

2008

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



COLLABORATIVE
QUALITY

OCTOBER 13-15, 2008

CONFERENCE PAPER EXCERPT
FROM THE

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

The Tao¹ of Software Defect Testing and Estimation

Scott Martin

The Regence Group
sdmarti@regence.com

James Eisenhauer

The Regence Group
jreisen@regence.com

About Scott:

Scott D. Martin currently works for The Regence Group as a Performance Analyst for the Regence Information Technology Services (RITS) division. Mr. Martin has over 16 years of multi-level collaborative experience across a broad range of industries and employers which includes two Fortune 100 companies (Raytheon and Hewlett Packard), and two large international companies (Japanese-based Kyocera and German-based Infineon Technologies).

Over his career, Mr. Martin has won repeated recognition for his work constructing a variety of forecasting and trending models, mapping process workflows and quantifying product contribution margins, improving supplier quality and enhancing corporate workforce performance through orchestrating broad behavior-based performance reforms.

Scott has earned an MBA from Portland State University, a BS in Business from the University of Oregon and a BA in Management from George Fox University.

About James:

James R. Eisenhauer currently works for The Regence Group as a Software Process and Quality Analyst. Prior to Regence, James was a Sr. Software Architect at Lockheed Martin Information Technology, and was the driving force behind a progressive software solution approach to IT Service Delivery and Governance.

During his tenure at Lockheed Martin, James was the lead consultant on many application architecture and software process engagements with major clients that included; Nike Inc, Goodyear Tire & Rubber, Symetra Financial, Department of Homeland Security, and the United Negro College Fund.

He holds a Master's of Business Administration (MBA) from the University of Tampa, Six Sigma Green Belt, ITIL Certification, ISO Auditor and a certificate of Software Engineering from the Oregon Graduate Institute School of Science and Engineering.

¹ Defined via Google search (i.e. "define: Tao") www.Summerjoy.com defines Tao as "The all that is" and www.Buddhanet.net defines Tao as "The way".

Abstract

Webster defines the *act of estimating* as “**1.** A rough or preliminary calculation, as of work to be done. **2.** An opinion: judgment.” While this is all that is defined in theory, in practice, when millions of project dollars are at stake, estimating requires the focused efforts of its end users (stakeholders) and a great many subject matter experts (SMEs).

This paper will explore the collaborative process as it unfolded for two Regence analysts tasked with building a model that estimates the completion date for the software testing cycle. Neither analyst had a clear view of the path forward, but through collaboration began to: define the problem, identify the limitations (of both the techniques employed and the data available) and to construct a defensible model that the senior leadership team could confidently present to its board of directors. The many problems encountered and limitations to implementing ideal solutions are chronicled along with the authors’ experience-based recommendations for readers facing similar challenges in their own profession.

Much has been written on the subject of software development and testing by noteworthy authors. As such, it is not the authors’ intent to expound or refute the validity of their work. Rather, the intent is to inform the reader of the understated complexity involved in building software testing models, to share some of the limitations we discovered are inherent in all models, and to encourage a collaborative approach among the in-house talent of their organization to produce a useful estimating model.

Copyright

Copyright © 2008 • The Regence Group • All rights reserved

1 Introduction

It was a mild summer afternoon when the rote tasks assigned to me as the new performance analyst were nearing completion. Without warning, I (Scott) was whisked away to my director's office and joined by one of her peers (a vice president). "We have a project for you," he stated as a host of curious staff groupies encircled me. "Predict with some level of accuracy how many defects will be found and corrected in our current software development and testing process and when that effort will end." I recited the project's scope for clarity, adding, "sure" and looking confident. "Work with whomever you need," he added, "and try to have something for me by Monday to review." It was a Thursday. I returned to my desk confident in my new purpose, having constructed predictive models before (both in graduate school and within the high-volume high-tech manufacturing industry), but this was software development, and already I sensed that it was different...

2 The Problem

Uninitiated in the ways of software development, my first task was to understand and define the development and testing steps involved in "manufacturing" software prior to production. Once the steps were defined, including the distributions of each testing step, estimating a completion date would be a matter of simple addition. To quantify the current performance of the division's testing process I planned to use techniques and descriptors borrowed from the high-technology manufacturing sector like Little's law² and flow factor³, respectively.

