QSM® Software Almanac

Development Research Series

Winter 2017 Edition

QSM® Software Almanac

Development Research Series

Winter 2017 Edition

Published by Quantitative Software Management, Inc.

2010 Corporate Ridge, Ste 500
McLean, VA 22102
800.424.6755
info@qsm.com
http://www.qsm.com

Third Edition

TABLE OF CONTENTS

FOREWORD

The Continued Value of Parametric Estimation for Software

Capers Jones,
President and CEO of Capers Jones & Associates, and
Founder, former Chairman, and Chief Scientist Emeritus
of Software Productivity Research

Introduction

The QSM® software almanacs are valuable collections of historical information regarding software topics, in general, and about software estimation, in particular. I am pleased to be invited to participate in this current edition.

Let me begin by stating that software, unfortunately, has achieved a poor reputation as a troubling technology. Large software projects have tended to have a very high frequency of schedule delays, cost overruns, quality problems, and outright cancellations of large systems. While this disturbing reputation is often deserved, however, it is important to note that some large software projects are indeed finished on time, stay within their budgets, and operate successfully when deployed.

Successful software projects, overall, differ in many respects from the failures and disasters. One important difference is how successful projects arrived at their schedule, cost, resource, and quality estimates in the first place. It often happens that projects exceeding their planned schedules or cost estimates did not use state-of-the-art methods for determining either their schedules or their costs. Although the project is cited for overruns, the root cause of the problem is oftentimes simply inadequate planning and estimation at the beginning of the project.

Non-parametric vs. Parametric Estimation Accuracy

To better understand how I arrived at this conclusion, I performed a comparative analysis of 50 non-parametric (e.g., spreadsheet, role-based, analogy, or expert opinion, to name a few) software cost estimates against 50 parametric software cost estimates (e.g., those using parametric estimation tools such as QSM®'s SLIM®, Software Productivity Research's KnowledgePlan®, and Namcook Analytics' Software Risk Master®). The projects (shown in Table 1) were fairly large, ranging in size from about 750 to 2,500 function points (FPs), and had an assumed mode (or modal) value of roughly 1,000 FPs. Approximately 20 of the projects consisted of information technology projects; 20 were systems and embedded software; and 10 were web and cloud applications. The comparisons between the non-parametric and parametric estimates provided interesting insights and clearly favored parametric estimation.

As can be seen from Table 1, non-parametric estimates tend to be excessively optimistic. The average non-parametric estimate in the study predicted schedules that were 27% shorter than actual schedules, and costs that were 34% below actual. Undoubtedly, the project managers were blamed for being late and over budget, but the real problem was the intrinsic errors of their non-parametric estimates. By contrast, the parametric estimates were conservative, with their estimates forecasting schedules that were 6% longer and costs that were 12% higher than what actually occurred.

Table 1. Comparative accuracy of non-parametric and parametric estimates.

Non-parametric vs. Parametric Estimates			
	Non-parametric Estimates	Parametric Estimates	Difference
Optimistic by >25%	29	0	-29
Optimistic by >10%	17	1	-16
Projects within ± 5%	4	32	28
Pessimistic by >10%	0	14	14
Pessimistic by >25%	0	3	3
Total:	50	50	0

One contributing cause that helps explain why some of the parametric estimates were pessimistic is most likely the large amount of reuse within the projects, a factor that was not incorporated into the estimates at the onset. In other words, the pessimism was due to the clients neglecting to include a key factor in the estimate inputs, and not a shortcoming of the parametric methodology itself. The fail-safe mode, then, appears to be that software cost estimates should be slightly conservative or pessimistic, and never err on the side of being overly optimistic. Certainly, software estimates should not be optimistic by more than 30%, which seems to be the mode for non-parametric estimates (Jones, 2007).

Software Cost Drivers

A more detailed root cause analysis of the problems of non-parametric estimates revealed some interesting issues. To explain them, it is best to start with "software cost drivers," or the major elements of the software projects that need to be estimated.

There are six general cost drivers for software projects in the 1000-FP-size range. Table 2 ranks these six cost drivers in order of magnitude, with the most expensive topic at the top of the list, which is the cost of finding and fixing bugs. Some readers will be surprised that the second most expensive is that of producing paper documents, which often costs more than writing the code itself. In fact, for military software projects, document volumes are about three times larger than for similarly-sized commercial projects. Very often, the cost of producing paper documents is the number one cost driver for defense software. Conversely, the agile software development methodology has reduced paperwork costs and, for agile projects, the coding and paperwork places are reversed. Even for agile projects, however, finding and fixing bugs is still the number one software cost driver.

Table 2 clearly demonstrates that, when compared against non-parametric estimates, parametric estimates tend to be far more accurate for all six cost drivers. Although non-parametric estimates are fairly good for coding predictions, they are extremely poor for the two major cost drivers of finding and fixing bugs and producing paper documents.

Table 2. Software cost drivers that impact the accuracy of non-parametric vs. parametric estimates.

Estimating Accuracy by Software Cost Driver *(Assumes 1000 FPs or 53,000 Java statements)*			
		Non-parametric Estimates	Parametric Estimates
1	Finding and fixing bugs	Very Poor	Excellent
2	Producing paper documents	Very Poor	Excellent
3	Programming or coding	Good	Excellent
4	Requirements changes	Poor	Good
5	Project management	Poor	Good
6	Meetings/communications	Poor	Very Good
Overall Combined Assessment:		**Poor**	**Very Good**

Summary

Unfortunately, less-than-optimal non-parametric estimates are not the only problem that software projects encounter. Poor quality control is an endemic problem. Excessive requirements creep is another pervasive challenge. But these only intensify the difficulties when poor, overly optimistic, non-parametric estimates start projects off in a hazardous direction. Unsurprisingly, software executives often regard software as perhaps their least capable technical organization due to the high numbers of total failures and the frequency of cost and schedule overruns. Good parametric estimates, on the other hand, can start projects on the right track.

The articles that follow provide additional insights into the use and advantages of parametric estimation, and I hope readers will enjoy the collection of interesting papers in this year's edition of the QSM® Almanac.

References

Jones, C. (2007). *Estimating software costs*. New York, NY: McGraw Hill.

EXECUTIVE SUMMARY

"You do not move ahead by constantly looking in a rearview mirror. The past is a rudder to guide you, not an anchor to drag you. We must learn from the past but not live in the past."

– Warren W. Wiersbe,
American pastor and conference speaker

"We keep moving forward, opening new doors, and doing new things, because we're curious and curiosity keeps leading us down new paths."

– Walt Disney,
American entrepreneur, animator, and film producer

"Discovery consists of seeing what everybody has seen and thinking what nobody has thought."

– Albert von Szent-György,
Hungarian-American physiologist and winner of Nobel Prize in Medicine, credited with discovering Vitamin C

"Many ideas grow better when transplanted into another mind than the one where they sprang up."

Oliver Wendall Holmes,
American physician, poet, professor, lecturer, and author

The Double-Edged Rearview Mirror

Angela Maria Lungu, Editor

The preceding article by Capers Jones touches on a common set of challenges that have plagued the software and estimation community for the last thirty years: projects that come in over budget and over schedule, yet still end up underperforming, particularly in quality, despite having overworked the development staff in the process. Seriously? After more than thirty years, have we not yet learned how to better manage software development and produce high quality products within at least the estimated cost and budget?

Perhaps we have, but we are failing in our interpretation and practical application of those lessons, or perhaps, as Warren Buffett once noted, "In the business world, the rearview mirror is always clearer than the windshield." Very possibly the underlying reason for our failure to improve significantly over the last three decades is that, although we can very clearly articulate what went wrong and what should be done better, we have difficulty recognizing or differentiating conditions requiring specific approaches and applying these lessons appropriately – we require more adaptability and flexibility in applying yesterday's lessons to today's scenarios.

Our last almanac focused on the basic skills and metrics that are as relevant today as they were yesterday. Larry Putnam, Sr., eloquently described the estimating environment at the beginning of his career, and we gained insight into the value and foundation of those basics, such as the five core metrics and power of leveraging historical data. This year's almanac reaffirms these foundational skills, and then looks at how these have been adapted to account for changing conditions, particularly regarding new methodologies and across both the public and private sectors, to show where adaptation is especially needed.

The importance of core metrics, to include quality, is revalidated. D. Putnam/Putnam-Majarian give an elegant overview of quality and reliability ("Understanding Quality and Reliability") to begin the discussion. Below expands on this ("Are Pre-Release Defects Predictive?") by analyzing the predictive nature of pre-release defects using a sampling of projects from the last 15 years, concluding that they are still strongly correlated with post-release defects. In addition to quality, Madden applies all the core metrics in a very useful methodology for improved outsourcing across the software development lifecycle ("Five Core Metrics to Reduce Outsourced Software Project Failure").

Leveraging historical data is not just "déjà vu all over again;" it is critical to estimating future projects and validating proposed plans. Staiger ("Why Should I Care about Actual Data") discusses the role of historical data collection and the importance of validating and collecting this critical information, and highlight the need for continuous tracking and data collection in nice detail.

The complexity and interwoven nature of today's software also require constant vigilance. Continuous tracking and monitoring is essential as new ways of using software, more re-use of legacy systems, and new code to integrate/customize/configure the various pieces introduce more potential for derailed projects. Ensuring that estimates are realistic and empirically developed is only one mitigating step. Continuous monitoring and reforecasting is another. There are several articles presented by D. Putnam/Putnam-Majarian ("Full Circle Estimating"), L. Putnam ("The Importance of Continuous Project Tracking"), and Ciocco ("Increasing Project Success by Tracking the Big Picture" and "Adaptive Forecasting for Agile Project Tracking"), which all address this topic in detail, touching on traditional monitoring and reforecasting, as well as effectively adapting this cycle for agile projects.

Ciocco ("Better IT Budgeting") and D. Putnam ("Taking the Guesswork out of IT Project Budgeting") take it up a notch by describing how to better integrate capacity planning and demand management for enhanced IT budget planning, while Shuman ("Avoiding Doomed Software Projects") focuses on staff buildup practices.

Can these techniques be adapted for new methodologies, such as agile? The answer is resoundingly positive. While Avery ("How Agile Are We, Really?") reports on the pulse of the community and what troubles most developers, Berner ("What Should We Estimate in Agile?") addresses many of the issues head on, summarizing the first discussions coming out of the newly established QSM® Agile Round Table. Daniel ("Agile Sizing Matters") presents a wonderful look at agile sizing, showing how to adapt existing techniques for use in these types of projects.

Finally, we have several practical articles that addressing the management side of development, from Dekkers ("10 Steps to Better Metrics"), Beckett ("A Business Leader's Primer"), and D. Putnam ("How a Center of Excellence Can Help Teams Develop Excellent Software"). Beckett lays out an especially nice overview of how to overcome many common mistakes when managing software projects. Dekker's lessons for successful software measurement provide valuable insights and complements Beckett's nicely, and leads up to D. Putnam's final discussion on the value of establishing a successful estimation center of excellence. Clearly, the focus of these last articles is on "active management" and how best to incorporate it into our own organizations and teams.

Looking back critically on our projects, both good and bad, provides important insight into successful (and unsuccessful) methods and practices. This is one "edge" of the double-edged rearview mirror. Not adapting or being flexible in applying (or misapplying) those lessons, however, is the second of those "double-edges." The challenge posed is how to build on these foundational methods to better address and anticipate how our community is evolving.

To meet that demand, we must look ahead at how to increase efficiencies, speed, and quality, essential to the way software is being integrated into our lives. We must be flexible and adaptive in applying the old in new ways, requiring innovation and creativity, without being locked to the past (or with our eyes on that mirror). As Bill Joy, co-founder of Sun Microsystems, observed, "You can drive a car by looking in the rearview mirror as long as nothing is ahead of you. Not enough software professionals are engaged in forward thinking." Our hope at QSM® is that these articles will help stimulate discussion and thinking along these lines, while keeping us focused on the road ahead.

From all of us, enjoy this year's edition of the QSM® Software Almanac!

1. FIVE CORE METRICS

"We're entering a new world in which data may be more important than software."

– Tim O'Reilly,
founder of O'Reilly Media; popularized the terms open source and Web 2.0

"The bitterness of poor quality remains long after the sweetness of meeting the schedule has been forgotten."

– Anonymous

"How does a project get to be a year late?...One day at a time."

– Frederick P. Brooks,
American computer architect and software engineer, best known for his seminal book, The Mythical Man-Month

"Quality is the ally of schedule and cost, not their adversary. If we have to sacrifice quality to meet schedule, it's because we are doing the job wrong from the very beginning."

– James A. Ward,
Software developer and conference speaker

Five Core Metrics to Reduce Outsourced Software Project Failure

Joseph A. Madden

This article originally appeared in the April 8, 2016, edition of the GCN online journal, and is reprinted here with permission.

Outsourcing was supposed to make government IT executives' lives easier. Yet in too many cases, it's had the opposite effect, leading to cost overruns, inefficiencies, and solutions that do not work. Remember the initial rollout of Healthcare.gov? Exactly.

It doesn't have to be this way. Believe it or not, there's a proven solution that has withstood the test of time. In 1977, Lawrence Putnam Sr. discovered the "physics" of how engineers build software by successfully modeling the nonlinear relationship between the five core metrics of software: product size, process productivity, schedule duration, effort and reliability. The five core metrics make a powerful tool that can be used at each phase of the software acquisition lifecycle to help government IT program managers make more objective, quantitative decisions (Figure 1).

Figure 1. Summary of five core metrics incorporated in all four phases of the software acquisition lifecycle.

Phase 1: Pre-acquisition

In this phase, the five core metrics are used to develop an independent "should cost" estimate using a parametric estimation tool that includes an assessment of expected effort, staffing and schedule duration to deliver the required scope of functionality at a target reliability (Figure 2). The independent government estimate should explore all viable options. If done right, this should lead to reasonable program parameters and expectations that will be specified in the request for proposal when it is issued. Note: in the chart below, product size is measured in implementation units (IU), which is equivalent to writing a logical source line of code or a technical step in configuring a commercial off-the-shelf package.

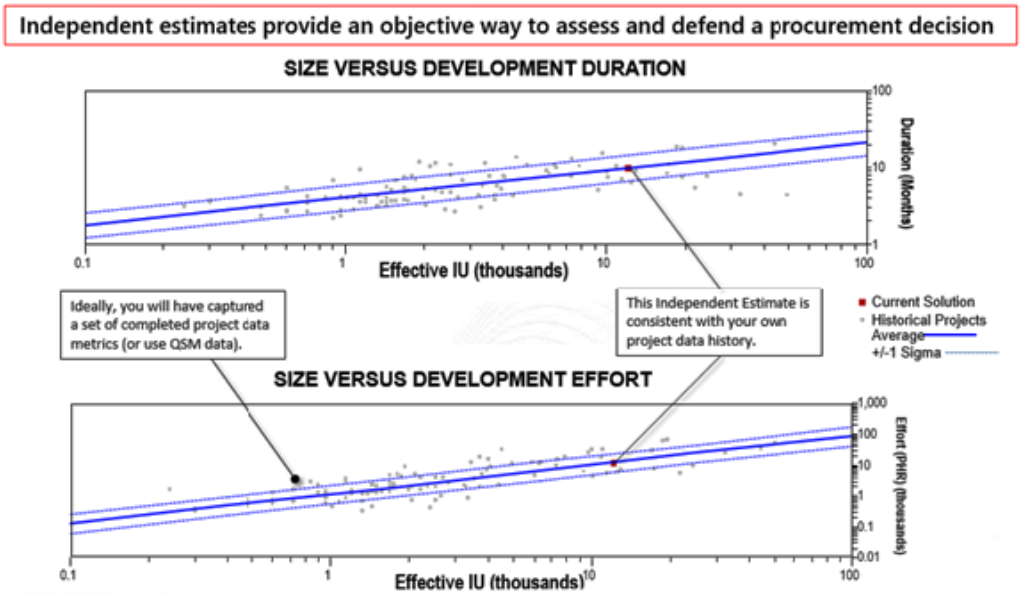


Figure 2. Scatterplot of independent estimate compared against historical data for schedule and effort to assess the proposed project plan.

Phase 2: Request for proposals

During this phase, it is very important to ensure the RFP (1) quantifies the scope of required functionality, (2) identifies any key management constraints and (3) requires vendors to report regular, well-defined status metrics to include construction progress vs. plan and defects discovered. Example status metrics:

Did work on the project start on time? Many vendors struggle with initial ramp up of a new project after contract award. By monitoring the plan vs. actual staffing curve IT managers can get an early indication of whether the project is actually starting on time.

Is the project release on track to deliver? Measure the amount of functionality planned for the next release that has been developed and unit tested. (Note: this should use an agreed upon sizing unit such as lines of code, function points or user stories.) Unlike percent complete status, which can easily be "fudged," working software is an objective measure of progress that is hard to dispute.

Will it be a quality product? The cost to find and fix defects goes up exponentially over time. Measure development defects discovered by month and by severity, which is an objective benchmark of the vendor's efforts to remove defects early through inspection and testing.

Has there been a change in scope? Change can be embraced as long as those revisions to the scope of required functionality are quantified and schedule and cost estimates are revisited.

Phase 3: Award

The third phase is about the analytical process of objectively assessing the bidders and scoring their cost and technical proposals. A cost evaluation should weed out vendors who appear to be lowballing to win, as well as those who appear to be padding their estimates (see Figure 3, below).

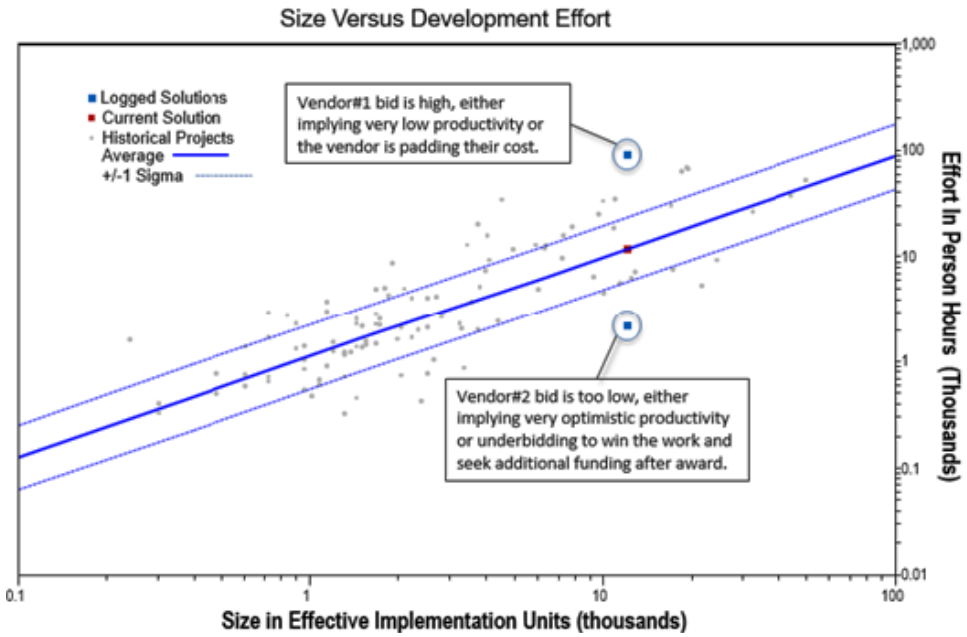


Figure 3. Scatterplot of historical data (size versus development effort) and an independent estimate used to validate individual vendor bids.

The technical evaluation should assess the skill of the development team, not the proposal writer. It should take a hard look at whether bidders can provide quantitative data (i.e., the five core metrics) for each of their past performance qualifications to demonstrate they can perform the work.

