III), while meeting entrance criteria did not. Unsurprisingly, by the time software reaches formal test, the quality of the code is already determined, and therefore meeting entrance criteria that assures the organization is ready to test does not influence quality.

Not only is the measurement capable of detecting a difference in the defect densities between the two populations, but the experienced difference in capability aligns with the industry experience on the effectiveness of traditional types of testing – averaging 30-35% defect removal [17]. This indicates that our measurement for opportunity produces results which align with industry findings about differences in defect density using KLOC or FP as the opportunity measure. The result suggests that the measurement system is measuring something akin to what KLOC or FP was doing, and acting in the traditional sense of what engineers would mean size of system.

TABLE II. CORRELATION COEFFICIENT OF ALL DEFECTS (TEST AND PRODUCTION) VS. TEST CASES

Test	Pearson Correlation	p-value	Spearman Rank Correlation
Defects (test and production) v. Test Cases	0.677	0.004	0.753

TABLE III. MANN-WHITNEY TEST COMPARING DEFECT DENSITY OF PROJECTS AGAINST VARIOUS FACTORS.

Group	N	Median	P-Value
Unit Testing			
No unit testing	162	0.4672	p < 0.05
Had unit testing	68	0.3428	
Missed Deadlines			
Did not miss deadline	51	0.2574	p < 0.05
Missed deadline	360	0.5362	
Change Controls			
Change controls	48	0.5545	p < 0.05
No change controls	145	0.3000	
Met Test Entrance Criteria			
Met criteria	125	0.3737	p = 0.4977
Did not meet Criteria	99	0.4949	

With potential additional explanatory factors in hand, we can construct a linear regression model (Figure 2) which can be subsequently used by XCorp's teams to predict defect densities prior to any testing occurring.

Note, that optimizing the regression equation required log transformation of the non-normal data elements (defects and test cases) prior to use in the equation. In addition, we identified an interaction effect between missed milestones and change controls. Although when considered independently both variables seem to predict higher defect densities, early regression models indicated missed milestones was unnecessary. However, the model was greatly improved by including an interaction effect. Overall, the model shows decent predictive capability for the organization; additional explanatory variables would likely improve the r-squared further.

VI. ADDITIONAL ADVANTAGES

Like FPs' early availability in the project, the majority of functional test cases can be created early in the process as well – effectively in parallel with requirements. As a result, the use of test cases as a measure of system size can be extended for use in the size of analysis, design and coding phases of the project as well.

Initial measures of size-of-system would be coarser than later measures. It often isn't until the organization reaches testing that the entire suite of test cases is fully known – some cases are written specific to the final design or coding nuances of the delivered system. This refinement should be incremental.

VII. EXTENDING THE MEASURE

Once a measure of opportunity, or size of system, is created the extension to many other metrics is a trivial activity.

```
log(total defects) = -0.133 + 0.624 log(test cases) –
0.287 documented_ut + 0.166 change_controls + 0.279
change_controls_and_missed_milestones (CC & MM)
Predictor          Coef     SE Coef      T        P        VIF
Constant          -0.133    0.083     -1.6     0.111
log(test cases)    0.624    0.034     18.22    0.000    1.275
Documented_ut     -0.287    0.050     -5.71    0.000    1.137
Change_controls    0.166    0.080     2.06     0.040    2.038
CC & MM            0.279    0.101     2.76     0.006    1.971

R Sq(adj) = 60.8%
R Sq(pred) = 60.12%
```

Figure 2. Regression Equation for Defect Prediction

Development productivity is simply development effort / test cases. Such a measure would have to exclude Quality Assurance to avoid encouraging the QA teams from artificially inflating the number of test cases created. Similar measures of requirements productivity, design productivity, etc. can be created.

Once a healthy baseline of data are established, the data could be leveraged for estimating future projects. McConnell notes that methods such as extrapolating potential time and cost based on simple extrapolation can be powerful estimating tools [18].

Development speed would be measured as development duration / test case. It would be possible to also calculate design duration / test case and similar measures against each phase of the project for estimating future project durations in addition to costs.