As I began to familiarize myself with software development projects, I learned that no two software development projects are alike. Each project is different from the last and "defects" occur randomly with varying complexity throughout the process. In this case, the software development project began over a year ago, with some design features (known as functional specifications) entering the Integration Technical Testing (ITT) sequence 8 months later. The project employed over 100 Subject Matter Experts (SMEs), with over 40 integration and performance testers running more than 50,000 acceptance test cases. Like the software development projects before it, very little information was available on past projects that related in a useful way to the new project, but the need to coordinate project delivery with sales and government filing requirements was a constant. So there it is, the "gift" given to the new guy to solve. I was stuck between the absence of useful historical data with which to construct a testing timeline (think "rock") and the need to set target delivery dates for management (think "hard place").

Finding no software superhero cape and tights in my bag of tricks, I sought a collaborative alliance with anyone who had more knowledge than I on the topic of software development and testing. After enduring many a theoretical rant from well-meaning, but nonetheless would-be collaborators, I finally met a capable insider with a big picture view who saw the problem as a classic one. Jim Eisenhauer works as a Software Process and Quality Analyst and explained that this problem is well documented in books such as the "Mythical Man

² Little's Law is defined as "Dynamic Cycle Time (DCT) = Work in process (WIP) / "Go Rate" (GR)

³ Flow factor is defined as "Cycle Time (CT) / Raw Process Time (RPT)

Month” and remains an ongoing issue among all software development and testing projects. I began reading the book, and Jim and I began a collaborative effort to construct a useful software testing defect prediction model.

3 Collaborative Evolution

My original intent required only minimal collaboration with others to define the software testing steps of the associated features, known as “functional specifications”⁴ (FS), and to assign historically related times to each process step. However, the need to involve others increased significantly with the absence of useful historic data. In addition, development projects with over 1,400,000 lines of code (1,400 KLOC) and more than 50 functional specifications, such as ours, are complex and need to be analyzed in the most efficient manner possible. Jim’s involvement added the researching resources and efficiency to forge a new estimation method for such a complex task.

We began our effort by inquiring if any predictive models were being used that we could adopt, leverage, or build upon. We discovered that the Application Development (AD) group used a predictive model based on the CONstructive COst MOdel (COCOMO) methodology to predict their testing duration, so we met with them to learn what we could. In reviewing their model we learned that they applied known industry averages, such as defect density per thousand (K) Lines Of Code (KLOC), and relied heavily on internal expert opinion for the model inputs.

After analyzing the results of their model, with its use of benchmark and qualitative inputs, we discerned that the completion dates and progression rate it produced were very different from what was planned and what we knew to be true respectively. For example, the AD group’s model used a percentage of the known development time to estimate the testing duration. Specifically, they assigned an additional 20% of the time it took to develop the software to ITT and another 20% to user acceptance testing (UAT), but those figures produced an end date that was surpassed months ago.

Though the model was inadequate for our needs, we didn’t leave empty handed. In meeting with them we observed that they were diligently measuring and recording the lines of code produced and that knowledge later proved very useful.

Next we adopted a somewhat broader view and assumed that if we couldn’t construct the process steps and populate them with historic data, then perhaps we could construct a ratio of the defects discovered for each completed FS and apply it to the remaining FS to be tested.