QUALITY OF METRICS PROVIDED	BENCHMARK COMPARISON W/ INDUSTRY, CLIENT	KEY PERSONNEL INCLUDED	ASSESSMENT	RATING
Incomplete/low quality/cannot be verified	N/A	N/A	High risk. Likely low process maturity (below CMMI Level 2)	Unsatisfactory
Satisfactory	Below Average	N/A	Medium-high risk. Productivity below average.	Marginal
Satisfactory	Average or Above Average	No	Medium risk. Favorable past performance, but proposed personnel did not work on those projects.	Acceptable
Satisfactory	Average	Yes	Low risk. Key personnel worked on past performance projects with average productivity.	Above Acceptable
Satisfactory	Above Average	Yes	Low risk. Key personnel worked on past performance projects with above average productivity.	Outstanding

Figure 4. Example summary of technical evaluation of vendor-provided metrics, used to assess vendor's ability to perform the proposed work.

Phase 4: Post-award

The fourth phase is about assessing progress against the contract baseline to ensure that the program is on track. If changes in direction are proposed, they need to be understood and quantified in order to evaluate the impact to schedule and cost.

Remember that openness and trust are important components of the vendor/customer relationship. The phases described above allow government IT program managers to have a better understanding of how applications are being developed so they can make sure they are receiving a high-quality product without overpaying. Likewise, the vendor gets the opportunity to potentially develop a long-term relationship with the agency by sharing valuable quantitative information from beginning to end. It's a win-win for everyone.

How Agile Are We, Really?

Ethan Avery

An often-discussed topic within the developer community is the permeation of agile into its project estimation processes. While the Agile Manifesto recently celebrated its 15th birthday, it has just been over the last several years that agile appears to have gained substantial momentum in becoming the official method by which many companies shepherd their projects. Or has it...?

Table 1. Comparison of relative waterfall and agile project success.

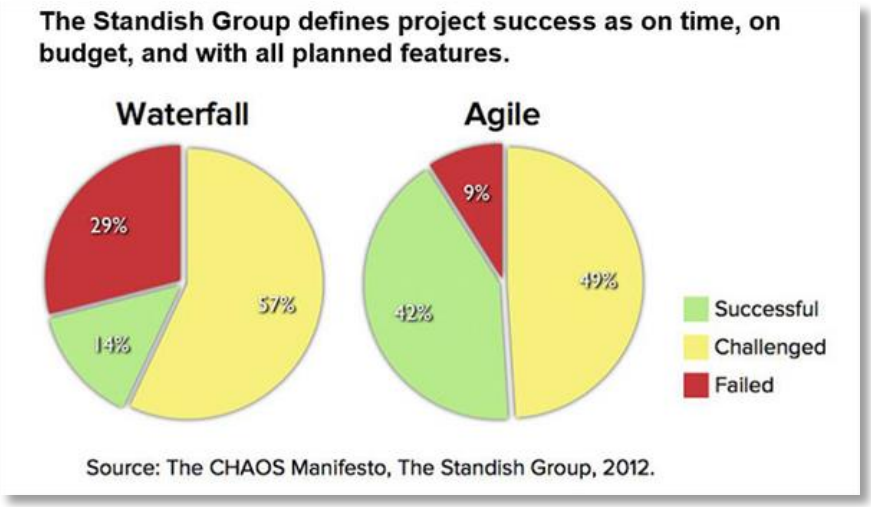


Table 1 above shows that the use of the agile development methodology over the traditional waterfall methodology has yielded desirable results in terms of reaching business goals of delivering on time, within budget, and with complete functionality. Yet, IT project measurement people can still be heard voicing concerns about fully adopting agile for their projects.

The following observations, gleaned from talking to many clients trying to estimate agile projects, are from a purely estimating perspective. Before agile emerged, many development shops were building their projects via waterfall, ERP, RUP, and other methods, all of which provided respective value. Upon agile's debut, it was but a whisper, among a few, as the next leading development methodology. Over the years, skeptical minds were slowly changed and adoption increased. Today, more organizations are using agile to develop their projects, but a surprising number of estimators report, in a hushed tone, that they aren't really developing in agile, despite the proclamation from "on

high." Rather, they are using more a combination of agile and other development methods, yielding an agile hybrid, sometimes sheepishly referred to as "wagile" or "agilefall" .

Regardless of the development method, the "C-level" still wants a project done on time, within budget, and stable at delivery, but faith in agile has not yet seemed to fully blossom, hence the inclination to interject older (and familiar) development methods along with agile.

One aspect that has been reported as quite challenging is that the sizing of agile projects is quite challenging due to the potential subjectivity as to how to measure size consistently in story points across divisions within an organization. A story point may mean different things to different groups across a company. However, this not unlike the challenge faced when trying to define any standard size entity across an organization (aside from function points, which is a global standard).

Yet another point of concern is the transition of terminology from older development methods to agile. Many people appear to realize that the term "project" is not used in pure "agile speak." Instead, agile thinks in terms of "delivered functionality." And there are other new terms that, in some cases, refer to familiar concepts, but they are new, nonetheless, and need to be employed in the agile world if developers and managers are to talk to each other.

Perhaps this is the natural evolution of a methodology maturing, and agile is continuing to find its place in the estimation/development world. At the end of the day, however, the "C-level" leader is looking for how much a project will cost and how long it will take to deliver, agile or not.

Agile Sizing Matters

Jay Daniel

This article originally appeared in the May 18, 2016, edition of the Projects at Work online journal, and is reprinted here with permission.

Do not deny the classical approach, simply as a reaction, or you will have created another pattern and trapped yourself there. — Bruce Lee

Though agile methods are steadily gaining in popularity, not everyone has bought into them because they don't always see the benefits agile is supposed to bring to the table. Some of the frustrations include never ending development and lack of documentation. But the problem doesn't lie with agile because it was never intended to be a "silver bullet" solution. It was meant to offer a way to be adaptable to changing circumstances.

Unfortunately, many people have hidden behind the guise of "doing agile" development to cover up poor fundamentals, lazy behaviors, and a lack of mature and disciplined processes. No matter your chosen method of "how" to deliver, any successful software development effort cannot escape the importance of knowing and quantifying "what" you want to build. And to do this, you can't get away from certain elementary inputs to the estimation and planning process, one of which is size.

Don't Be Dogmatic

In 1967, world-renowned martial artist Bruce Lee realized that as proficient as he had become with the *Wing Chun kung-fu* style he studied for so many years, it was limited, as were all traditional fighting styles, when considered singularly. Doesn't a boxer with a longer reach have an advantage? What about the environment? Is a style that relies heavily on kicks, like *Tai Kwan Do*, as effective if you're fighting in closed quarters? One of Lee's students was basketball Hall of Famer Kareem Abdul-Jabbar who stood 7'2", while Lee was 5'7". Would a single style be equally effective for these two men?

Lee understood that these styles were developed with ideal situations in mind, but that didn't necessarily match reality, equating it to swimming on dry land. This led to his creation of a series of concepts, a style without style called *Jeet Kune Do*. But while Lee saw the value in not limiting his style and remaining fluid to his environment, he also saw the necessity to maintain key concepts, — which he called the "four ranges of combat" — that are fundamental to being a complete martial artist.

In 2001, a group of 17 software developers gathered together in a Utah resort to discuss the challenges and limitations of the traditional approaches to software development, recognizing the need to streamline the bloated, cumbersome processes that had been instituted over the years. Like

Lee, they were looking for a way to reduce what is deemed unnecessary, but also realized that there were foundational areas or concepts that should remain. And here is where it seems many agile projects seem to miss the boat. Teams that believe that there is no need for foundational components of software development because they "are doing agile" (rather than "being agile") do so because they lack the discipline and process maturity necessary to implement the methodology appropriately.

The writers of the Agile Manifesto didn't say that process and tools weren't important, only that they believed there was greater value in individuals and interactions. It's not that following a plan isn't valuable, just that the ability to respond to change offered greater value than the dogmatic following of the plan. A mature development team, whether they prescribe to agile or not, understands this and will inherently demonstrate that level of discipline.

The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. — Jim Highsmith

Is It Bigger Than a Breadbox?

Agile development should be flexible enough to fit any container, but you need to know the size of the container. Knowing the size of the system you want to build is crucial to setting your team on a path toward success, however that is defined for you.

"But we're doing agile, so we don't gather requirements up front, thus can't size the system." That is nonsense. If I want a house built, I have to have an idea of how big it needs to be for my family to live in it and store my possessions. Why are we building this system? I must be able to find out some of the initial reasons behind the decision to build this software.

Will this be the final size? Most likely not, but that's ok. Just like I can put on additions to my house, I can add to my software estimate. And this seems to be where some developers take issue with trying to size their efforts — *estimates are not commitments*. This is where the beauty of agile methodologies can really shine and show their true value, by providing a way to adapt to those changing needs. However, someone is footing the bill for the effort and should know what they are signing up for. Business value drives prioritization within a project (product backlog), so why shouldn't it carry the same weight in deciding whether to do a project?

Where Do I Begin?

Agile says no upfront requirements should be gathered as those requirements will change so the details and specifics will be fleshed out as the project progresses. In the beginning, all that is needed is an approximation that compares the relative size of the user stories identified. Planning poker is one popular exercise used to accomplish this. The same can be done with sizing. You only know what you know up front, but as time goes on greater details will reveal themselves about the system and the accuracy of your estimate will become more refined, as can be seen in Figure 1 below.

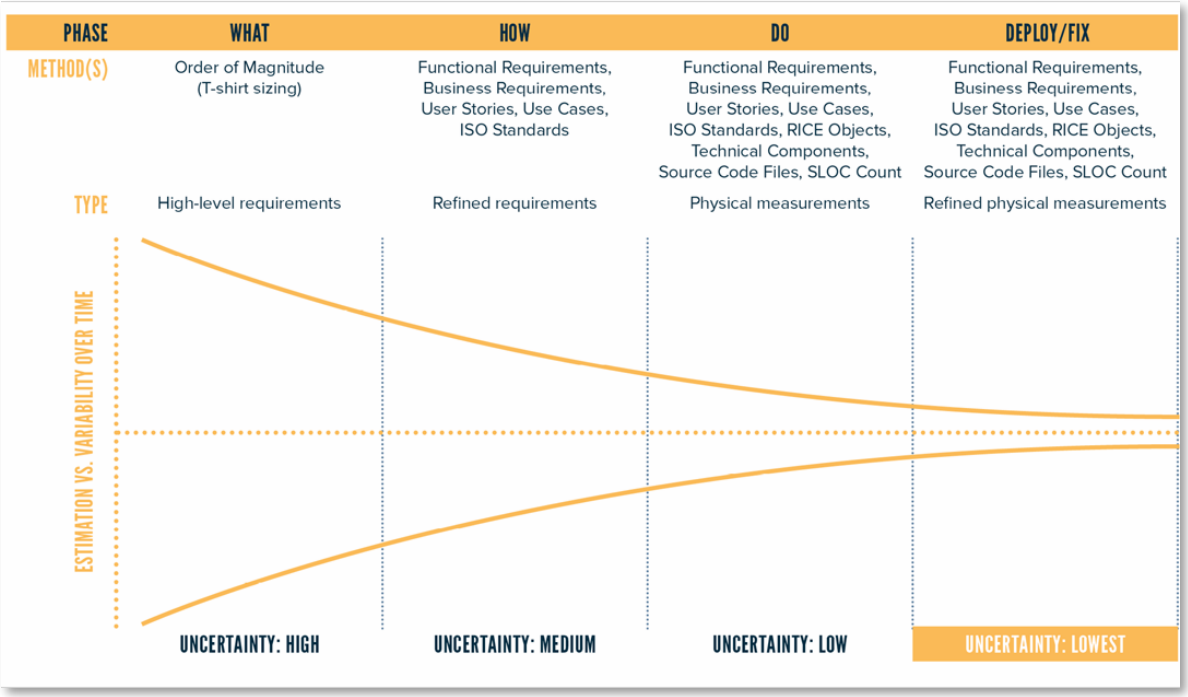


Figure 1. Cone of uncertainty and sizing methods throughout the software development lifecycle.

One of the 12 principles of the Agile Manifesto is that working software is the primary measure of progress. But progress can't be measured effectively without quantifying the size of the software that needs to be built, the size of working software completed to date, and the size of changes to scope. QSM®'s "Sizing Infographic" displays numerous ways to determine software size and correlations to project development. This offers an excellent starting point in determining just how big of a "house" your team is looking to build.

Fundamental elements, like sizing, remain crucial to software development projects. Embracing new delivery approaches, such as agile, does not mean we need to abandon practices that allow us to be successful in our delivery. The writers of the Manifesto were looking to rid us of the layers of unnecessary routines that found their way into software development and get back to the basics.

Are Pre-Release Defects Predictive?

Paul Below

Abstract

Pre-release defects are predictive of post-release defects. It is demonstrated that a model using software size, peak staff, and pre-release defects usefully predicts 83 percent of the variation in post-release defects.

Demographics

This article examines quality prediction, specifically whether defects detected prior to product release can be used to predict defects encountered release. Prediction of post-release defects is very useful for planning and scheduling, as well as to evaluate quality improvement actions. Estimation of post-release defects provides information on what is to be expected: what is likely to happen when the software is released.

Table 1. Basic demographic statistics of the selected projects.

		Effective SLOC	MB Duration (Months)	Productivity Index (PI)	MB Peak Staff (People)
N	Valid	2361	2363	2361	1842
	Missing	2	0	2	521
Mean		51437.09	7.3038	12.556	22.937
Percentiles	25	3038.00	3.6000	8.405	6.000
	50	8424.00	5.8300	12.660	10.765
	75	27153.00	8.7600	16.786	21.000

Table 2. Additional defect statistics of the selected projects.

		MB Effort (MM)	Defects (SIT-Del)	Defects First 30 Days
N	Valid	2363	2035	981
	Missing	0	328	1382
Mean		147.896	169.60	22.5168
Percentiles	25	8.530	6.00	.0000
	50	19.022	27.00	6.0000
	75	54.470	112.00	18.2499

The sampling frame used for this study contains projects in the QSM® database with full operational capability (FOC) between 2001 and 2011, and that reported either pre- or post-release defects. Post-release defects are those detected during the first month (30 days) of production. Pre-release defects are those detected from Integration Testing through Release. Tables 1 and 2 above show the statistics for the selected projects.

The defect data is heavily skewed right (is indicated by a Mean much larger than the Median or 50th percentile), and is shown in Figure 1. In a histogram, the tail extends off the right side of the graph, which is referred to as "skewed right." Size, effort, and peak staff are also all skewed right.

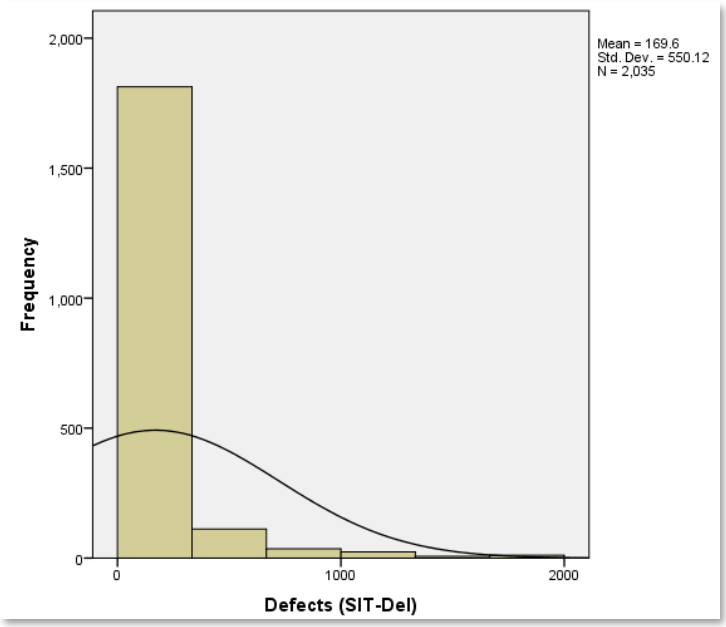


Figure 1. Defect histogram for the selected projects.

Therefore, to normalize the data, it will need to be transformed. the log transformation will be used on all the variables except Productivity Index (PI). PI is already an exponential transformation, and so will not need to be transformed. Most of the projects are business applications (Table 3).

Table 3. Respective percentages of application types.

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Avionic	27	1.1	1.1	1.1
	Business	2044	86.5	86.5	87.6
	C&C	97	4.1	4.1	91.7
	Microcode	3	.1	.1	91.9
	Process Control	4	.2	.2	92.0
	Realtime	15	.6	.6	92.7
	Scientific	17	.7	.7	93.4
	System	138	5.8	5.8	99.2
	Telecom	18	.8	.8	100.0
	Total	2363	100.0	100.0	

Defect Correlations

The pre- and post-release defects are correlated at .455. This agrees with the pre-release defects being predictive of post-release (Table 4).

Table 4. Correlation of pre- and post-release defects for the study projects.

		Log Defects SIT Del	Log Defects First 30 Days
Log Defects SIT Del	Pearson Correlation	1	.455**
	Sig. (2-tailed)		.000
	N	1750	565
Log Defects First 30 Days	Pearson Correlation	.455**	1
	Sig. (2-tailed)	.000	
	N	565	713

**. Correlation is significant at the 0.01 level (2-tailed).

Size: Pre-release defects show an equally strong correlation with size. However, post-release defects have a weaker correlation with size, indicating that size alone is not a good predictor of post-release defects. What if both size and pre-release defects are used (Figures 2 and 3)?

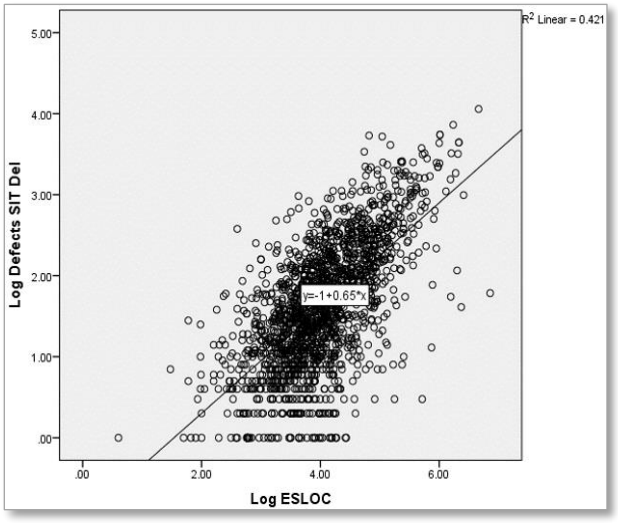


Figure 2. Log of pre-release defects against size.

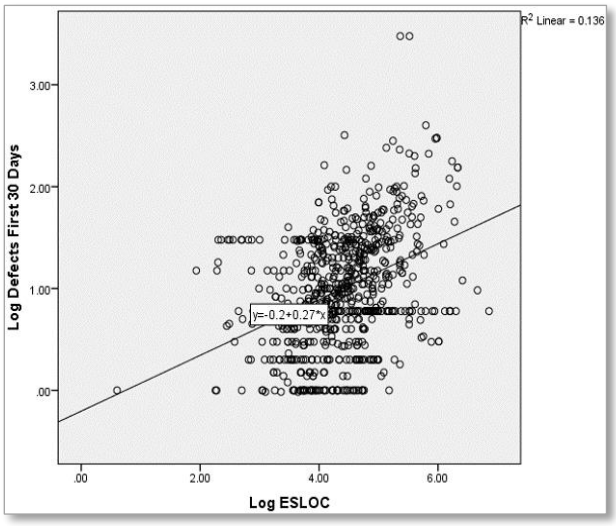


Figure 3. Log of post-release defects against size.

Multivariate Regression

Next, a regression model using size and pre-release defects together was created, to predict post-release defects. This produces a good result (Tables 5 and 6 and Figures 4 and 5).

Table 5. Model summary of the regressions[c,d].

Model	R	R Square[b]	Adjusted R Square	Std. Error of the Estimate
1	.906[a]	.821	.820	.50397

a. Predictors: Log Defects SIT Del, Log ESLOC
b. For regression through the origin (the no-intercept model), R Square measures the proportion of the variability in the dependent variable about the origin explained by regression. This CANNOT be compared to R-Square for models which include an intercept.
c. Dependent Variable: Log Defects First 30 Days
d. Linear Regression through the Origin

Table 6. Model summary of the coefficients[a,b].