VIII. CONCLUSION

Because of the nature of XCorp's quality assurance process, the use of test cases as a proxy for size of system appears to be valid. Challenges with such a method would arise if the test organization was not acting as an independent appraiser of quality. Were development in control of the number of test cases needed to appraise the system, it would be a trivial effort to increase the number of cases beyond what is reasonably needed to assess functionality, thus artificially increasing the denominator and reintroducing the problems created by LOC calculations.

However, XCorp's Quality Assurance organization is not motivated by the need to do so. In practice, XCorp does not assess QA's efficiency as the cost of testing divided by the number of test cases. Instead, quality assurance's efficiency is measured on the cost per defect detected. While such a measure is described by Jones as being economically perverse [2, pp. 376-378], XCorp believes it is appropriate to evaluate the appraisal organization (and only the appraisal organization) in this fashion. By doing so, they encourage QA to create no more test cases than necessary to assess the system completely and instead reward these teams for the efficient discovery of defects. Assessing QA independently of the rest of the development organization creates the necessary tension between the teams and their metrics to assure using the proposed method of system sizing is an effective measurement.

REFERENCES

[1] Gack, G. "Cost of quality: a key effectiveness metric for software and IT.' Process Fusion. 2007. Available at: http://process-fusion.net/pdfs/Cost%20of%20Quality%20-%20a%20Key%20Effectiveness%20Metric%20for%20Software%20and%20IT%2020090417.pdf

[2] Jones, C. Applied Software Measurement. New York: McGraw-Hill, 1996.

[3] Albrecht, A.J. "Measuring application development productivity." Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.

[4] Symons, C.R. "Function point analysis: difficulties and improvements." IEEE Transactions on Software Engineering. Jan 1988. pp. 2-111.

[5] Hemmstra, F. and Kusters R. "Function point analysis: evaluation of a software cost estimation model." European Journal of Information Systems. 1991. Vol 1, No 4. pp 229-237.

[6] Jeffery, R and Stathis, J. "Specification-based software sizing: An empirical investigation of function metrics." Proceedings of the Eighteenth Annual Software Engineering Workshop. 1993. p 97-115.

[7] Symons, C. Software sizing and estimating: Mk II FPA (Function Point Analysis). John Wiley & Sons, Inc. New York, NY, USA ©1991

[8] Demarco, T. "An algorithm for sizing software products." ACM Sigmetrics Performance Evaluation Review. 1984. Volume 12, Issue 2. pp 13-22.

[9] Jeffrey, D.R, Low, G.C. and Barnes, M. "A comparison of function point counting techniques." IEEE Transactions on Software Engineering. 1993. Volume 19, Issue 5. pp 529-532..

[10] Hastings, T.E. and Sajeev, A.S.M. "A vector-based approach to software size measurement and effort estimation." IEEE Transactions on Software Engineering. 2001. Volume 27, Issue 4. pp 337-350.

[11] Verner, J. and Tate, G. "A software size model." IEEE Transactions on Software Engineering. 1992. Volume 18, Issue 4. pp 265-278.

[12] Živkovič A., Rozman I,, & Heričko M. "Automated software size estimation based on function points using UML models." Information and Software Technology. Volume 47, Issue 13. 2005. pp 881-890.

[13] Ruhe, M. Jeffery, R, & Wieczorek, I. "Using Web objects for estimating software development effort for Web applications." Proceedings of the Ninth International Symposium on Software Metrics. 2003. pp 30-37.

[14] Laranjeira, L.A. "Software size estiamtion of object-oriented systems." IEEE Transactions on Software Engineering. 1990. Volume 16, Issue 5. pp 510-522.

[15] Gack, G. "A blast from the past - scoping and estimating with context diagrams." Process Fusion. June 2009. Available at: http://process-fusion.blogspot.com/2009/06/blast-from-past-scoping-and-estimating.html

[16] Gartner Consulting Worldwide IT Benchmark Service.

[17] Jones, C. "Measuring defect potentials and defect removal efficiency." STSC Crosstalk. 2008. Available at: http://www.stsc.hill.af.mil/crosstalk/2008/06/0806jones.html

[18] McConnell, S. Software Estimation: Demystifying the Black Art. Washington: Microsoft Press, 2006.