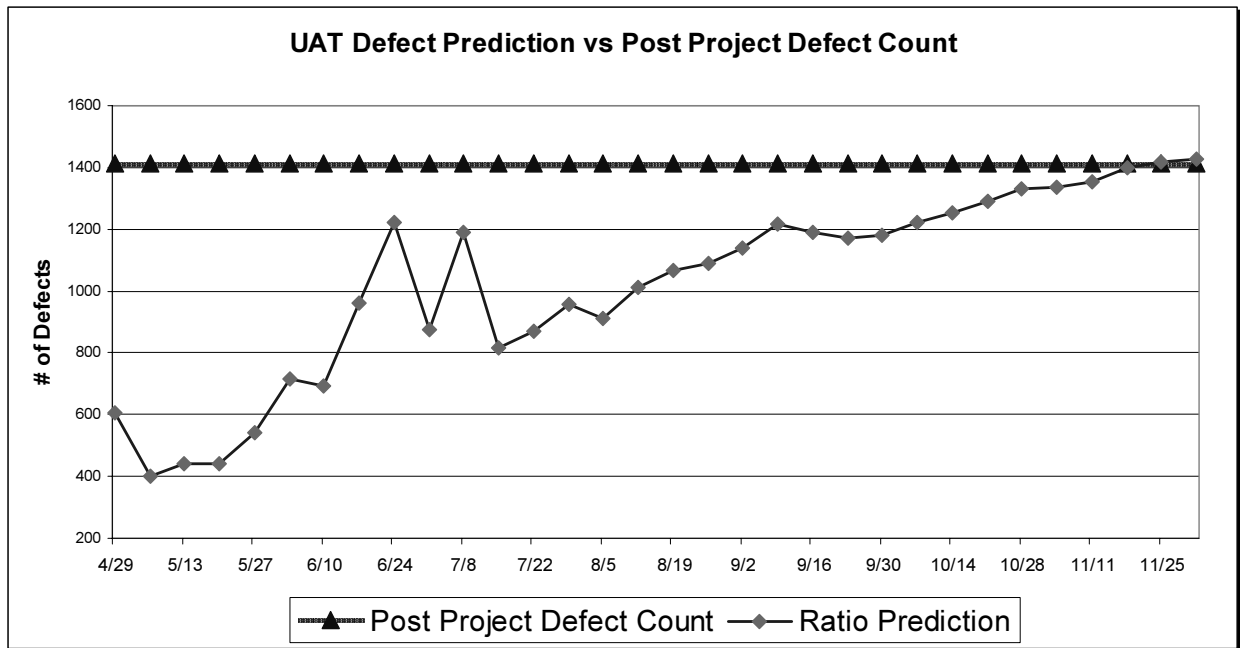
In essence we would be solving for a testing end date of “x” from a ratio of the defects found per completed FS⁵. To accomplish this we elicited the aid of the Testing Management group, which is responsible for conducting ITT and UAT testing. The testing group was invaluable in providing the testing status of each FS (because each varies in complexity) and the number of defects encountered thus far. With the data they had available we were able to apply ratios to the remaining FS (assuming the same distribution of FS complexity) to determine how many defects remained and approximately how long it would take to resolve each.

⁴ Functional Specifications are a design category that describes the intended software feature to be built.

⁵ Calculated as “Forecasted end date = (Number of Defects Discovered / Completed FS) * Remaining FS).

Unknown to us at the time was the fact that we had only found approximately 22% of the total defects to be discovered by project end. As such we observed that the ratios produced by this approach varied significantly due to the limited testing history (small sample size). While the ratios do attain some accuracy near the very end of the project, they are highly suspect at the beginning of the project. Post project analysis later revealed the inaccuracies of this method. (See Chart 1.)

Chart 1.



Post development testing efforts, we learned, are like FS development projects. They deviate. Often, defects found in testing make repeat trips through different parts of the testing sequence after a “fix” is implemented (very similar to “rework” in manufacturing). Knowing where each rework effort begins within each testing sequence (what point in ITT or UAT) is key to predicting future testing times, but that iterative information, like the prior development histories, was not available.

It soon became clear that piecing together a predictive model by summing the process parts or grouping similar product features (functional specifications) was also not viable because first-time build efforts lack a detailed historical reference.

We also recognized that, despite the size-based complexity of the build project, at the core we essentially had three problems associated with the testing phase that we needed to solve:

1. How many defects remain?
2. When will they be found?
3. How long will it take to resolve each defect?

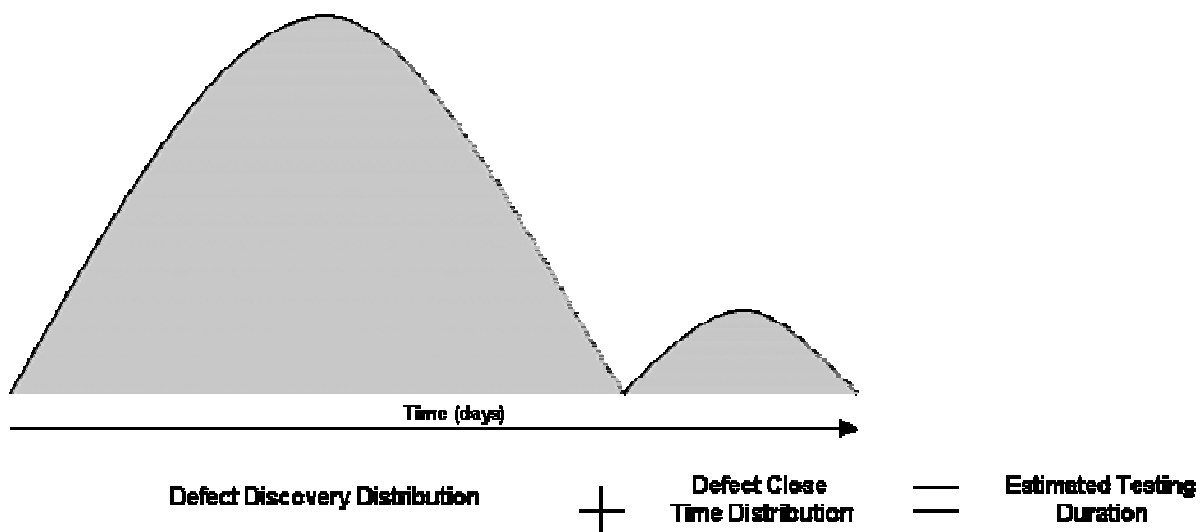
We addressed each problem in a unique way. The total number of defects was estimated using a KLOC-based ratio (the value of which we learned during our visit with the AD group). From that estimate of the total defects to be found, we subtracted the defects we had already found in order to obtain the number that likely remained.

The average defect discovery rate was an average of the sum of defects discovered each week and provided some insight into how rapidly we would find the defects we anticipated.

The average defect resolution rate was an average of the sum of defect resolutions, and was also tallied each week. This number would provide the final time period we would attach to the last known defect to estimate the testing completion date.

Although our predictive model was using a somewhat arbitrary number for the total number of anticipated defects, the discovery rate and resolution periods were based on actual performance, and we had a method. In short, adding the defect discovery period (total defects anticipated divided by the average defects discovered each week) to the average defect resolution period, will find the completion date. (See Illustration 1. below)

Illustration 1.



Although this was a good start, the model had a number of limitations. One limitation was that the model was static. It used a fixed defect discovery and resolution rate that could not account for efficiency improvements (as the development and testing teams became more familiar with the type of defects they would discover). In other words, it assumed the same level of performance from the current level to the end of the project, when in fact there would be improvement gains along the way. In addition, for any average to be useful it must relate to the same set of variables within its population, apples to apples, oranges to oranges. In our case all testing resources needed to stay constant, with no loss of key skills or reassignment of individuals from one week to the next. We acknowledged this was not the case in practice and accepted that we could do little about it without developing a more complex econometric model, which was a time-intensive endeavor beyond what both of us could commit to with our existing work loads.

However, we could make the model more dynamic by using a polynomial equation (an algebraic equation with more than one term). After a few weeks of additional data, to give the equation more data points to work from, we tested several equations before settling on a third order polynomial (an algebraic equation with three variable terms). This equation⁶ provided a reasonable inflection point rather than a flat (linear) regression trajectory for both ITT and UAT testing. (See Charts 2. and 3. below)

Chart 2.