Model		Unstandardized Coefficients		Sig.	95.0% Confidence Interval for B	
		B	Std. Error		Lower Bound	Upper Bound
1	Log ESLOC	.121	.017	.000	.088	.155
	Log Defects SIT Del	.262	.037	.000	.189	.335

a. Dependent Variable: Log Defects First 30 Days
b. Linear Regression through the Origin

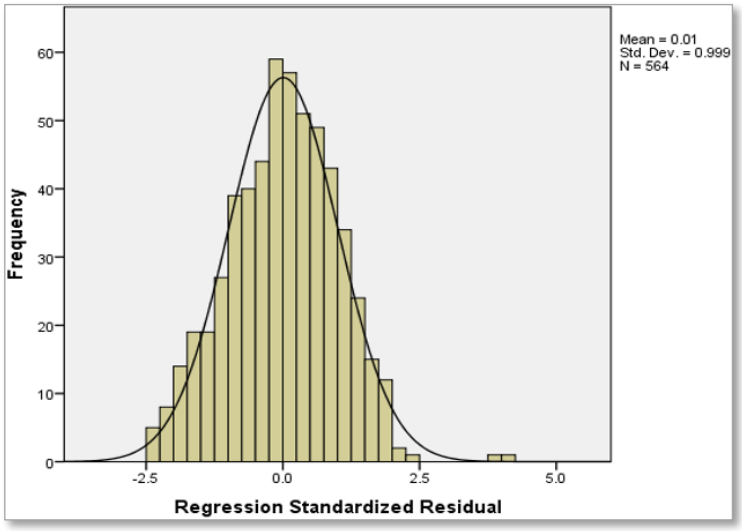


Figure 4. Histogram of dependent variable log of post-release defects.

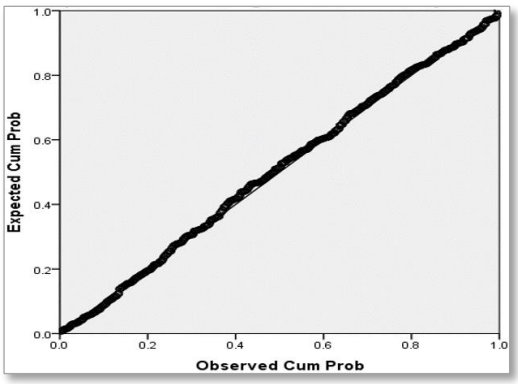


Figure 5. Normal P-P plot of regression standardized residual (dependent variable log post-release defects).

The model can be improved slightly by adding an additional variable, peak staff. Other variables examined do not improve the model (Tables 7, 8, and 9 and Figure 6).

Table 7. Model summary[e,f].

Model	R	R Square[b]	Adjusted R Square	Std. Error of the Estimate
1	.902[a]	.814	.814	.53114
2	.910[c]	.828	.828	.51085
3	.913[d]	.833	.832	.50357

a. Predictors: Log ESLOC
b. For regression through the origin (the no-intercept model), R Square measures the proportion of the variability in the dependent variable about the origin explained by regression. This CANNOT be compared to R Square for models which include an intercept.
c. Predictors: Log ESLOC, Log Peak Staff
d. Predictors: Log ESLOC, Log Peak Staff, Log Defects SIT Del
e. Dependent Variable: Log Defects First 30 Days
f. Linear Regression through the Origin

Table 8. Coefficients[a,b].

Model		Unstandardized Coefficients		Sig.	95.0% Confidence Interval for B	
		B	Std. Error		Lower Bound	Upper Bound
1	Log ESLOC	.244	.005	.000	.233	.254
2	Log ESLOC	.156	.015	.000	.127	.185
	Log Peak Staff	.328	.052	.000	.226	.430
3	Log ESLOC	.105	.020	.000	.066	.144
	Log Peak Staff	.226	.058	.000	.113	.340
	Log Defects SIT Del	.177	.046	.000	.087	.267

a. Dependent Variable: Log Defects First 30 Days
b. Linear Regression through the Origin

Table 9. Excluded variables[a,b].

Model		Beta In	t	Sig.	Partial Correlation	Collinearity Statistics Tolerance
1	Log Defects SIT Del	.441[c]	6.257	.000	.276	.073
	Log MM	.357[c]	5.326	.000	.237	.082
	Log Peak Staff	.345[c]	6.297	.000	.277	.120
	Log MB Duration	.036[c]	.590	.556	.027	.105
	Productivity Index (PI)	-.138[c]	-1.893	.059	-.086	.073
2	Log Defects SIT Del	.301[d]	3.856	.000	.174	.058
	Log MM	.112[d]	1.159	.247	.053	.038
	Log MB Duration	.037[d]	.630	.529	.029	.105
	Productivity Index (PI)	-.067[d]	-.933	.351	-.043	.071
3	Log MM	-.083[e]	-.759	.448	-.035	.030
	Log MB Duration	-.047[e]	-.763	.446	-.035	.092
	Productivity Index (PI)	.016[e]	.218	.827	.010	.065

a. Dependent Variable: Log Defects First 30 Days
b. Linear Regression through the Origin
c. Predictors in the Model: Log ESLOC
d. Predictors in the Model: Log ESLOC, Log Peak Staff
e. Predictors in the Model: Log ESLOC, Log Peak Staff, Log Defects SIT Del

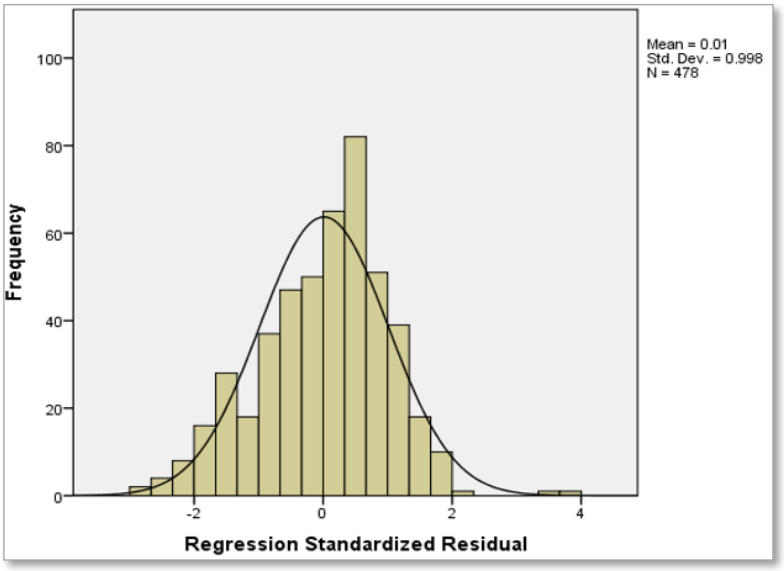


Figure 6. Histogram of dependent variable log post-release defects.

Stratification

Stratification of the data set can often be used to reduce the variation in an estimating model. Application Type is one variable that is often used to stratify. Earlier, it was shown that the pre- and post-release defects had a Pearson correlation of .455.

Looking at just the business applications (Table 10), the sample size is reduced from 565 to 385 (sample includes only those projects that had *both* pre- and post-release defects), but the correlation coefficient does not increase (decreasing from .455 to .409).

Table 10. Business only application correlations[a].

		Log Defects SIT Del	Log Defects First 30 Days
Log Defects SIT Del	Pearson Correlation	1	.409**
	Sig. (2-tailed)		.000
	N	1465	385
Log Defects First 30 Days	Pearson Correlation	.409**	1
	Sig. (2-tailed)	.000	
	N	385	516

**. Correlation is significant at the 0.01 level (2-tailed).

a. App Type = Business

On the other hand, the Avionic Project sample size is small (27), although the correlation is quite high (.832) (Table 11).

Table 11. Avionic only application correlations[a].

		Log Defects SIT Del	Log Defects First 30 Days
Log Defects SIT Del	Pearson Correlation	1	.832**
	Sig. (2-tailed)		.000
	N	27	27
Log Defects First 30 Days	Pearson Correlation	.832**	1
	Sig. (2-tailed)	.000	
	N	27	27

**. Correlation is significant at the 0.01 level (2-tailed).

a. App Type = Avionic

When dealing with a large data set, it is always good practice to examine various strata that seem to be the most important to determine if further improvements to the model can easily be made.

The SLIM® tool suite estimates and forecasts defects, both pre- and post-release. It does this based on size and staff, and then allows tuning to historical defect metrics. The regression technique used in this article could be used to further tune defect estimates in Phase 4 within tools such as SLIM-Estimate® and SLIM-Control®. It could also be used as an independent validation or adjustment of an earlier estimate that was based on pre-release defects collected on the project to date.

Abbreviations Used in Tables and Figures

- ESLOC: Effective Source Lines of Code
- MB: Main Build
- MM: Man-Months
- N: Sample Size
- P-P Plot: Probability – Probability Plot
- Sig: Significance, also known as P-value
- SIT-DEL: System Integration Test through Delivery
- SLOC: Source Lines of Code
- Std Dev: Standard Deviation

2. MANAGEMENT

"It's easy to come up with new ideas; the hard part is letting go of what worked for you two years ago, but will soon be out of date."

–Roger von Oech,
American speaker, conference organizer, author,
and toy-maker whose focus has been on the study of creativity

"If two men on the same job agree all the time, then one is useless. If they disagree all the time, both are useless."

–Darryl F. Zomuck,
American film producer and studio executive,
earning three Academy Awards

"If I have seen further, it is by standing on the shoulders of giants."

– Sir Isaac Newton,
English mathematician, astronomer, and physicist,
one of the most influential scientists of all time

"If you think it's expensive to hire a professional, wait until you hire an amateur."

– Anonymous,

The Importance of Continuous Project Tracking

Lawrence H. Putnam, Jr.

This article originally appeared in the August 2016 online edition of Project Management Times, and is reprinted here with permission.

Developing early software project estimates is an industry best practice, but creating those estimates is only half the battle when it comes to improving productivity. By continually keeping the pulse of a project—measuring performance against estimates and adjusting when necessary—teams can gain valuable insight into their software development processes. This insight can be leveraged in future development cycles, leading to more efficient production and a better bottom line.

Estimates are just the beginning. Project tracking, reforecasting, and post-project review are three valuable strategies teams can employ to monitor the development process and improve outcomes.

Strategy #1: Project Tracking

Current popular methodologies, like agile, promote metrics tracking, specifically with respect to size and velocity. But all developers, agile and otherwise, should take care to ensure that they're tracking the correct metrics, which can include:

- **Cumulative size produced**: this can include source lines of code, agile stories, screens and so forth;
- **Monthly effort**: the number of person months or person hours spent;
- **Staff**: the number of full time equivalents billing time to the project; and
- **Defects**: how many errors are discovered.

Data can be collected weekly or monthly, depending on the project's schedule, and, to gain the most insight, should be compared with the project's original estimates. Then, teams should try to figure out why deviations from estimates are happening. Is there an imbalance between management and developers, leading to a lack of skilled employees needed to complete the project? Has the team introduced new tools, methods, or team members? It can take time to ramp up on production if any of these variables have been introduced.

Strategy #2: Reforecasting

Software development cycles are rarely executed exactly to plan. Team members may have been pulled away to other projects, a large number of errors may have been found, or other variables may have been introduced. Any one of these situations will cause a deviation from a project's original

estimate. When teams examine metrics and notice that deviations are happening, it is best to reforecast, or re-estimate the project.

Certain project management tools use metrics to automatically recalculate project roadmaps. These tools can give new forecasts based on the data the team provides. It is certainly more advantageous to correct deviations early, rather than waiting until the end of a development cycle and risk missed project commitments.

Whether teams use prepackaged tools or their own estimation techniques, they should inform stakeholders of revised project plans as they develop. Reforecasting provides a window into the project's progress at regular intervals. As more data is collected and forecasts are continually refined and communicated, project transparency and accountability also improve.

Strategy #3: Post-Project Review

When a project is finally finished, teams are usually ready to power down, close the books, and take a break before diving into the next big thing. The project's level of success is often perceived as somewhat irrelevant—team members usually want to move on quickly.

However, taking time to do a post-project assessment is a great strategy to continuously improve team outcomes. Just like project tracking and re-estimating, post-project assessments provide the opportunity for a collective pulse taking.

In its most basic form, a post-project review gives team members a chance to evaluate what went well and what could be improved upon for the next project. Post-project assessments also give teams opportunities to assemble final sets of metrics that, when examined with data previously collected on the project, may provide more accurate estimates for projects going forward.

The Bottom Line: A Better Production Cycle

Strategies such as project tracking, reforecasting, and post-project assessment all help team members refine project estimates on the fly. When updated estimates are shared with stakeholders, communication among project team members and accountability are improved.

Over time, the process of collecting data, updating trends, and using trends to estimate future projects creates a circular process that creates more refined estimations over time and across multiple project cycles. More accurate estimates can translate into happier stakeholders, a more judicious use of resources and, ultimately, a better bottom line.

Adaptive Forecasting for Agile Project Tracking

Keith Ciocco

Before an agile project starts, many product owners will run an early release estimate. Once the activities get started, managers or scrum masters begin to track the progress. When they track, they usually include the person hours of effort and the number of user stories within each sprint. To accomplish these tasks, there are several agile tracking tools and methods in the marketplace that can be used.

But wouldn't it be great if the tracking and estimation process could be combined, using the actual tracked effort and user stories to run new and improved ongoing estimates at the release level? In fact, just such a method has been developed and successfully used on hundreds of software projects, and which will be briefly shown in this article. This type of adaptive forecasting has helped save time and effort by showing when a software release is headed down the wrong path. It can also help organizations avoid signing up to inflated resource planning numbers that cause many companies to waste millions of dollars at the release and enterprise levels.

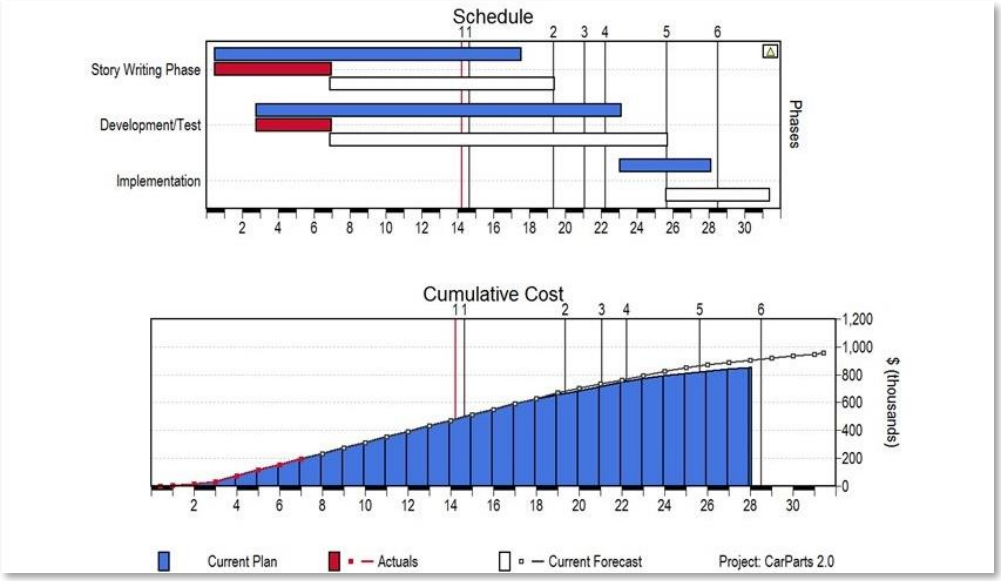


Figure 1. Delivery time and cost of a sample project.

The SLIM-Control® charts above (Figure 1) show a hypothetical project's planned schedule and costs (in blue) versus the actuals (red) and new forecasts (white). The focus is on capturing the total effort spent and actual work delivered each week, then using that information to generate mathematical models that produce new, empirically-based forecasts at the release level.

It is also important to see how actual data compares to the initial plan. In the SLIM-Control® charts below (Figure 2), there are plans versus actuals, as many tools show, but the advantage here is that these charts show specific control bounds. These customizable green and yellow indicators provide early notice of whether this project release has a good chance for success. This statistical analysis results in added credibility when negotiating targets and, because the caution signs are available early, there is still enough time to make a difference and at significantly less cost.

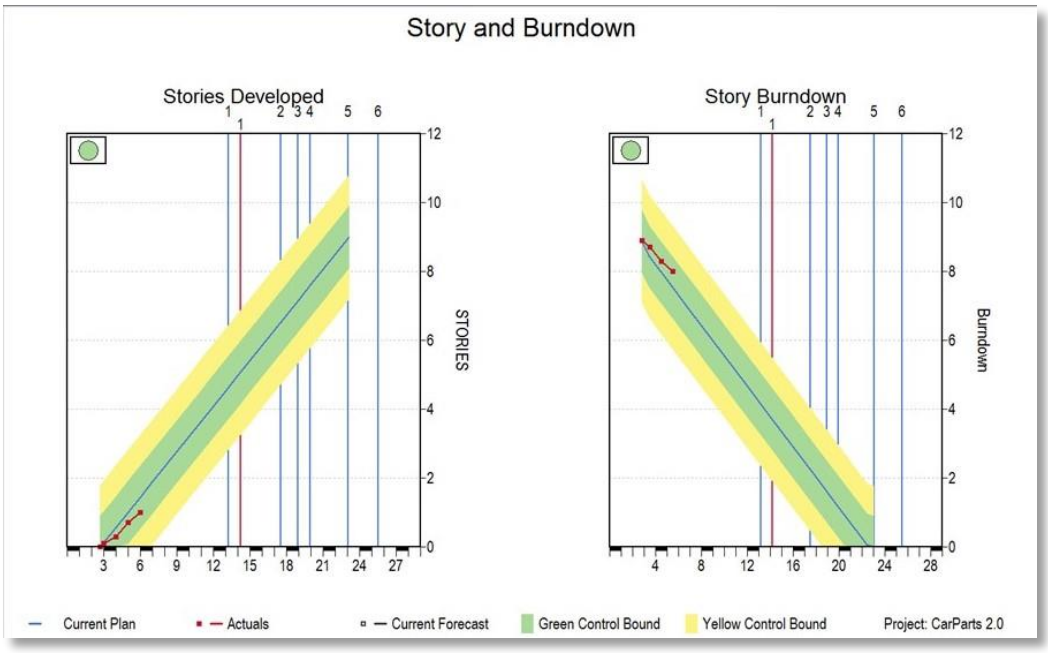


Figure 2. Example project's initial plan and actual performance displayed with control bounds.

At the very minimum, managers should be tracking agile projects. To be successful, however, they need the ability to determine if the initial plans are realistic. Project actuals then must be used to generate new forecasts at the release level, while also "grooming the backlog." This type of analysis will create added visibility and enable managers and stakeholders to more effectively negotiate new release and enterprise-level targets.

Why Should I Care about Actual Data; The Project Is Complete!

John A. Staiger, Jr.

"The game ain't over 'til it's over." - Yogi Berra

Baseball season will soon be here and, with apologies to the late Mr. Yogi Berra, “it’s like déjà vu all over again.”

Why would a project team or program management office (PMO) take the time and spend the resources to collect information about a project that was just completed? Isn’t this the time when victory is declared and everyone runs for the hills? In many cases, delving into what happened and what actual costs and durations were incurred can seem like an exercise in self-flagellation.

Historical data is arguably the most valuable input available in the software estimation process. While other inputs such as size and available duration or staffing can be seen as constraints, properly collected historical data moves the activity from the realm of estimating closer to “factimating.”

In some cases, project managers (PMs) use “engineering expertise” to develop an estimate. While this is a very valid estimating technique if properly performed, historical performance should also be a factor in the process. The fastest player at the 2015 National Football League (NFL) Scouting Combine was J.J. Nelson, a University of Alabama Birmingham wide receiver. His time in the 40-yard dash was 4.28 seconds. This was a real piece of historical data that can be used as a comparison or performance measure. However, if one asked Mr. Nelson if he could attain a time of 4.00, his answer might be tempered by the fact that his time was the absolute best that he could do. So why would a project manager or estimator not take into consideration previous performance when they develop estimates? The clear majority of software developers may be justifiably optimistic in their assumptions of team capabilities, requirements stability, and budget availability. However, most experienced PMs will admit that not much goes as planned.

As an example, Figure 1 below depicts a log-log plot of historical projects for one organization using size and duration as metrics. Mean values are shown as the solid blue line, while the dashed lines represent ±1 standard deviation. Data from completed projects are shown as black dots. The plot demonstrates that as size increases, duration increases, which should be no surprise to most PMs.

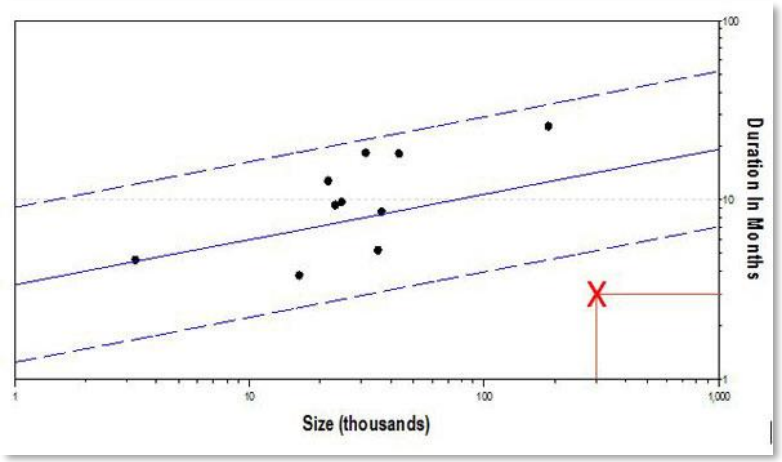


Figure 1. Log-log scatter plot of notional organization's historical projects using size and duration.

If a duration estimate for a project of approximately 120,000 size units is two months, as depicted by the red "x" in Figure 1, it should be obvious that this accelerated schedule is far shorter than the organization's historical performance. Of course, this historical view compares only duration to size, and an argument might be that the PM will just throw the "entire student body" at the project in order to shorten the duration. In this case, an analysis of historical staffing profiles by project size would be useful to determine if this approach has succeeded in the past. Concurrently, an analysis of historical quality performance could be completed to demonstrate that higher staffing profiles almost always generate markedly higher defect generation.

Additionally, if one is using the SLIM® suite of tools, statistically validated trend lines for many metrics are available, as is the ability to create trend lines similar to those shown in Figure 1 above. Trend lines are developed from historical data and present statistically bounded views of an organization's or an industry's actual performance. As depicted above, the relative position of an estimate against actual historical data can paint an interesting picture that affords analysts and decision makers a straightforward view of the validity of an estimate.

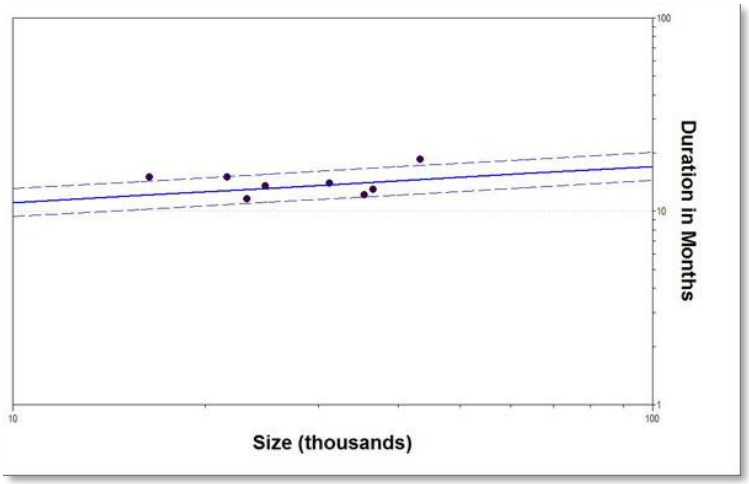


Figure 2. Example scatter plot of duration versus size (x-axis) showing trend line and standard deviations, in this case, with very little variation.

However, one should be cautious about historical data that looks "too good." In Figure 2 above, the variation in performance, illustrated by the standard deviation lines, is very slight across a wide range of size metrics. Taken at face value, this presentation would seem to indicate that the organization has tight schedule control across a range of project sizes. This may be a true statement, but this is not the hallmark of the average organization, and one might consider digging a little more deeply into the source of the data and its incorporation into any analysis tool.

Gathering historical data can be highly comprehensive or limited to certain core metrics. The data can be collected for the entire project or, preferably, by software development lifecycle "phase." Agile project managers also can and should collect data, if not by sprint, then certainly by release.

When beginning the data collection process, it is important to identify potential sources of data. This information can usually be found in the following artifacts:

- Software lifecycle methodology
- Project schedule chart used in briefings
- Microsoft Project® file, Clarity® / other PPM export, or similar detailed planning tool
- Requirements documents
- Staffing profiles
- Defect reports

At a minimum, it is advisable to collect:

- Size - in terms of what work must be accomplished, such as (but not limited to) source lines of code, user stories, epics, or function points
- Start date, end date (duration)
- Effort
- Peak staff
- Defects

Data collection is the foundation of project estimation, tracking, and process improvement. When historical data is collected across a program management office or enterprise, it is recommended that the data be stored in a data repository that can be accessed for estimating future projects or for individual project forensic analysis.

Establishing a repository of historical data can be useful for a variety of reasons:

- Promotes good record-keeping
- Can be used to develop validated performance benchmarks
- Accounts for variations in cost and schedule performance
- Supports statistically validated trend(s) analysis
- Helps make estimates "defensible"
- Can be used to bound productivity assumptions

One must also be careful in the collection and use of historical data. Consistency in data collection is important since currency values, size differences, "typical" hours per month values, and defect severity levels should all be normalized. Some of the problems linked to the use of poor-quality data include:

- poor project cost and schedule estimation
- poor project cost and schedule tracking
- inappropriate staffing levels, flawed product architecture, and design decisions
- ineffective and inefficient testing
- fielding of low quality products
- ineffective process change

In short, an organization's measurement and analysis infrastructure directly impacts the quality of the decisions made at all levels, from practitioners and engineers through project and senior managers. The quality (and quantity) of the historical data used in that infrastructure is crucial.

Those organizations fortunate enough to have licensed the SLIM® suite of tools or similar parametric tools already have all the resources needed to not only store the historical data, but to perform analysis of the data and build custom metrics regarding that data. In the case of SLIM®, the use of software metrics and data for estimating and controlling projects is made much easier by seamless interfaces to the estimation and control tools.

Full Circle Estimating

Douglas T. Putnam and Taylor Putnam-Majarian

This article originally appeared in the June 18, 2015, edition of the Projects at Work online journal, and is reprinted here with permission.

Suppose that after much consideration, an organization puts forth an estimate with the intention of carrying it out as a project plan. At this point, many estimators would believe that they are done. However, to maximize the benefits of the estimate, it will be necessary to track the project's progress through completion. While this may seem like a tedious process, project tracking allows for organizations to gain the greatest insight into their software development lifecycles.

To do this, estimators will want to collect data on the current actuals for the following metrics at regular intervals:

- Cumulative size produced – this metric is customizable and can include but is not limited to source lines of code, agile stories, screens, or similar units.
- Monthly effort – the person-months or person-hours expended
- Staff – the number of full-time equivalent personnel are currently billing time to the project
- Defects – how many errors were discovered

Depending on the length of the project's schedule, it is recommended to collect data on the actuals at the end of each month or, for shorter projects, at the end of each week. This data can then be compared against the original estimate to track the project's "in-flight" progress, as well as any deviations from the estimate. Using practices such as these to regularly maintain status can significantly increase accountability within the organization.

Additionally, tracking these various metrics over time can help identify areas for improvement within an organization. For example, if the project's actual data match the projected values from the original estimate in staffing and cumulative effort, but not cumulative size, this observation could indicate a couple of potential scenarios (see Figure 1). One possible explanation might be that the organization has staffed the project with the correct number of people at a given point in time, but has not appropriately accounted for the skills necessary to complete the project. An excessive number of management personnel may have been staffed for the project, resulting in a decreased staffing allotment for the developers. With fewer developers staffed or those lacking the skills needed for the project, producing code at the estimated rate would be impossible and deviations from the original estimate would be expected.

Another possible explanation for the decreased code production is that the actual productivity of the organization is, in fact, lower than originally estimated. There are several explanations for why this occurs. Introducing new tools, development methodologies, or team members impacts the development environment and can require additional time for developers to perform at the previous productivity level. The next section describes some potential solutions if this happens.

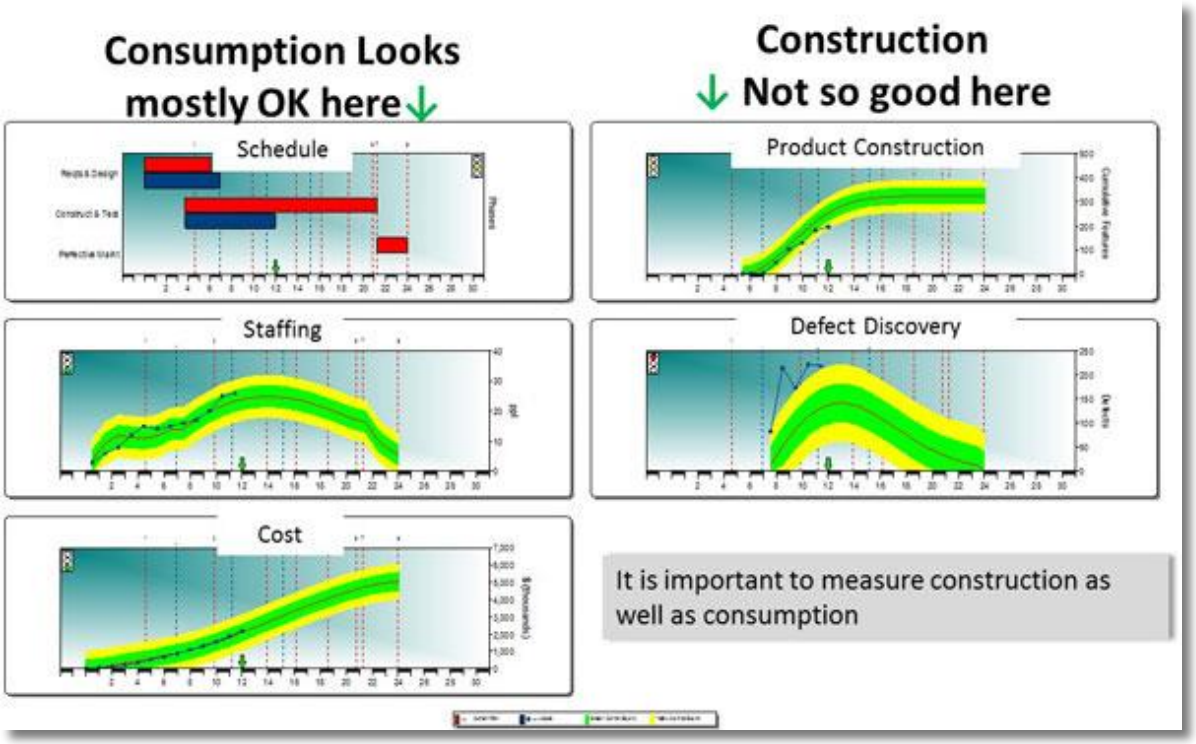


Figure 1. Tracking actuals against the planned values: consumption vs. construction.

Forecasting and Execution

Life is rarely predictable, and that holds true in software estimation as well. Even the best estimates cannot anticipate some of the curveballs that life throws at them. For instance, if several team members are pulled off the project and temporarily reassigned to another, it is unlikely that the project will progress as originally planned. In such situations, the best suggestion would be to reforecast the estimate.

Like a GPS, various parametric estimation programs can recalculate a new trajectory, or roadmap, if the project gets off track. They factor in what has already been done and re-estimate a new productivity based on the actual values entered. If a project drifts so far off course that the original plan is no longer feasible, a new plan can be implemented and briefed to stakeholders so that everyone has similar expectations (see Figure 2).

In Figure 2, the project's plan (in blue) is compared to the actual data collected in monthly intervals (in red). In January, the amount of code produced was slightly below the quota outlined in the plan, yet the cumulative effort expended was above the allotted amount. Additionally, they were finding more defects in the system than initially predicted. Since the project has begun to get off track from the original estimate, the wise choice would be to reforecast the project plan. The new forecast

is displayed in white and shows that, based on the current trajectory, the project should end about 2.5 months later and require 6,600 additional person-hours of effort than the original plan. Knowing this information early can allow time for orderly changes while the project is still underway.

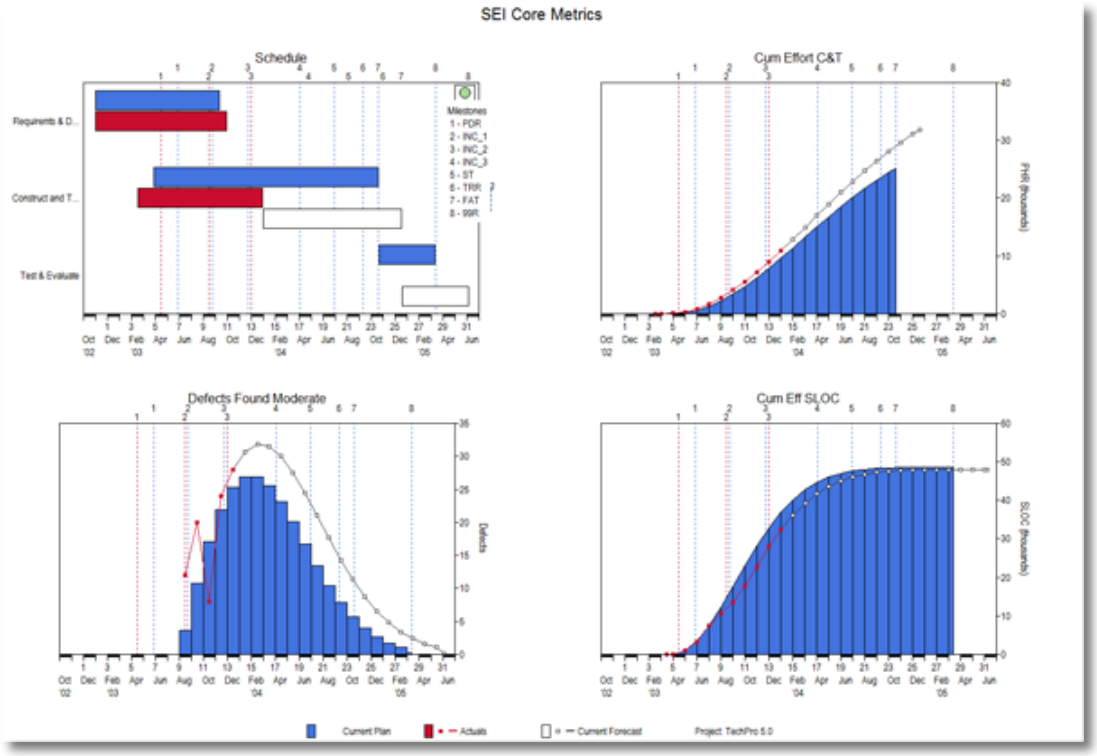


Figure 2. Forecasting chart.

One of the greatest benefits of this technique is that it provides transparency into the development process by showing the project's progress at regular intervals. Development teams know how much functionality must be delivered at each interval, and can plan accordingly. If a project begins to drift off course, it is much easier to determine this from the start and take measures to mitigate risk early. Additionally, as new data is added, the estimation program can reforecast the plan like a GPS would recalculate the road trip. If the new data shows improvements, it will be reflected in the forecast going forward, thus creating a refined learning system. Having this kind of insight and level of detail up front can dramatically improve accountability across the organization.

Post Mortem

Once a project has reached completion, it is very easy to move onto the next without a second thought. Regardless of whether the project went particularly well and the momentum is continuing, or if the project was so utterly painful that the wounds are still fresh, a post mortem assessment is always a valuable exercise. Post mortems provide opportunities for teams to discuss what went well and what areas could be improved. Additionally, post mortems allow teams to reconvene at the end of the project to collect the final data for the project repository. The data from the newly completed project can be entered into the estimation data repository, along with the rest of the completed project data. A new set of custom trend lines can then be created using the most recently added historical data, which should be used for more accurate estimates in the future. Over time, this

process of collecting historical data, updating trends, and using those trends to estimate future projects allows the estimation process to come full circle, refining itself each time.

Summary

While creating estimates is a great way to start a project off on the right foot, using parametric methods to track and monitor a project's progress provides even greater insight into the development of the project. Knowing the status of a project "in flight" and having the ability to share that with developers and stakeholders, when most can merely guess, is extremely beneficial and puts project managers in a much more powerful position. The project can be tracked all the way through completion, at which point a post mortem assessment can be done to collect data and further refine the estimation process. Using this model creates a refinement cycle whereby collecting accurate data leads to creating more accurate estimates.

In Agile, What Should We Estimate?

Dr. Andy Berner

The QSM® Agile Round Table was formed to provide a platform to brainstorm the role of estimation in agile environments and to chart a path toward better understanding for all stakeholders. This is the second of a planned series of articles resulting from this forum. A version of this article was originally published in Agile Connection in December 2016, and is reprinted here with permission.

Sometimes the highest priority is not enough.

"Ladies and gentlemen, we have reached our cruising altitude. You have experienced the new release of our avionics software. 'Takeoff' was the highest priority feature, and we loaded it on the plane as soon as it was done. As you just saw, it worked flawlessly. The next priority for the development team is 'Landing,' and the team will radio that software up to the plane as soon as it's potentially shippable. Don't worry, we have plenty of fuel, so sit back and relax, and thank you for flying Agile Airlines."

This is a joke, of course. But while no professional software organization would let passengers take off if they couldn't land, it illustrates that the answer to the question, "When will software be ready to deliver?" isn't as simple as "Just choose the highest priority features first, then you can deliver working software at the end of any sprint."

What Should We Estimate?

There's been a lot of confusing discussion in the agile community surrounding the question, "Should we estimate?" Often, however, confusion can stem from lack of clarity. So, it can be helpful to expand the question to make it, and thus the response to it, more useful and straightforward. Perhaps the question that should be asked is, "What should we estimate when we are using agile methods, and what choices will we make based on an estimate?"

The QSM® Agile Round Table was formed to discuss the role of estimation in agile environments. QSM® customers shared their questions, challenges, and experiences on the relevance and benefits of scope-based estimation in an agile environment. "What should we estimate?" was the first question addressed. Everyone agreed early on that there were two distinct meanings to the term "estimate" in the agile community, and that leads to confusion. On the one hand, the term "estimate" is used for sprint planning. This is trying to answer the question, "What stories should we develop over the next two weeks (or however long the time-boxed sprint will be)?" That's a fundamentally different question from the larger-scale question, "What duration and cost should I plan for, and how likely is it that the plan will succeed?" The group agreed that this larger-scale question was still important in an agile

environment, with participants wanting to understand how agile methods impacted the way they go about such estimates.

The first thing the group needed to understand was, "To what do we apply this larger scale question?" No one asks this about each sprint! "How long will a two-week sprint take" isn't a very interesting question. Traditionally, this question is asked about a project. In many agile environments, however, teams no longer organize work as projects in the traditional sense, so what is it that they are planning for? What plays the role of "project" that they should estimate?

To start answering that, the group decided it needed to look at some techniques commonly used by agile teams.

What Is Potentially Shippable Software?