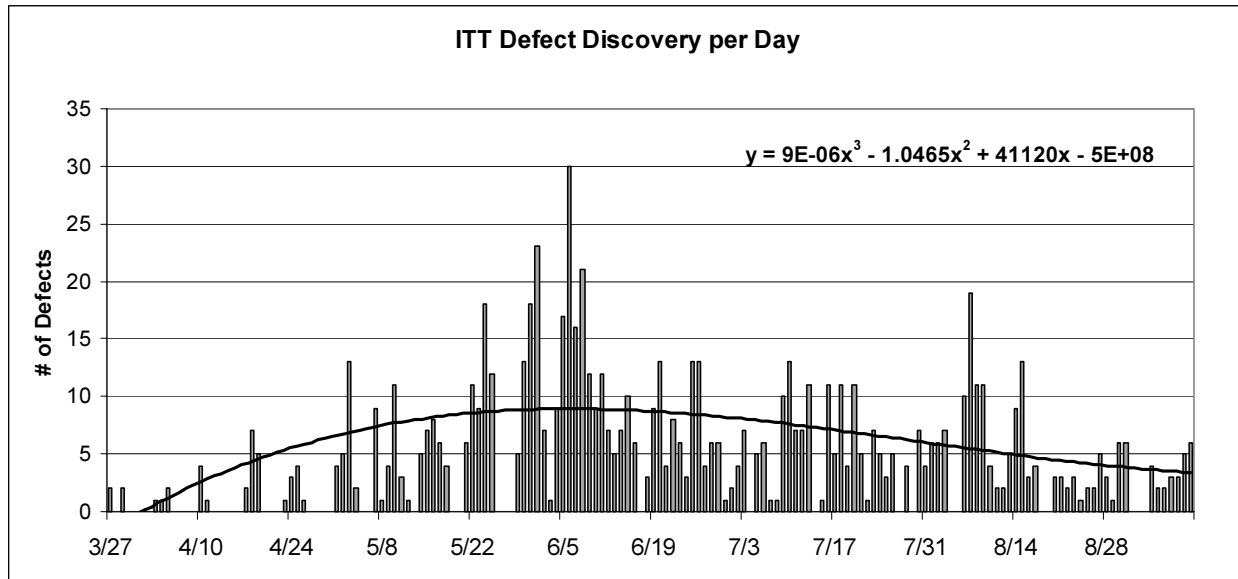
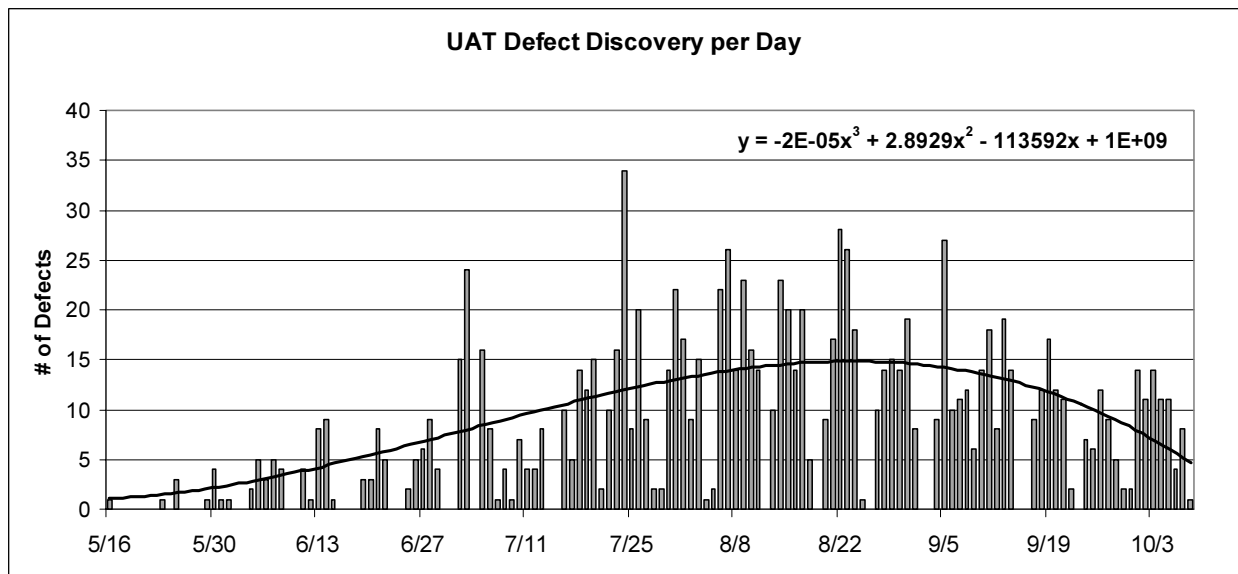


Chart 3.



⁶ Obtained from MS Excel. Right-click on the data series, select “add trend line”, “polynomial”, scroll to 3rd order.

With the addition of the polynomial equation, the forecasted completion date extended (post inflection point), as the average weekly discovery rate declined, and the resolution period extended as more persistent defects lingered. This approach, while more accurate in theory, compounded the difficulty in obtaining a project completion date from week to week as the defect discovery and resolution rates changed. This problem could be solved by using integral calculus to find the length of the area under the polynomial curve. However, getting Excel to perform this function would be a difficult script to write. Other applications, such as VB or C++ are much better suited for this, but the time demands to pursue this effort were as complex as the econometric modeling approach.