Agile methods emphasize value and quality. To maximize value, backlogs are prioritized, and to the extent possible, stories are developed in priority order. Additionally, the team only "counts" a story if it's complete, and to be complete, it must be well tested. At the end of each sprint, the highest priority functionality that the team has had time to develop is working—the software is "potentially shippable." But to complete a story within a single sprint, broader epics and stories must be broken down into developer-sized bites--parts of the story that can be completed in a short period.

These techniques allow each iteration review to accomplish its primary purpose: get feedback on software completed so far, so managers can learn what needs to change going forward. This prepares them to develop another chunk in the next sprint, making progress on software that will deliver value to the eventual users.

What Is Consumable Value?

An inevitable consequence of breaking stories into developer-sized bites is that the smaller pieces developed in a particular sprint may not be the full features that users will need. That's ok: the remaining pieces will be developed in the coming sprints, and the team learns from feedback on the earlier bites. But until enough of the higher-level epics and stories are developed to make the software useful for its intended purpose, users cannot consume the software.

Partway through the group's work on a release of products at QSM®, the members sometimes propose an enhancement to meet a specific customer request. To avoid having to delay the release, team may propose "just enough" to meet that specific request. The team lead often criticizes such proposals as "half a feature." When customers use the enhancement, they will also expect related functionality. The enhancement has value, but cannot be consumed unless there are other enhancements as well.

It's easy to come up with blatant examples:

- ATM software that lets the user put in her card and enter her PIN, but she can't get money from the machine.
- ATM software that lets a user get money from the machine after validating his card, but he can't enter his PIN.

- Payroll software that can compute the paychecks, but can't print checks or make direct deposits.
- An airplane that can take off but not land.

Consumable value comes in all sizes, and a large scale consumable epic can often be broken down into consumable stories, some of which may be further broken down into smaller consumable stories. Some of those may turn out to be developer-sized bites that can be developed in a single iteration. But often, when a consumable story is broken into developer-sized bites to fit into a single sprint, the individual bites, while useful for review and feedback, will not be consumable by themselves. Over several iterations, the bites combine into a consumable meal. Consumable value is a collection of stories that provide enough value to satisfy users. This cannot arbitrarily be forced into a time box.

Keeping the "Viable" in the Minimal Viable Product

Making sure that enough value is delivered to users doesn't mean features can't be incrementally released over time. A large epic may be broken into consumable pieces, with some of those pieces released first, and enhanced over time. Eric Ries popularized the term "minimal viable product," and the related concept, "releasing in small batches," is looked highly upon by many agile organizations. But "the smaller the better" works only up to a point, which is why Reis didn't call it "Minimal Product." The product must be viable, in other words, provide consumable value. And Reis is very clear that what is viable depends on one's customers, one's competition, and one's goals for the product. (2011)

#YesEstimate to Get to Consumable Value

Since team leads expect it will take multiple sprints to deliver consumable value, it's worthwhile to estimate what it will take to deliver that value. That is, it makes sense to answer the compound question, "What consumable value does the team expect to achieve (described at a high level, not in detail), what duration and cost should the team plan for, and how likely is it that the plan will succeed?" Based on that, the team can propose a feasible plan that is both attainable and meets the company's business requirements.

This is very different from iteration or sprint planning, where the primary question is, "given where the team is now, and given the backlog it has in front of it, what developer-sized bites should the team include in the next time-boxed sprint?" While that may involve estimates of how big the individual stories are (Planning Poker, anyone?), it's fundamentally different from an estimate to achieve consumable value. The #NoEstimates movement has grown up around simplifying sprint planning through proper backlog grooming, including the breakdown of bigger stories into smaller bites and careful prioritization.

The purpose of estimates based on consumable value is to help make overall decisions, often decisions that must be made before development of specific stories even starts. Some of these decisions are:

- Where should limited resources be invested? Most companies have many good ideas for new software, enhancements to existing software, new platforms and architectures, and other types of value. Usually there are more good ideas than resources to carry them out. Choosing the collection of ideas to invest in requires balancing the benefits to be delivered

with the resources and time it will take to produce those benefits, then choosing among those ideas based on the combination of benefit and what it takes to deliver.

- What team or teams should be allocated to a particular idea and for how long? There's usually a complex tradeoff among the time it takes to realize benefit, the likely cost of the development to get the benefit within that time, and the risk of meeting that plan.
- What are the tradeoffs between benefit and the time and effort it takes to deliver and the cost of delay? Agile teams have learned to appreciate ideas such as "minimal viable product," "develop in small batches," and "weighted shortest job first;" delaying delivery has costs of its own. However, reducing scope to speed up delivery impacts the consumable value of the delivered product. These tradeoffs cannot be made arbitrarily, so considering multiple estimates for several different scenarios allows one to make informed plans, yielding results that are both minimal and viable.
- How can software delivery be coordinated with other organizations involved? Software development is rarely an island. Plans for software delivery must be coordinated with other initiatives and the organizations involved in them. Other organizations must plan based on when they can expect value from the delivered software.

Delivery of Consumable Value Comes in Many Forms

It used to be a lot easier, in theory, to answer the question "What should one estimate?" The answer was "turn a proposal into a project, and include an estimate of the duration and effort it will take to complete that project." Of course, it was never actually that simple in practice. This seemingly clear answer was based on the classic notion of a project, a temporary endeavor undertaken to create a unique product or service with a definite beginning and end, and software development is hardly ever that delineated. Many agile methods are designed to mitigate the problems encountered trying to stick to that definition, and many agile organizations no longer organize their work around the classic notion of project. How one's company organizes software development work depends on many factors. Here are a few ways that may be familiar, among the many variations seen in practice. Most of these and more were represented in the QSM® Agile Round Table.

- The organization is developing internal systems for their company. Thus, there is a unique deployed instance of the system in production (or perhaps a few, segmented by organization), though there may be multiple other instances for development and testing. The managers promote changes through the varying levels and eventually those changes are consumed by the users of the production system. The managers may have an agile team working independently on each system, or they may be using an "agile at scale" methodology to coordinate the work of many teams across many systems.
- The organization is a product company; it develops software to sell to its customers. It periodically releases new versions, perhaps fix packs, or perhaps major releases with innovative new features. It may bundle up several innovative features into an annual release. Its customers decide whether each new release provides enough value to them to justify consuming it.
- The organization is a system integrator or contract development shop. Its customers desire new or enhanced software, and it bids for that business.
- The organization contracts out software development. It needs to plan the software budget to meet its business goals, and it will evaluate bids for the development.

- The organization pushes out updates of its software to its subscribers when they become available (continuous deployment, like those updates one usually blindly accept on one's cell phone).

Some of these are like classic projects, others much less so. The relationship between delivering consumable value with software and "a temporary endeavor with a definite beginning and end" is much more tenuous than project and portfolio planners used to think it was. Some may argue it was always so, but it is now being explicitly incorporated into methods.

Set Goals for Delivering Consumable Value and Estimate What It Takes

So, if one doesn't have projects to estimate, and since delivery can take so many forms, how does one make the decisions about what to invest in and the other related decisions? The specifics and the terminology depend both on the business and the methodologies used, but in all cases, one can set a goal for a collection of features that will provide consumable value, and estimate what it takes to reach the goal. There will no doubt be multiple possible ideas for such goals, including variations on what features should be included (different possibilities of what is simultaneously minimal and viable), and multiple estimates for each, trading off time and cost. This provides information to make decisions about which goals to invest in, where to apply resources, how to plan related initiatives and the other decisions needed to make to move forward with development.

These goals should be specific enough to estimate and make decisions about. "Improve the user's experience" is potentially the vision for software investments, but is likely too vague to determine concrete steps to reach the goal. On the other hand, one should not fall into the "big upfront requirements" trap. It is not necessary to define all the details of features in order to make decisions about the goals, and using agile methods to allow those details to emerge over the course of development will help "build the right product."

The way in which software is delivered can affect how goals are mapped to delivery. If new features are pushed to production, the features may be pushed when the goal is reached. Some features may even be pushed to production that can stand alone before the entire goal is reached (this may involve the development technique of "feature branches" to ensure consistent code). If instead there are annual releases, multiple goals may be bundled up into a single release, estimating whether it's plausible to reach that combined goal in the time allotted. If an agile at scale methodology is used, multiple goals may be planned for to be developed concurrently, using the estimates to vary the resource allocation over time; the actual releases of the software to production may be planned based on the specific delivery capabilities and methods.

Note, however, that when a goal is said to be a collection of features, the word "feature" is used in a generic sense. The development methodology may have a specific meaning for "feature," in which case different term should probably be used. It's important to realize that features come in all sizes. They may be very specific (send e-mail confirmations whenever the shipping status of a purchase changes, fix the bug in a specific calculation), or they may be higher level (revamp the reporting capabilities), or anything in between (allow a car rental to be added to hotel reservations). Consumable value, after all, is in the eye of the eventual consumer as well as the company's business needs.

Using the Estimate with Agile Methods: Is an Estimate a Commitment?

This is an old question, of course. Software teams have always been reluctant to give an estimate, which inherently has a degree of uncertainty, because management often takes the cost (which primarily affects staffing levels) and duration as firm commitments. This can lead, as everyone knows, to nightmare "death marches" and cancelled projects when the reality of development doesn't match the commitment. Part of the reason for this is poor estimation techniques that underestimate. But even with sophisticated estimation techniques based on history, estimates inherently have a degree of uncertainty.

In the QSM® Agile Round Table, the group discussed how organizations handle the question of uncertainty in an estimate. Bonnie Brown, Hewlett Packard Enterprise, said some teams handle estimation risk by putting the burden on the product manager to manage scope within the committed time. They estimate cost and duration based on the understanding of the planned scope (the "goal" in the terms of this article), and commit resources for the duration of that estimate, which determined the number of time-boxed sprints. Then, as Bonnie said, the product manager would decide on the priorities at each sprint and could get "whatever scope they want within that duration." This fits with what some agile practitioners say, "In agile, we time-box duration and vary scope." However, other members of the roundtable said that didn't work for their customers. The customer expected the result to meet the original goal. As the work progressed in an agile fashion, the customer certainly refined and modified the original scope, but the customer did not accept the notion that "all the scope that fits the original estimate" was sufficient if it did not meet the goal. Consumable value is in the eye of the beholder.

Agile Methods Increase the Uncertainty

Perhaps the biggest impact of agile methods on early, goal-level estimation is that no one even pretends to have a detailed description of the scope when estimates are needed. The details of the scope emerge throughout the work, as large epics are refined into development-sized bites. There is no upfront signed-off requirements document on which to base an estimate. Of course, part of the rationale for agile methods is that the signed-off requirements document never really stuck anyway, or if it did, the result was one that nobody liked. Emergent requirements based on feedback at each sprint is the best way to "build the right product."

But how, then, can an expected duration and cost from a scope only defined at a high level be estimated? In estimation terms, how can the size of the goal be determined? The QSM® Agile Round Table addressed these questions, and that will be the topic for future discussions.

References

Reis, E. (2011). *The Lean Startup*. New York, NY: Crown Publishing Group (Random House).

Understanding Quality and Reliability

Doug Putnam and Taylor Putnam-Majarian

This article originally appeared in the March 23, 2016, edition of the InfoQ online journal, and is reprinted here with permission.

One of the most overlooked but important areas of software estimation, measurement, and assessment, is quality. It often is not considered or even discussed during the early planning stages of all development projects, but it's almost always the ultimate criteria for when a product is ready to ship or deploy. Therefore, it needs to be part of the expectation-setting conversation from the outset of the project.

So, how can one talk about product quality? It can be measured in several ways, but two, in particular, give excellent insights into the stability of the product. They are:

1. The number of defects and errors discovered in the system between testing and actual delivery, and
2. The Mean Time to Defect (MTTD), or the amount of time between errors discovered prior to and after delivery to the customer.

The reason these two measures are preferred is that they both relate to product stability, a critical issue as the release date approaches. They are objective, measurable, and can usually be derived from most organizations' current quality monitoring systems without too much trouble.

Generally, having fewer errors and a higher MTTD is associated with better overall quality. While having the highest quality possible may not always be a primary concern for stakeholders, the reliability of the project must meet some minimum standards before it can be shipped to the customer. For example, experience has shown that, at delivery, most projects are about 95 percent defect free after running for about a day without crashing.

Another good rule of thumb is that the software typically will be of minimum acceptable reliability when testers are finding fewer than 20 errors per month. This applies to both large and small applications. In other words, the product will run about an eight-hour workday. Of course, this rule of thumb is mostly applicable for commercial IT applications. Industrial and military embedded applications require a higher degree of reliability.

The Rayleigh Defect Model

One approach to attaining optimal quality assurance is to use the Rayleigh function to forecast the discovery rate of defects as a function of time throughout a traditional software development

process. The Rayleigh function was discovered by the English physicist Lord Rayleigh in his work related to scattering of acoustic and electro-magnetic waves. A Rayleigh reliability model closely approximates the actual profile of defect data collected from software development efforts.

The Rayleigh equation can be used to predict the number of defects discovered over time. It can be formulated to cover the duration of time from the High-Level Design Review (HLDR - High-Level Design is Complete) until 99.9 percent of all the defects have been discovered. A sample Rayleigh defect estimate is shown in Figure 1.

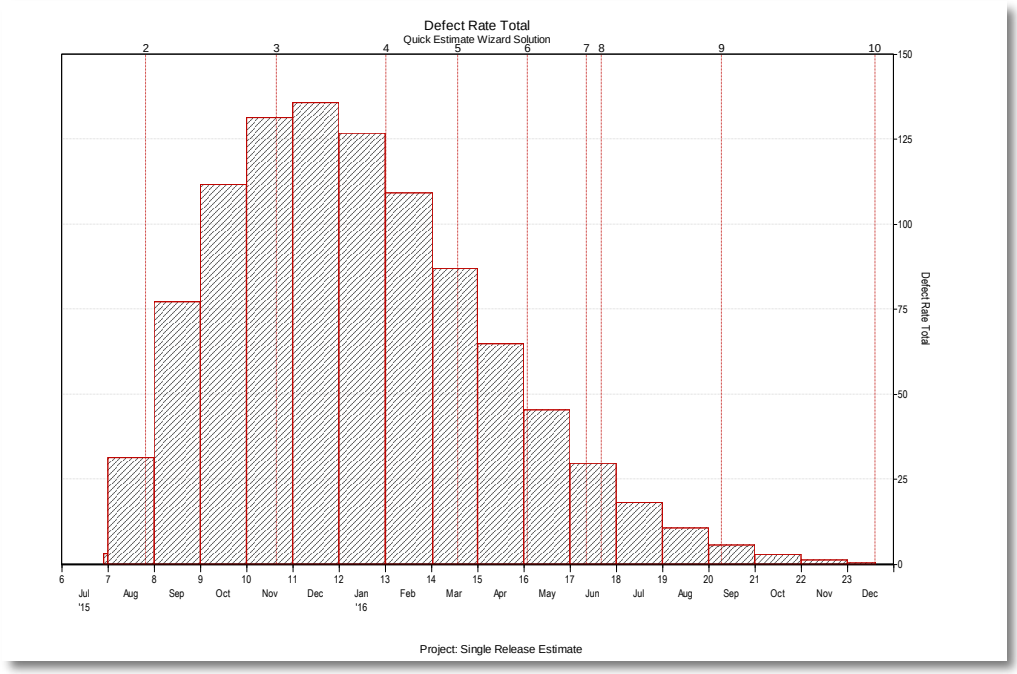


Figure 1. Sample Rayleigh defect estimate.

Note that the peak of the curve occurs early in the build and test phase. This means that many the total defects are created and discovered early in the project. These defects are mostly requirements, design, and unit coding defects. If they are not found early, they will surface later in the project, resulting in the need for extensive rework.

Milestone 10 is declared to be the point in time at which 99.9 percent of the defects have been discovered. Less than 5 percent of the organizations with whom QSM® has worked actually record defect data during the detailed design phase. **Industry researchers claim that it can cost 10-100 times more to fix a defect found during system testing rather than during design or coding** (Boehm, 1987; McConnell, 2001)**,** so one could make a compelling case to start measuring and acting earlier in the process**.**

Simple extensions of the model provide other useful information. For example, defect priority classes can be specified as percentages of the total curve. This allows the model to predict defects by severity categories over time, as illustrated in Figure 2.

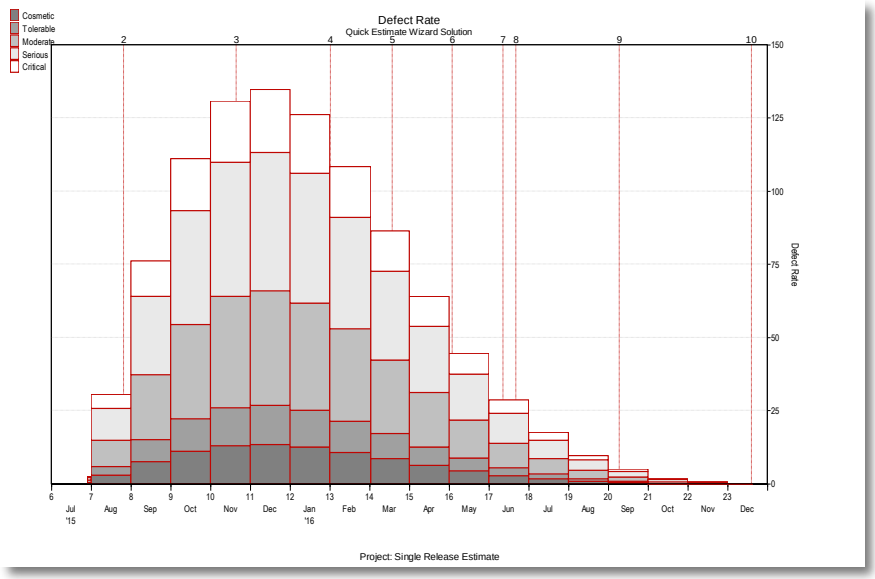


Figure 2. Rayleigh defect model broken out by defect severity class.

A defect estimate could be thought of as a plan. For a particular set of conditions (size, complexity, efficiency, staffing, etc.), a planned curve can be generated. A manager can use this as a rough gauge of performance to see if her project is performing consistently with the plan and, by association, with comparable historic projects. If there are significant deviations, this would probably cause the manager to investigate and, if justified, take remedial action.

Figure 3 shows how the defect discovery estimate can be used to track and compare actuals. Obviously, the actual measurements are a little noisier than the estimate, but they track the general pattern nicely. They also give confidence that the error discovery rate will be below 20 per month, the QSM®-recommended minimum acceptable delivery criteria, at the scheduled end of the project.

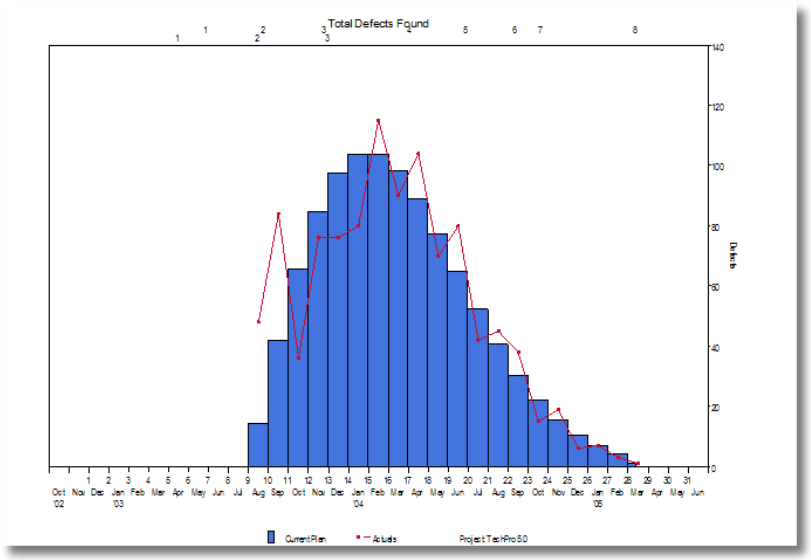


Figure 3. Defect discovery rate plan with actuals plotted.

Defect Model Drivers

Since 1978, QSM® has collected data on over 10,000 completed software projects. Analyses using this data have found that there are specific inputs that determine the duration and magnitude of the Rayleigh defect model. The inputs enable the model to provide an accurate forecast for a given situation. There are three macro parameters that the QSM® model uses:

- Size (new and modified)
- Productivity Index
- Peak Staffing

These driving factors impact the defect behavior patterns seen with regards to software projects. The next sections will examine these driving factors in more detail. Unless otherwise noted, these findings are based on analyses conducted using data from the QSM® database.

Size. Historically, as project size increases, so do the number of defects present in the system (see Figure 4). Stated simply, building a larger project provides more opportunities for developers to create system defects, and requires more testing to be completed. This rate of defect increase is close to linear.

Similarly, as size increases the MTTD decreases. This is due to the increased number of errors in the system, which often happens due to the communication complexities that are inevitably introduced through larger teams. With more errors in the system, the amount of time between defects decreases. As such, larger projects tend to have lower reliabilities because there is less time between defects. These trends are typical for the industry.

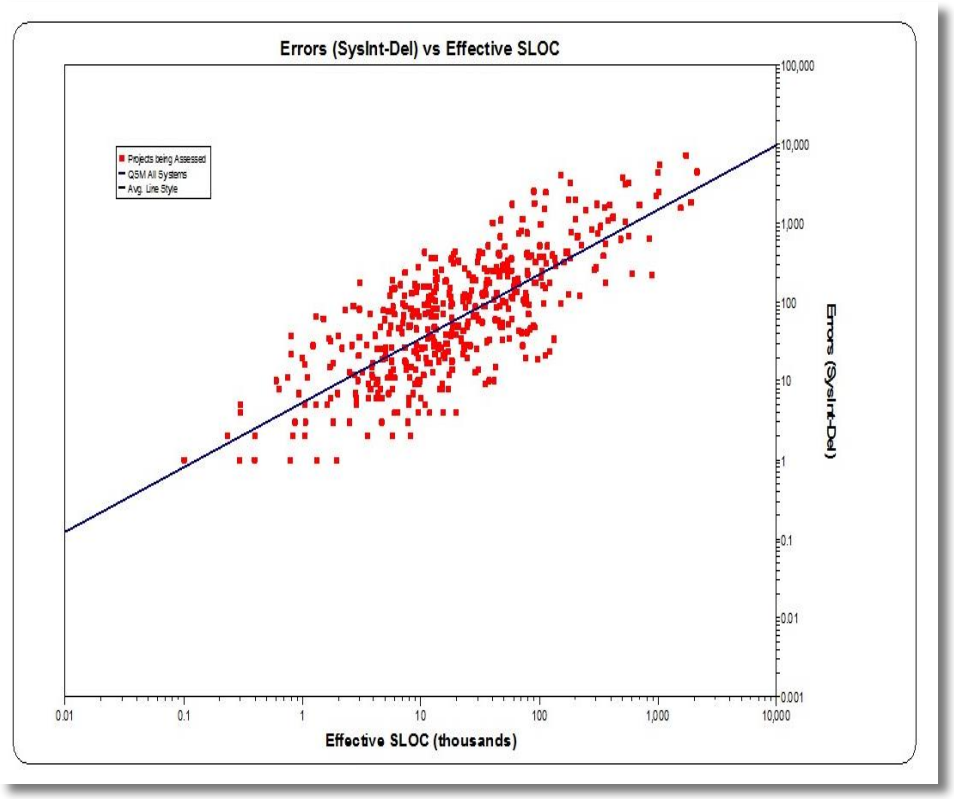


Figure 4. As size increases, so do the number of defects.

Productivity Index (PI). Productivity also tends to have a great impact on the overall quality of a system and can be measured via the PI, which is a calculated macro measure of the total development environment. It embraces many factors in software development, such as management influence, development methods, tools, techniques, and the skill and experience of the development team. It also accounts for application type and complexity, process factors, and reuse. It uses a normalized scale ranging from 0.1 to 40, where low values are associated with poor environments and tools and complex systems, and high values indicate good environments, tools and management, and well-understood projects.

Historic data has shown that the number of defects discovered exponentially decreases as the PI increases. Figure 5 shows the cumulative number of defects discovered for the similarly sized software application using two different PIs (*17 and 21 respectively*). The project with the higher PI not only delivers the application nine months faster, but also includes about 360 fewer errors in total. It makes sense that when development teams improve, they tend to make fewer errors to begin with, thus decreasing the number of errors found during testing. Operating at a higher productivity level can drastically increase software quality.

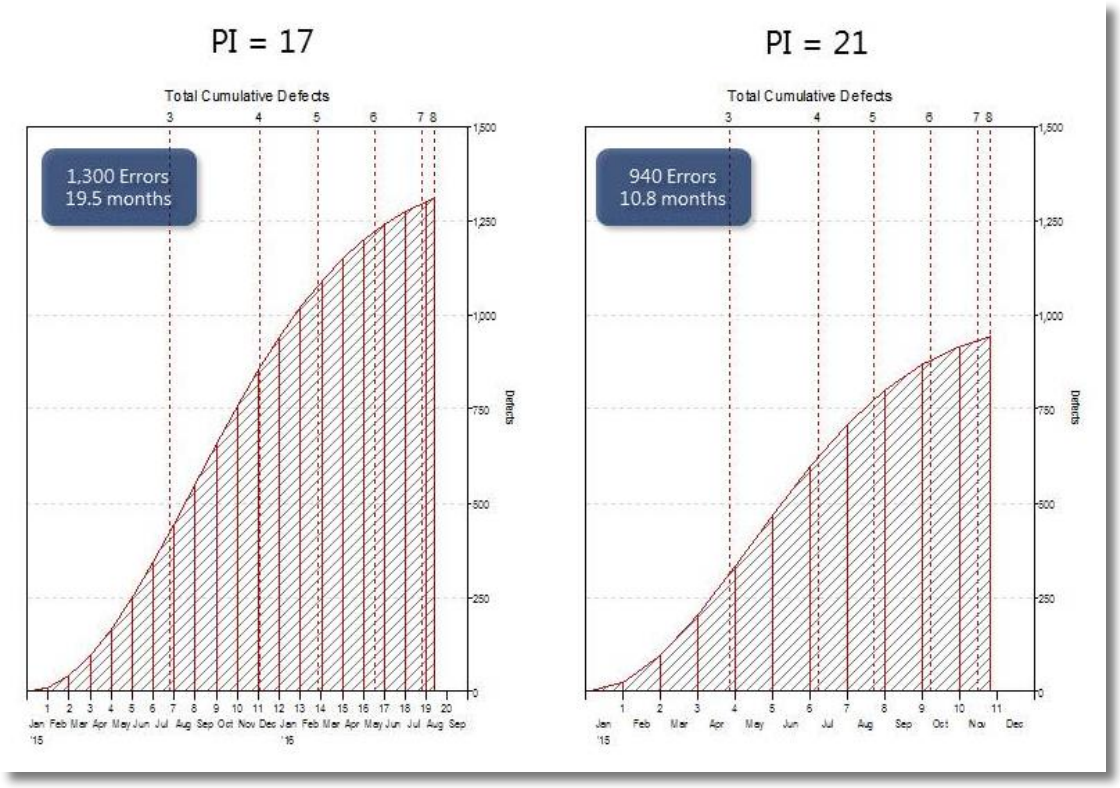


Figure 5. Comparison of defects produced for the same sized system, using different PIs.

Staffing. The size of the development team also can impact the quality of a system, as larger teams tend to produce more errors than smaller teams. As shown in Figure 6, when comparing a large team (red) with a small team (gray) at the same project sizes, the small teams produced between 50-65 fewer errors than large teams at all project sizes. Additionally, they paid little, if any, schedule penalty and used significantly less effort. This finding can be especially useful when looking to identify areas of waste within an organization, because it shows that adding resources to a project does not always improve its quality.

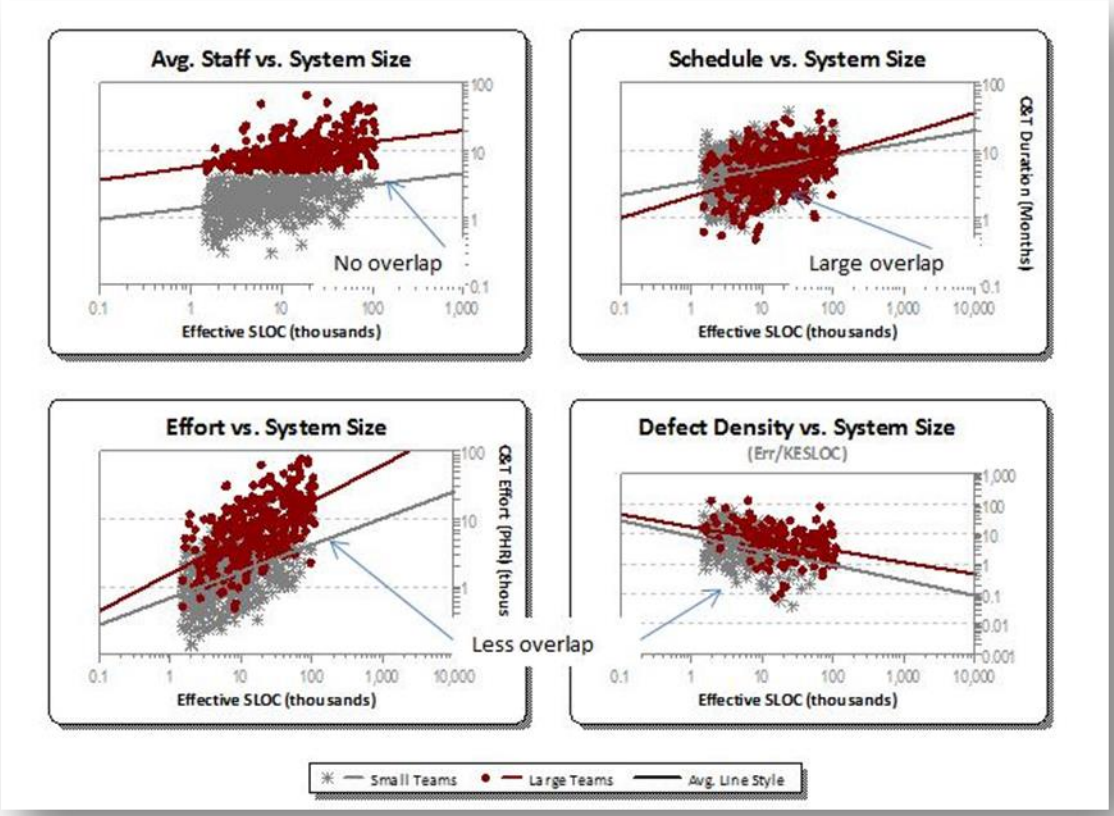


Figure 6. Small teams produce fewer errors than large teams at all project sizes.

Summary

Software reliability and quality are two areas that should be addressed in the expectation setting and project negotiation stages of a project. Quality reflects how well software complies with or conforms to a given design, based on functional requirements or specifications, while reliability pertains to the probability of failure-free software operation for a specified period in a particular environment.

As a leader, there are several strategies that can be used to improve reliability and quality. Keep the developed product size as small as possible, use smaller teams of people, and make regular investments in the environment to improve the efficiency and effectiveness of the development shop. All of these actions will pay reliability and quality dividends.

References

Boehm, B. (1987). Industrial software metrics top 10 list. *IEEE Software*, 84-85.

McConnell, S. (2001). From the editor: An ounce of prevention. *IEEE Software*, 18(3), 5-7.

3. BEST PRACTICES

"If you have an apple and I have an apple and we exchange these apples, then you and I will still have one apple. But if you have an idea and I have an idea and we exchange these, then each of us will have two ideas."

– *George Bernard Shaw, Anglo-Irish playwright and winner of Nobel Prize in Literature*

"To improve is to change; to be perfect is to change often."

– *Sir Winston Churchill, Two-time British Prime Minister and winner of Nobel Prize in Literature*

"Good judgement comes from experience, and experience comes from bad judgement."

– *Frederick P. Brooks,*

A Business Leader's Primer for Overseeing Successful Software Projects

Donald Beckett

This article was serialized and originally appeared in several editions during Nov 2016 of the Projects at Work online journal, and is reprinted here with permission.

Overview

Top-tier business leaders are not involved in the day-to-day management of software projects. However, the success or failure of these directly affects their ability to perform their jobs effectively. This primer outlines steps leaders can take that will dramatically increase the success rate of their organizations' software projects, without becoming mired in the details of directly managing them.

The Dilemma

A business leader's job is running an enterprise. Normally, this entails making decisions that affect strategic direction, policies, and profitability. In this era, computer software is the connective fiber that allows an organization to compete and to thrive. When software problems arise, they degrade the organization's ability to function optimally. As such, preventing software problems and dealing with them effectively when they do arise are top enterprise priorities. Here, business leaders face a dilemma: while software is critical to the enterprise's success, they do not have time to be software experts making the day-to-day decisions that affect development and maintenance, while at the same time running an enterprise. Even if they have a background in information technology, software technology evolves so quickly that their knowledge is not current. No training, no seminar, no book can overcome this problem. Outside of quantum physics, one simply cannot be in two places at the same time.

That is the bad news, but there is good news, too. Business leaders can establish procedures and practices that will help their organizations' projects succeed. Likewise, they can also put into place ones that will increase the failure rate. It's important to know the difference. This primer focuses on what they can do to promote the success of software development and enhancement projects in their organization. This is no panacea and does not guarantee that every software project will succeed. But, if success is measured in money saved or by a dramatic drop in the percentage of projects that miss their schedule and budget plans, there are decisions that a business leader can take that will promote this change.

What Is a Successful Software Project?

There are four characteristics that define a successful software project. They are:

- It completes on time.
- It meets budget constraints.
- It fulfills its business objectives.
- It meets quality objectives.

The first two, time and budget constraints, can be answered with a simple yes or no. Business and quality objectives need to be clearly defined to determine if the software meets them (frequently, they are simply assumed and not clearly defined).

The Laws of Software Development

There are specific processes and procedures that a business leader can establish that will promote successful software projects. Before embarking on these, it is crucial that he or she understand the laws of software so that these can guide their development. Does the term "laws of software development" seem contrived? It shouldn't. Software has been studied for 40 years now, perhaps beginning with Frederick Brooks' *The Mythical Man-Month*. There is a wealth of knowledge of how software projects behave and what works and what does not. Failure to acknowledge and obey these laws, whether deliberately or through ignorance, is the principle cause of software project failure. Here is the short list.

Law 1. For any software project, there is a minimum development time. Just because a business leader wants or needs a project to complete in five months, it won't if the minimum development time is six. There is no way around this. Rationalization and wishful thinking will only postpone the day of reckoning. Back in the 1970s, Brooks identified lack of sufficient schedule as the primary cause of project failure. It still is. The solution to this is to plan for enough time to complete the work. How to determine both the minimal and optimal development times will be addressed presently.

Law 2. Schedule and cost/effort do not trade off evenly. For any development effort, there is a range of schedules and costs that are feasible. However, the more aggressive the schedule, within, of course, the boundaries of what is possible, the more it will cost. And not just a little bit more, either. The normal practice for reducing schedule is to add additional staff. Through no fault of their own, the new staff will take time to come up to speed, increase the communication complexity within the project, and decrease the productivity of the existing staff, who must train and coordinate with them if they join the project when it is already underway. If they are planned into the project from the beginning to reduce the schedule, the project will suffer the burden of having too much staff to work efficiently. In both cases, the closer the project is planned to the minimum development time, the more it will cost. The quality of the software will suffer, too. Unfortunately, when projects scramble to meet deadlines, sufficient testing finds itself on the chopping block. And a project that meets cost and schedule constraints, but shortchanges quality, only creates problems that will be present for a long time.

Law 3. Projects grow. When a software project begins, it has a schedule and budget that are based on delivering certain requirements. Normally, both schedule and budget have been pared down and are optimistic. What happens next compounds the problem. Between the time a software project begins and when it is delivered, it grows. The amount of functionality it delivers increases. Why is this so? There are several reasons. First, at the outset of a project, when budget is allocated

and schedule determined, its requirements are fairly high level. As these are refined and coding begins, things that need to be addressed appear that inevitably increase the project's scope. Second, new requirements are frequently added to a project, often without adjusting the budget and schedule. The net result is that a project that already has trouble meeting its schedule and staying within budget now has more to do than was planned, with no additional time or money. There is a fairly simple solution to this, which is to build contingency into both schedule and budget while enforcing a change management process that evaluates every request for its impacts and adjusts schedule and budget accordingly. So, how much do projects grow? A 2007 study by QSM® found that project scope increased on average by 15%, schedule by 8%, and cost/effort by 16% (Beckett, 2007; Beckett, 2008). Simply building these contingencies into the project from the outset will eliminate many issues. This is not to say that one should not strive to do better; one should. But allow for the fact that sometimes one won't.

Law 4. Small teams are better. Large software projects do not require large (and expensive) development teams. The analysis is conclusive on this. Smaller teams deliver superior quality software at less cost in about the same time as do large teams. (Beckett, 2013b) Figure 1, based on a study of over 2,200 completed software projects, clearly illustrates this. In it, the vertical bars represent the median schedule for the smallest to largest staffing quartiles for various-sized projects.

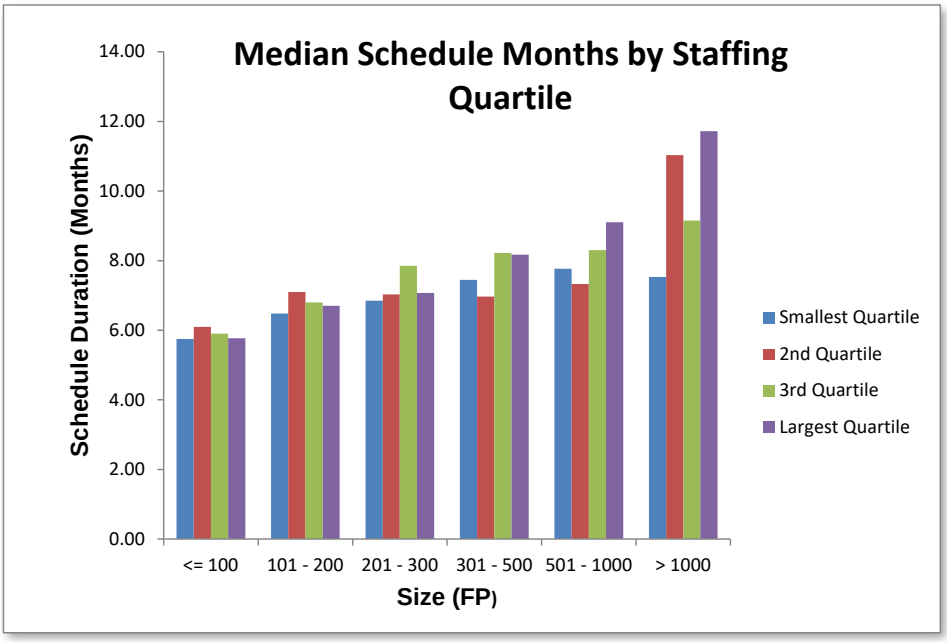


Figure 1. Schedule duration versus staffing quartiles.

Note that larger teams do not succeed in reducing schedule, even though they normally represent an effort to accomplish exactly that. All they succeed in doing is being more expensive. How to determine the appropriate project staff will be addressed later in a case study.

Law 5. Look before you leap: allow sufficient time and effort for analysis and design. When projects are faced with "aggressive" schedules, there is an almost irresistible pull to jump right in and start producing software code. This is wrong. In fact, it is exactly the opposite of what should be done.