After several weeks of additional data gathering to refine our numbers, division management pressed for results. We made several informal presentations throughout the estimation effort, but now needed to provide a defensible completion date via a user friendly tool (GUI). We reviewed our methodology, simplified the front-end tool and presented the polynomial model to management.

Management was pleased with our approach and accepted our insights given the lack of detailed historical data and both the time and resource constraints we were under in building a more responsive model. However, in the absence of a model capable of responding to changes in FS complexity, testing resources and retest scenarios, the CIO (a PhD. in mathematics) insisted that we incorporate Monte Carlo Analysis⁷ (MCA) into the model. Of course! Why didn't we think of that?!

This was a logical request given the sum of our constraints and would provide a statistical measure of accuracy to support the model's conclusion if done correctly. As a graduate student, I used MCA to predict the outcome of various investment strategies and knew that it was well-suited for our application.

MCA essentially exposes an equation or model to random variation within a known reference distribution (most often gathered from historical data). In our case the reference population had a mean and standard deviation derived from just four months of testing. MCA then uses the reference population to modulate the results of our prediction model to define a greater range of possible outcomes.

Finding a quick way to incorporate MCA into the estimating model put us in collaboration with the finance group who were actively using a commercial application of this statistical tool. What we discovered, however, was that their version required qualitatively weighted entries around anticipated outcomes (again based on internal expertise rather than statistical data we now had available). So, in the absence of a commercially viable MCA tool that we could simply "plug and play", but with a good working knowledge of statistics, we collaborated with a talented VBA programmer and created it ourselves.

⁷ Wikipedia defines Monte Carlo Analysis as "a class of computational algorithms that rely on repeated random sampling to compute their results. Monte Carlo methods are often used when simulating physical and mathematical systems. Because of their reliance on repeated computation and random or pseudo-random numbers, Monte Carlo methods are most suited to calculation by a computer. Monte Carlo methods tend to be used when it is infeasible or impossible to compute an exact result with a deterministic algorithm"

4 The Model

Mechanically speaking, the MCA portion of the model works by creating a distribution of individual, or iterative, results within the statistical possibility of the reference population. While the mean and standard deviation values of the reference populations for ITT and UAT testing are hard-coded into the model, the user must input: the KLOC value, the number of defects discovered in the testing process thus far, and the number of MCA iterations to run. Limiting the number of inputs was requested by management to encourage use of the tool and to improve output consistency. We controlled the back-end calculations. (See Table 1.)

Table 1.

Testing Defect Prediction Model Input Table	
Project KLOC	1,500
ITT Defects (Discovered)	900
UAT Defects (Discovered)	1425
MC Iterations (1 - 1000)	100

Ratios are applied to the KLOC figure to predict a total number of defects to be found for the project.

ITT and UAT defect discoveries were updated each week (these numbers are for illustration purposes only)

The model can run up to 1000 iterations of Monte Carlo Analysis, but the results change very little from running 100 iterations.

When the user selects “run” the model begins the first iteration by subtracting a random, but statistically viable, number of defect discoveries from the “Remaining Defects” column and continues to subtract a statistically viable number until the defect balance goes negative. At that point the model records the number of subtraction events (days) and begins the next iteration. (See Tables 2. and 3. below)

Table 2.

MCA of UAT Defect Testing				
Capture Date	Binary Outcome	UAT Mean (Historical)		x
		UAT Sigma (Historical)		x
		# of Monte Iterations		10
		UAT Test Comp @ 95% C.I. (Earliest)		x
		UAT Test Comp @ 95% C.I. (Latest)		x
		Days Remaining	Randomized Defect Disc. Rate	Remaining Defects
0	1	1	4.151565498	1183.3
0	1	2	8.06270953	1175.3
0	1	3	0.505096696	1174.8
...
0	1	144	8.124268956	24.6
0	1	145	17.34453569	7.3
146	0	146	16.67944873	-9.4

Table 2. This is an example of how each iteration was tabulated in a separate Excel tab away from the user interface.

The programmer enters the UAT Mean and Std Dev. Values here.

The iterations are displayed from the user interface.

The upper and lower 95% confidence intervals (an output) from the model’s previous use are displayed here for quick reference.