In the past decade, QSM® has twice compared projects that expend more than average cost/effort on analysis and design with those who used less. The results leave no room for doubt. The projects that spent an above average percentage of effort (cost) in analysis and design

- Completed sooner,
- Cost less,
- Were more productive, and
- Had fewer defects.

Now, dedicating time and effort to analysis and design won't solve the problem of an impossible schedule (see Laws 1 and 2). However, incorporating at least 20% of planned project effort into analysis and design pays big dividends. Table 1 summarizes the results from the more recent study that compares projects that allocated more than 20% of their effort to analysis and design to those that didn't (Beckett, 2013b).

Table 1. Metrics comparisons at 20% design effort point.

Comparison at 20% Design Effort		
Medians		**% Difference**
PI <= 20%	11.04	
PI > 20%	14.19	29%
FP/PM <= 20%	6.20	
FP/PM > 20%	7.93	28%
Duration <= 20%	7.23	
Duration >20%	6.20	-17%
Total Effort <=20%	22.59	
Total Effort > 20%	20.29	-11%
Average staff <= 20%	2.34	
Average staff > 20%	2.50	7%
FP size <= 20%	157.00	
FP size > 20%	171.00	9%
Defects <= 20%	20.00	
Defects > 20%	19.50	-3%

What CAN Be Done to Promote Successful Software Projects?

When properly implemented and adhered to, the following actions, which can be initiated by business leaders, will improve the predictability of the organization's software projects, while reducing unpleasant surprises and cost.

Get a good chief technology officer (CTO). This is almost self-evident. He or she is the one who should be the focal point for software development; not the business leader.

Determine the organization's capabilities and plan within them. There is an old myth about King Canute who was so impressed with his own abilities that he believed that everybody and everything would do as he commanded. He had his throne set on the seashore at low tide and commanded the sea to stay out. It didn't work. Organizations that do not know their capabilities are very much like the good king: they base their decisions on their desires. Often, this backfires. So how can these capabilities be determined? The answer is simple, yet it forms the crux of how to promote successful software projects: **they must be measured, their data must be collected and analyzed, and decision making must be based on the results.** Organizations have patterns in their software

development: how they staff projects, whether they emphasize cost or schedule, and the overall quality of their work. These patterns have a strong inertial pull and define an organization's capability to produce software. They can be changed over time, but only if the underlying reasons for their existence are altered; not by executive fiat.

Measure for Success. Software projects leave a trail that can be measured and used to determine current capabilities and plan for ways to improve them. What a senior executive must do is ensure that this information is collected and used, which provides the basis for decision making. So, what is this forensic evidence that every software project creates? Every software project:

- Takes place over time. This is **project schedule.**
- Requires staff that needs to be paid. These are **effort** and **cost.**
- Creates an end product. The measurement of this is **project size.**
- Experiences glitches both in development and the end project. These are **defects.**

This seems like a simple list, and it is. However, analysis of this data from the organization's projects provides the basis for determining minimum development time, optimal staffing, and schedule, and allows the benchmarking of capabilities against other organizations. It will allow project plans to be distinguished between those that are feasible and those that are not. This does not just happen because it is a good idea. Three things are required. Two of them are the responsibility of business leaders.

- A small measurement group within the company that collects, stores, and analyzes the data
- A commitment to establish processes to collect and analyze project data and ensure that they are adhered to
- A commitment to base project decisions that involve staffing, cost, and schedule on what the data say; not on wishful thinking

Establish a Measurement Group. It is a dirty secret, but true nonetheless: many software measurement programs fail. To be successful, it is important to identify potential pitfalls and how to avoid or ameliorate them. Here are some of the more common reasons for failure and suggestions for addressing them.

Ensure support from senior management. Without this, a measurement program is doomed from the outset. If management does not adequately fund the effort, establish the necessary processes and procedures, provide justification for these to those tasked to follow them, train employees on how to follow them, and establish and follow through on consequences if they are not adhered to, the program will thrash around for a while and ultimately end up being cancelled as an unnecessary expense that provided little value.

Underestimate resistance. Resistance is very much like the tide in the King Canute analogy: people and organizations are very accustomed to the way they do things and feel threatened by change. Most resistance will not be overt; but it is there, nonetheless, and must be dealt with. A common source of resistance is that employees believe that a measurement program will be used to evaluate them. Thus, it becomes an enemy. When they believe this, project data will be manipulated to paint a good picture of their performance, which, of course, invalidates the data. There is a compelling reason not to use software project data for performance evaluation: it doesn't provide accurate insight into individual performance and any decisions regarding performance based on it are inaccurate and invalid. What it does is provide a basis for evaluating project performance and whether the processes and procedures in place are helping or harming the organization.

Performance management is a valid organizational concern: one that should always be separated from a software measurement program.

Too much, too soon. It is leadership's role to address the two concerns listed above. The following reason for failure is one that both leadership and the measurement group need to concern themselves with. When instituting a software measurement program, it is important when beginning to make it as unobtrusive as possible. When leaders see the possibilities inherent in software measurement for improvement and profitability, there is often a desire to jump in. The software measurement team, desiring to demonstrate its value, may be all too willing to oblige. The problem is that transforming an organization from one that bases its software project decisions on desires and hunches into one that bases them on proven, measured capabilities requires time and effort. Here are some ways to avoid overwhelming employees.

- Take advantage of processes already in place. The current time tracking procedure is an excellent place to obtain cost and effort data without requiring additional work from employees. It can also provide information on when projects begin and end (schedule). Some work may be necessary to massage it into a form that the metrics group can use, but this is a whole lot simpler than implementing a completely new process. Likewise, if there is a defect tracking system and/or a ticket system for recording problems and change requests, take advantage of them. Most of the information required for data driven management already exists; it just hasn't been organized, analyzed, or used.
- Start small and build on success. With cost/effort, project duration, defects, and project size, an amazing amount of project analysis can be performed. Rather than attempting to do everything at once, select a specific "pain point" or problem of the organization and focus on it. A lot will be learned from this that will help make solving the next issue proceed much more smoothly and help resolve a particularly important issue while not overwhelming the organization in the process.

Use Software Project Metrics. Planning based on actual performance is a core principle of any successful enterprise. Try imagining sales, revenue forecasting, budgeting, or inventory management without it. Managing a software portfolio and planning for ongoing development activities should be no different. Earlier in this primer, three problem areas for software projects were identified: budget (cost), schedule, and staffing. Here it is shown how software measurement can be used to address these. Measurement by itself does not resolve any one of these; but it does provide the basis upon which informed decisions can be made.

The ancient philosophical imperative of "know thyself" applies to organizations, too. Here are some examples of how software measurement can be used to determine capabilities, to determine whether project plans are feasible, and what trade-offs could be made if they are not. For these examples, a series of trends based on an organization's historical project data have been constructed. Let's begin with what management wants: a project that completes in five months with an average staff of no more than 10 persons. The following graphic (Figure 2) contains two scatterplots, each of which has trend lines based on the organization's history. The blue dots are the completed projects which the scatterplots are based upon. A good (feasible) plan will be consistent with historical performance.

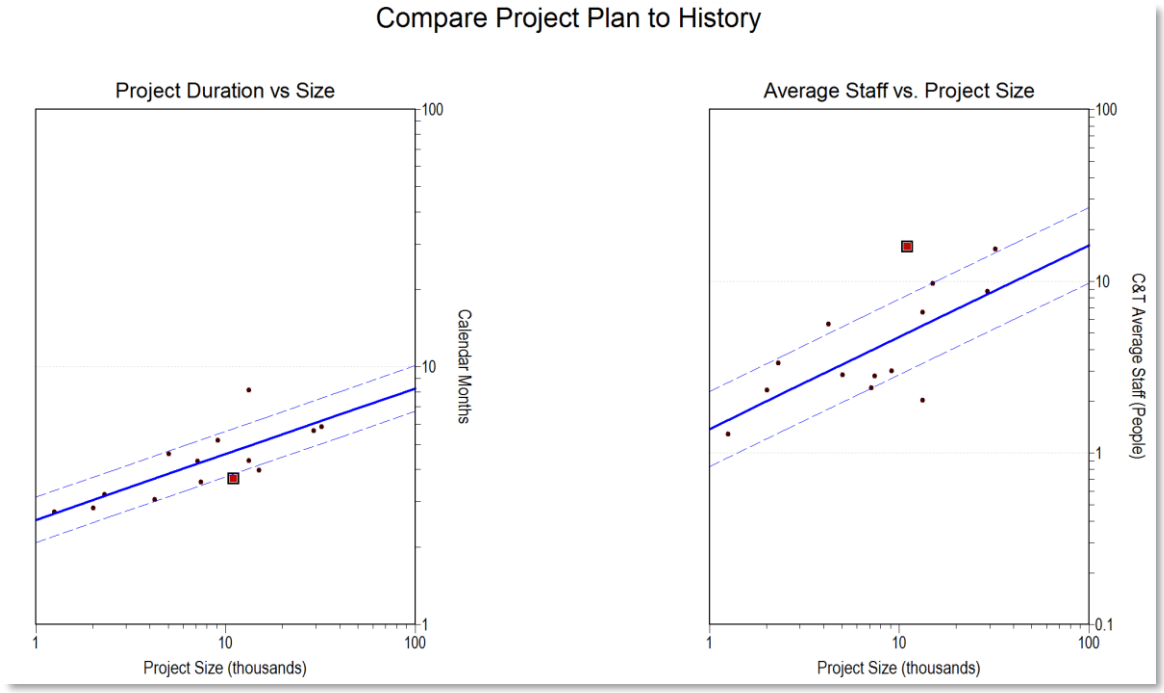


Figure 2. Five-month plan compared against historical trend lines.

In this example, average productivity (based on the organization's history) is assumed and constrained the project to complete in five months. The organization's history indicates that only once have they completed a project of this magnitude in five months and that it would require over 20 full time staff. In short, the project plan is not feasible. While this may not be welcome news, it is far better to receive it now rather than learning it later.

Consider alternatives. When confronted by a situation like the one above, there are three basic alternative courses of action. They are to extend the schedule, reduce the scope of the project, or add staff. In this case, adding staff is not a practical solution, since the staff required to complete the project in five months is already well outside of both historical performance and the budget. Reducing scope may be a viable alternative if some of the functionality can be deferred or eliminated. By far, the best solution is to relax the schedule and provide the project more time to complete, which has the additional bonus of allowing the project to be planned with a smaller staff. The impact of planning for the project to complete in 6 ½ months (Figure 3) will now be addressed.

This plan is consistent with the organization's history, with both project schedule and staff very close to average. It is clearly within the organization's capability to deliver this project in 6 ½ months with a staff of less than 10, while attempting to do it in five months with a larger staff would invite failure.

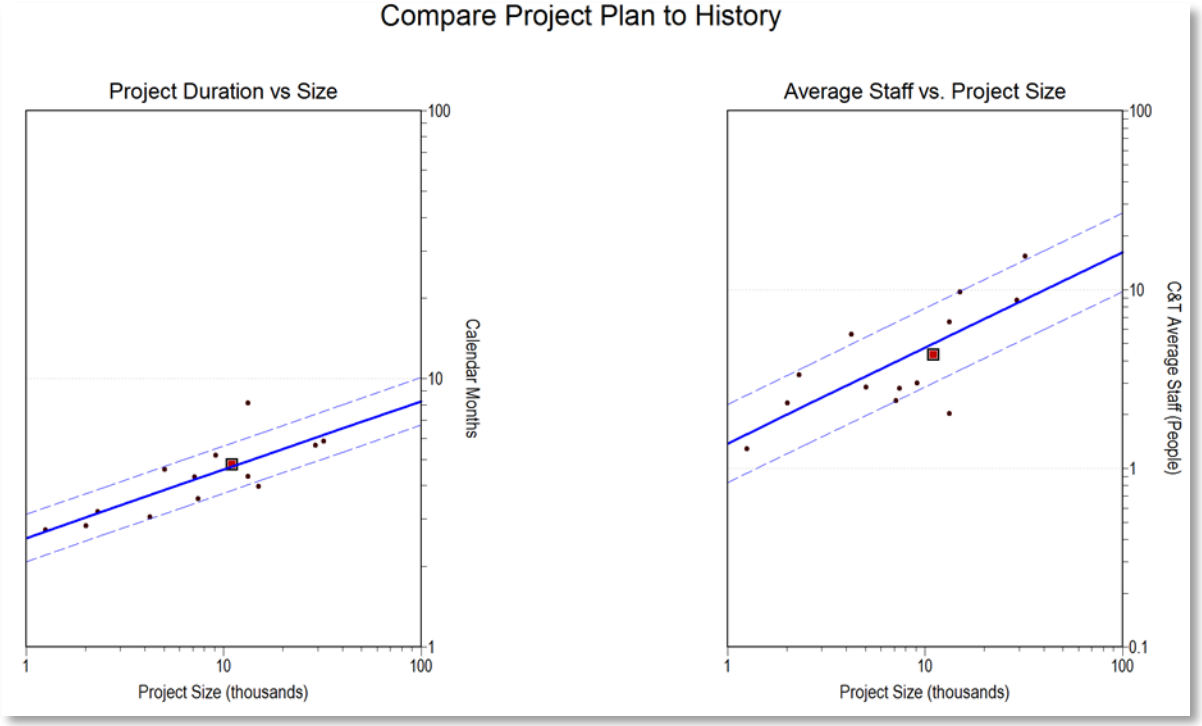


Figure 3. Impact of planning for 6.5-month schedule.

Wrapping It All Up

The examples above illustrate what can be done with software project metrics. The principles upon which measurement-based software management are founded are not complex. The real challenges lie in collecting the data and, above all, using it as the basis for decision making.

References

Beckett, D. (2007). *Using metrics to develop a software project strategy*. Proceedings from CMMI Technology Conference and User Group, November 12-15, 2007. Denver, CO.

Beckett, D. (2008). *Using Metrics to Develop a Software Project Strategy*. Proceedings from 3rd Annual International Software Measurement & Analysis Conference. September 14-19, 2008. Arlington, VA. Retrieved from http://www.ifpug.org/Conference%20Proceedings/ISMA3-2008/ISMA2008-02-Beckett-using-metrics-to-develop-a-software-project-strategy.pdf

Beckett, D. (2013a). *Function point analysis over the years*. [Web log comment and video file]. Retrieved from http://www.qsm.com/webinar/function-point-analysis-over-years?webinartitle=Function+Point+Analysis+Over+the+Years

Beckett, D. (2013b). *Staff that project*. [Web log comment]. Retrieved from http://www.qsm.com/blog/2013/staff-project

10 Steps to Better Metrics

Carol Dekkers

This article originally appeared in the July 15, 2015, edition of the Projects at Work online journal, and is reprinted here with permission.

Fred Brooks' observation that nine women can't make a baby in one month is perhaps the earliest and best known analogy between software development and parenting, and it's an apt one. An effective software measurement program — like good parenting — requires careful planning, regular monitoring, and a significant long-term investment of time and energy.

Unfortunately, many organizations collect reams of data that produce negligible return on investment (ROI). Metrics programs that can't demonstrate tangible results are usually the first casualties when the budget belt tightens. Over 20 years' experience with software measurement and process improvement indicates that management interest in software measurement follows a cyclical pattern:

1. Management gets bitten by the measurement "bug."
2. Money is spent to collect and analyze software project data, often with unclear goals (*"Just start collecting software project costs and other estimating/ benchmarking data"*). Little thought is given to how metrics will be used to support future decisions.
3. Data is collected and analyzed, resulting in changes to software cost estimation and project management processes. Software estimation gets better, reports are produced, and productivity appears to improve. The measurement program appears to be "on track" and life is good.
4. Often within six months of starting software measurement, something in the business changes (management or finances or competitiveness) and suddenly management questions the ROI for software measurement and other initiatives like process improvement or quality assurance.
5. Software measurement becomes the scapegoat for missed deadlines and overrun budgets *("If we weren't spending so much time collecting metrics, we could have finished on time!"*). Unrealistic expectations *("Measurement was supposed to help us manage things better!"*) prevail.
6. Management changes, budgets are scrutinized, and software measurement is abandoned, canceled, or postponed.
7. In hindsight, everyone saw the train wreck coming: (*"I knew software measurement was going to fail."*) Measurement is blamed, and becomes a dirty word.
8. Months or years pass before the organizations cycles back to step one.

The dangers and costs associated with poorly planned programs include wasted time and money, flawed decision making, loss of morale, frustrated staff, rework and competitiveness. There is a better way to create success with measurement. Effective and successful software measurement requires careful planning and active management that anticipates and addresses dysfunctional behaviors measurement can cause. Here are 10 steps to creating and maintaining a successful software measurement program.

1. Identify the Dangers

Simple awareness that poorly planned metrics can cause damage is enough reason to follow best practices in metrics program design. Two approaches that embrace this are the GQM (Goal Question Metric) method by Victor Basili and the PSSM (Practical Software and Systems Measurement) from the Software Engineering Institute, formerly of Carnegie Mellon University.

Limit data access to those who are skilled in data analysis. One wouldn't allow a child to play with matches — therefore, do not allow management access to raw or invalidated data. Proper data analysis and correlation is a critical success factor for any metrics program.

Be realistic with management about their expectations. A program designed to meet champagne tastes (or measurement results) on a beer budget seldom succeeds. Moreover, sometimes historical data can be collected if available, while at other times data are impossible to collect after the fact.

2. Make a Plan and Start Small

Project data can be collected quickly and easily, but it takes careful planning to make sure such data will be *useful*. In the same way that litter can pose dangers to infants, haphazard data collection can lead to measurement rework and failed programs. Identify a use for all data (that's the Goal and Questions part of the GQM approach) *before it is collected*. A few well-chosen metrics are better than hordes of unusable data.

QSM®'s Larry Putnam, Sr. recommends using five core metrics to measure the performance of software development projects: size, productivity, time (duration), effort, and reliability (2003). Simple metrics like this are immediately useful for helping to quantify risk on future estimates. In an industry study completed by QSM®'s Kate Armel, 26% of projects overran their planned budget by at least 20% — disasters that might have been avoided by collecting schedule and effort overrun data from completed projects and using it to buffer future estimates.

3. Pilot the Plan and Adjust Where Needed

Once core metrics and data to collect have been identified, test the measurement processes by doing a pilot project with a small group of participants. Run them through training, pilot the data collection, perform preliminary data analysis, and ensure that the plan and results are solid before rolling them out. In this way, flaws in the process and overall plan are identified and addressed *before* they create the need for costly damage control.

Pilot the measurement and metrics in a controlled environment before rolling the program out. Train the right people to collect and analyze the metrics to make sure the intended results are achievable. It is far easier to identify and correct undesirable behavior (often unintended consequences of measurement) in a controlled environment and minimize potential damage while the program is still small.

4. Be Honest and Open

Metrics-driven process improvement is often one of the noble goals of measurement, yet companies often hide metrics planning behind closed doors. This secrecy often gives rise to rumors. Remember: "perception becomes reality in the absence of fact." If the rationale for measurement is kept a secret (even if it is to analyze productivity levels in anticipation of outsourcing), rumors will replace facts.

Minimize rumors and their impact by making the measurement program details open and transparent.

5. Check that the Process Is Working, then Check Again

Just as it is important to check that doors are secured and escape routes have been blocked for infants, it is important to ensure that data is being collected properly. Change and measurement seldom appear comfortable (or even desirable) to those collecting data. A few will find workarounds to avoid (or even sabotage) the data collection.

When data comes in that is too good to be true or outside reasonable control bounds, spot check adherence to the collection process to determine whether the outliers reflect real variation or faulty measurement. Put checks and balances in place to ensure the data is collected and interpreted consistently and correctly.

6. Beware of Silence

Silence often warns parents that children are doing something they shouldn't (children are always loud unless they are sleeping or getting into trouble). Don't assume that no complaints mean everything's copacetic. There *will* be resistance to measurement. If grumbling isn't heard, discontent may have gone underground. It is far better to air complaints and address opposition head on than to have them undermine a program down the road.