The “Randomized Defect Disc. Rate” column subtracts a random number of defects (within the parameters of the UAT Mean and Std Dev.) from the previous remaining defect balance.

The “Binary Outcome” column changes states when the remaining defects go negative, which triggers the program to capture the total days remaining.

This MCA iteration produced an ending date 146 days from the start of UAT Testing.

Table 3.

UAT Monte Carlo Iteration Data	
Iterations 1-1000	Defect Det. Days Remaining
99	157
100	165
101	

Table 3. This is an example of how iterations 99 and 100 were recorded (apart from the user interface). With the data now in column format, the necessary statistical calculations can be performed with ease.

When all the iterations are complete the resultant distribution mimics the statistical properties of the reference population and can be used to make statistical inferences about a completion date as long as the mean and standard deviation of the reference population is valid.

At management’s request, we also kept the model’s output as simple as its inputs. Three key outputs include: a project completion date, percent to forecast values for ITT and UAT and a graph of key statistical markers for both ITT and UAT completion. The completion date is the sum of the upper 95% confidence interval values for the UAT defect discovery and resolution periods (from the date UAT testing began). The percent-to-forecast values are ratios of the defects discovered to the total predicted. Finally, the graph of statistical markers for ITT and UAT provides additional insights into the resultant MCA distribution.

Predicted Completion Date

Conditionally formatted ITT and UAT % to completion windows

Iteration Countdown window(s) (only active during use)

The “Run” button

ITT and UAT graphical representation of the defects discovered to the forecast.

The ITT and UAT defect forecast windows

The predicted UAT testing period.

Graphs of Key statistical markers for UAT.

Independent UAT and ITT “Run” Buttons for testing.

Graphs of Key statistical markers for ITT.

Note: all the statistical calculations and MCA related tables ran on a separate tab to minimize the size of the interface and to simplify its use.

Today, with this software project complete, a review of the forecasts generated with 20% of the total defects detected (approximately when the polynomial equation was added to the model) showed reasonable accuracy. Specifically, as the model approached to within six months of the project end, it yielded 90% accuracy at times (with reduced deviation as the project neared completion). Given the skewed distribution⁸ of the ITT and UAT defect discovery and resolution rates, and the "non-dynamic" methods used in the model, we were quite surprised by the overall accuracy.

Another factor that influenced accuracy was the number of total defects, calculated from a KLOC ratio each week, but this was mainly attributed to the simplicity of the approach and the lack of sufficient data to obtain a more accurate number.

Despite the model's many limitations, we were pleased with the forecasted completion date it provided. The methods outlined in this paper could be duplicated for any project, but to enhance its ability to scale to other projects would require many improvements. An econometric approach, if feasible, would allow the model to adapt to different types of software projects. A few of those project variables would include:

- Size & Skill of Testing and Development teams
- Project Risk Level
- Application, Project complexity, Domain, Programming Language
- New Development vs. Maintenance Release

However this method requires lots of data about your software development lifecycle. Designing processes and implementing systems that will enable us to capture this type of data will be our next challenge.

5 Conclusion

In completing our estimation model we collaborated with a great many talented people that tested our assertions and shaped our thinking. Throughout our journey we ran into many of the problems that have plagued software estimation for many years: size and complexity measurements, multiple levels of software development maturity, and the fact that no two software projects are the same. We did not solve any of these problems, but through our collaboration found methods to work around them. In the end, we were able to provide a tool that provided significant confidence to executive management, making way for final project budget approvals and successful system implementation.

6 Resources

- (1) Norman E. Fenton, Member, IEEE Computer Society, and Martin Neil, Member, IEEE Computer Society, "A Critique of Software Defect Prediction Models", IEE Transactions on Software Engineering Sep/Oct 1999.
- (2) Brad Clark, Dave Zubro, "How Good is the Software: A review of Defect Prediction Techniques", 2001 Carnegie Mellon University
- (3) Frederick P. Brooks, Jr. "The Mythical Man-Month", 1975, 1995, and 1999
- (4) David W. Gerbing, "Relevant Business Statistics Using Excel", 1999.

⁸ See charts 2 and 3 on page 8 of this report. The skewed distribution is also thought to have influenced the LPI figures in the graphs above, producing an MCA result lower than the Min value.