7. Test Results Before Trusting Data to the Masses

Before success is claimed and findings are reported, reports should be tested with peers or colleagues to make sure the data supports the conclusions. The worst damage often occurs when management or ill-informed (but well intentioned) people make decisions based on what they think the data is telling them.

For example, one of the most common "fatal" errors in measurement programs is to collect data and prematurely publish metrics that result in punishing of individuals. Productivity (hours per function point, for example) depends on tools, techniques, language, and many factors other than team skills. Faulty analysis can lead to premature or wrongheaded changes in policy or processes. It's better to test out the metrics and see what kind of causality comes up (or how the data can be misinterpreted) ahead of time before sending it out in wide distribution.

Do not allow data to get into the hands of untrained users. Often sponsors want to "play with the data" once some measurement data is collected. Avoid the temptation to release the data to anyone who simply asks for access.

Perform a dry run with any metrics reports before distribution. After data analysis is done, gather a small focus group together to evaluate the results. It is far easier to identify and correct misinterpretations of a chart with a small group than it is after a large group sees it. For example, if a

productivity bar graph of projects leads viewers to the wrong conclusion, it is easy to correct the charts or add context. It only takes one wrong action based on data misinterpretation (i.e., firing a team from an unproductive project when it was instead a problem of tools) to derail a metrics program.

8. Don't Shoot the Messenger

Collected measurement data are never good or bad — they simply reflect information about a project or process as it exists today. The worse the data appear, the more opportunity exists for process improvement. But when initial productivity levels or quality data are rarely as "good" as anticipated, it can be tempting to blame the messenger or discount the message.

When professionals follow directions, collect requested data, and submit their analyses, they should be rewarded for their efforts — no matter what results the data show. Remember that measurement and metrics data report the results of a process and not of the people doing the process.

9. Be Clear and Consistent

Be honest and specific about project resources, schedules, and intended results. Condition management not to expect unreasonable results from measurement. It often takes upwards of six months to a year to collect and analyze enough data to support solid analysis that improves strategic decision making.

10. Accept that Curiosity and Wishful Thinking Are Unavoidable

Recognize that inflated expectations and wishful thinking will not disappear overnight. Management and staff may not understand that good measurement requires training (in metrics concepts), planning, design (measurement processes), implementation (initial data collection), training (for the program), communication, etc. Thus, people may give lip-service to the overall initiative without understanding that metrics projects require active management. Communication and goal-setting will be an ongoing process.

Remember, new software measurement programs are vulnerable. Protecting them requires active supervision and proactive risk management. Just as experienced parents cover electrical outlets, block stairs, and place sharp objects out of reach, experienced metrics managers must proactively identify risks, prevent misuse of metrics, and protect new measurement programs until they are strong enough to stand alone and prove their value to the organization.

References

Putnam, L. H., & W. Myers (2003). *Five core metrics: The intelligence behind successful software management*. New York, NY: Dorset House Publishing.

Increasing Project Success by Tracking the Big Picture

Keith Ciocco

Many project managers track their software and systems projects at a very detailed level, using detailed spreadsheets or other tools to track hours and tasks daily. This is fine, but it is also important to manage the big picture to avoid assigning detailed tasks to duration and budget goals that are unrealistic.

"Big picture" in this case means tracking at the project-release level and focusing on a few key actuals: size, duration, effort, reliability, and efficiency. It is important to track these actuals to a reliable plan because these are the measures that can provide the biggest and quickest insight into a project's potential success or failure. This is demonstrated in an example analysis, shown below in the SLIM-Control® graphs (Figure 1). The notional planned effort is shown in blue against the actuals, shown in red.

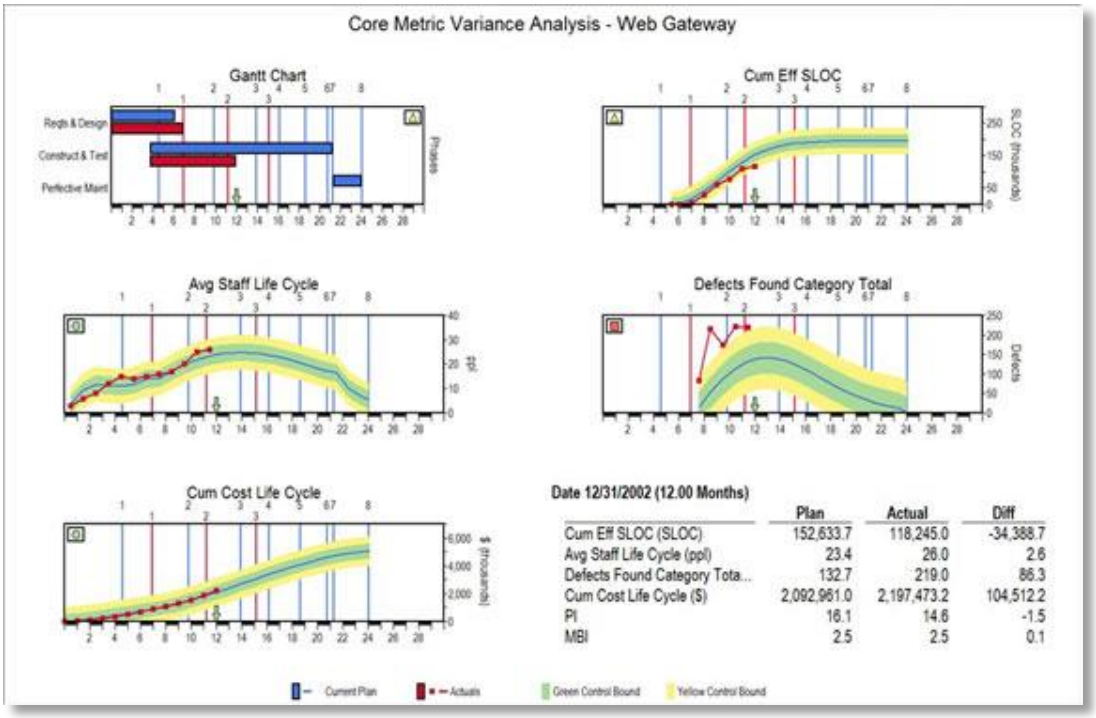


Figure 1. Tracking example of planned against actual performance using SLIM-Control®.

Once the project is underway and the actuals are being tracked, new project forecasts are generated based on the actual work delivered, time elapsed, and effort spent. These new forecasts are empirically-based and will enable managers to adapt to change requests, see when the project will finish, and how much it will cost. The graphs below (Figure 2) again show the plan (in blue) versus the actual performance (in red), but this time has the addition of the new forecasts, based on the recent project actuals (shown in white). The actual performance metrics are being "curve-fitted" against this data using empirically-based mathematical models to generate the new forecasts.

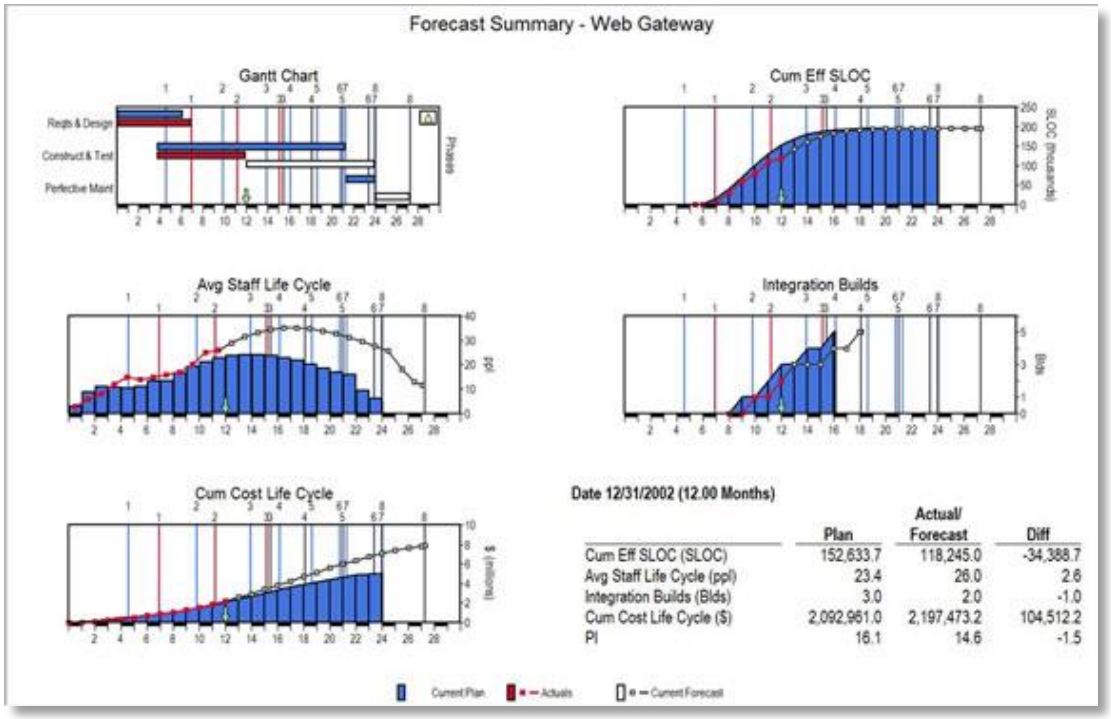


Figure 2. New forecast example, with planned and actual performance for comparison.

Also in Figure 2 above, the chart in the bottom right hand corner shows an initial plan column next to a new forecast column. This is the information (the actual effort, actual duration, and actual work delivered) that is used to forecast what is likely to happen, based on the demonstrated work performance thus far. In this case, this team is not going to achieve either its delivery or its budget goals. Also shown is the Productivity Index (PI), which is an empirically-derived measure of efficiency calculated using a project's size, duration, and effort. The control chart shows that the team is late and over its budget because it is not achieving the level of efficiency that it had planned. Seeing this analysis early in the project lifecycle will now allow the team to renegotiate its project goals.

This client can also benchmark its PI and other key measures with trends from the QSM® Industry database, one of the world's largest industry databases of completed projects. The chart below (Figure 3) show how this client's PI compares to trends in the same industry.

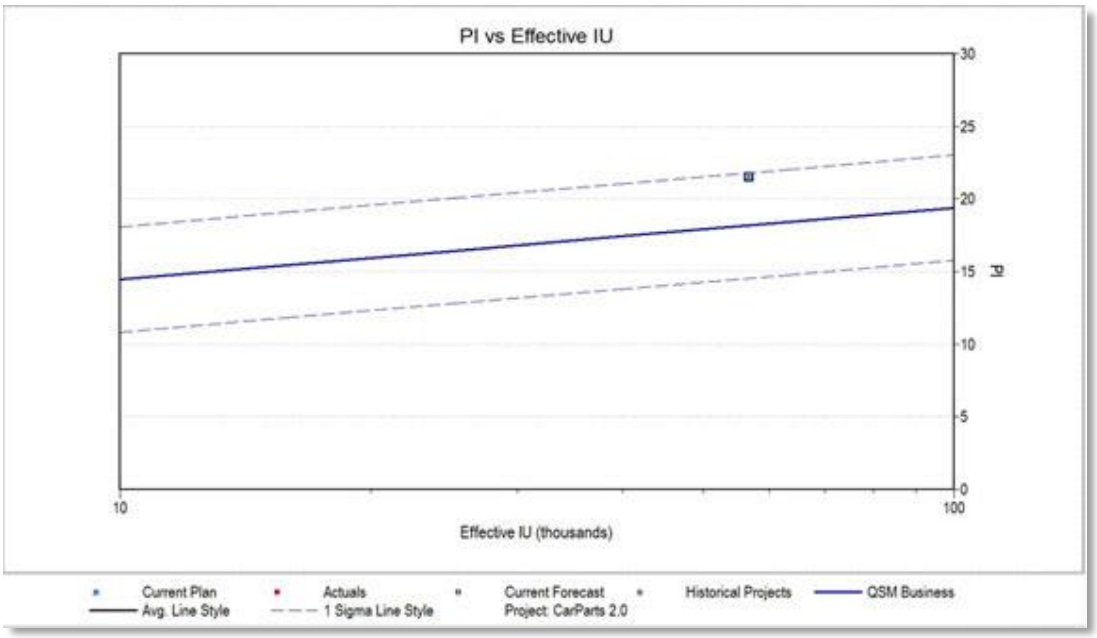


Figure 3. Sample client's productivity index (PI) compared to trends in the same industry.

It is easy to get mired in the details of a project if one does not also focus on the larger picture. Project managers need to see if their projects are in trouble early in the development process, and they need the ability to change direction and create new release-level forecasts when it is easier and less costly. Managing the big picture will help increase project success, using detailed tracking to guide their projects in the right direction.

Taking the Guesswork out of IT Project Budgeting

Douglas T. Putnam

This article originally appeared in the August 2016 online edition of ***Bright Hub Project Management,*** *and is reprinted here with permission.*

The Annual IT Budgeting challenge

The time has come to begin the annual ritual: IT budgeting. It seems like last year's budget was just finished and now it is time to start all over again. Not only is this task difficult, it is made worse by the fact that most organizations do it in an overly simplistic way. This often results in up to 40% of the projects grossly missing the mark, which wreaks havoc on the enterprise resource plans and results in disappointed business stakeholders.

Here is how the typical budgeting process exists today in most organizations. The call goes out to submit the project estimates for the portfolio of projects being considered, and the "sausage making" begins. It is often comprised of a multitude of disparate methods, which include expert opinions, detailed task-based spreadsheets, wild guesses, anticipated budget restrictions, and available resources, along with all the other methods. Often, it is simply a bucket of hours undifferentiated by roles and with no schedule attached. The individual estimates are aggregated and checked against the IT finance-directed budget limitation. If a reconciliation is required, it is usually in the form of a cut allocated to projects without regard to how it might affect functionality or schedule. Finally, there is a budget and, at the end of the year, upwards of 40% of the projects are significantly outside acceptable limits, with all the accompanying headaches. However, there is at least one way to transform this flawed method into a business process that is infinitely more efficient and will deliver better value to the organization.

The Process

This IT budgeting process consists of five steps. The first deals with the collection of project data that will enable the building of the budget. The second step is a basic feasibility assessment whose sole purpose is to identify grossly unrealistic projects that are likely to fail and the ones that are ultra conservative and wasteful. The third step is to build the budget and show the resource allocation, cash flow, and total costs, based on the "as submitted data." In the fourth step, the "at risk" or conservative projects are adjusted to more realistic scenarios and the overall budget is tuned to conform to any organizational capacity constraints. During the fifth step, all the alternatives that have potential to add value to the business or that are required to fit organizational budget constraints are considered. Ultimately, a decision is made to adopt a particular course of action. Finally, the agreed-

upon budget planning data can be fed into a Project Portfolio Management (PPM) system to facilitate specific resource allocations and portfolio management.

Step 1, collection process. In the first stage of the budget planning process, existing project estimates are collected or created from scratch. The goal is some basic information on each project. The necessary information includes:

- The estimated start and end dates of the project,
- The total effort and cost required to design, develop, and test the application, and
- A measure of the size (or functionality) of the system to be developed. The size measure can be any unit (this is entirely flexible), but usually it would be one of the following: requirements, epics, stories, use cases, function points, configurations and RICE objects, or source lines of code.

Depending on the maturity of an organization, this process can either be straightforward or may require a little more work. For example, if there is no standardization of estimation methods, then it typically requires more effort to sift through the spreadsheets and other artifacts to collect the information, make sure it is complete, and ensure there is a basic understanding of how the estimation method produced its bottom-line figures. If a high level of standardization is in place, then it might be as simple as going to a central repository and generating a report. In most cases, the former scenario is the most likely situation.

Once the information has been collected, verified, and understood, it is consolidated into a digestible format. An example of this is shown in Table 1.

Table 1. Consolidated budget data: summary table containing the budget submission data provided by each of the project managers responsible for the budget submission.

Product Area/Projects	Priority	Estimate Level	Start Date	End Date	Elapsed Months	Budgeted Amount (x 1,000)
WORK FLOW AUTOMATION		**Summary Task**	**4/1/2015**	**6/16/2017**	**26.53**	**$7,839**
1265 Field Support	High	Detailed	4/1/2015	1/19/2017	21.61	$3,860
1843 System Reliability Status	High	Detailed	1/25/2016	10/3/2016	8.32	$819
1869 Network Access Support	Low	Feasibility	9/1/2015	11/25/2015	2.83	$132
1902 BP Process Upgrade	High	Detailed	1/1/2016	6/16/2017	17.53	$1,727
1941 Dynamic Fleet Scheduling	High	Detailed	5/24/2016	6/7/2017	12.49	$1,302
PACKAGE IMPLEMENTATIONS		**Summary Task**	**12/21/2014**	**12/1/2017**	**35.39**	**$24,728**
2001 Help Desk Automation	Medium	Feasibility	6/1/2016	12/5/2016	6.16	$318
2202 SAP HR Upgrade	High	Detailed	9/25/2015	7/24/2016	9.97	$304
1993 CRM Upgrade	High	Detailed	7/1/2016	11/20/2016	4.67	$217
2052 SAP Financials	High	Detailed	12/21/2014	12/1/2017	35.39	$23,888
MISC		**Summary Task**	**12/1/2015**	**5/30/2016**	**5.97**	**$308**
2232 Dynamic Processing	Medium	Feasibility	12/1/2015	5/30/2016	5.97	$308
BACK OFFICE		Summary Task	7/17/2015	10/30/2017	27.45	$9,969
2782 Diaster Planning	Medium	Feasibility	1/1/2016	6/20/2016	5.67	$291
2945 Digital Conversion	High	Detailed	1/1/2016	9/8/2016	8.27	$856
2862 EPA compliance for 2018	High	Detailed	6/1/2016	4/29/2017	10.97	$1,579
2841 Electronic Payment Reconciliation	High	Detailed	7/17/2015	10/30/2017	27.45	$7,244
IT TRANFORMATION PROGRAM		**Summary Task**	**9/15/2014**	**1/17/2017**	**28.08**	**$11,806**
3524 IPP Replacement	High	Detailed	9/15/2014	12/30/2015	15.50	$7,407
3682 XTP Program 1st Increment	High	Detailed	5/1/2015	5/4/2016	12.13	$2,008
3103 FCC Filings	High	Detailed	10/1/2015	2/12/2016	4.41	$530
3462 Call volume reroute	Medium	Sanity Check	12/16/2014	5/7/2015	4.74	$635
2803 Unplanned expense reconciliation	Low	Sanity Check	3/24/2016	10/30/2016	7.23	$345
2945 Tranfer optimization	Low	Sanity Check	6/14/2015	11/21/2015	5.27	$239
3109 Linear Programming optimization models	Medium	Feasibility	4/1/2016	10/1/2016	6.03	$268
3321 XP power utilization	Medium	Feasibility	9/20/2016	1/17/2017	3.92	$374

Step 2, project feasibility assessment. The greatest cause of IT project failures is unrealistic schedule expectations. To improve IT project performance in the eyes of the business stakeholders, this issue will need to be addressed. The solution described here is to perform a basic feasibility assessment on each project as it enters the budgeting process. The goal is to identify wildly unrealistic or particularly conservative project estimates before expectations are set and "locked in concrete." Ultimately, these projects will be adjusted, making them more reasonable and improving the overall project performance.

So how is this done? It is initiated by creating a set of historical trend lines for schedule, effort, and staffing versus size of functionality produced. The trend lines provide a basis for the average capability that can be expected. It also provides a measure of the typical variability that can be expected. Next, the initial budget requests are positioned against the trend lines. The intent is to identify whether the projects are outside of the norm and typical variation; i.e., projects that are high risk or poor value. Figures 2 through 4 highlight some of the techniques used to identify those types of projects.

Figure 2 shows the risk level of the desired outcome of a project. When compared with historical data from the trend lines, the desired schedule of three months and budget of $250,000 would likely not be sufficient to develop 14 requirements. Moving forward with the current estimate would carry a high degree of risk.

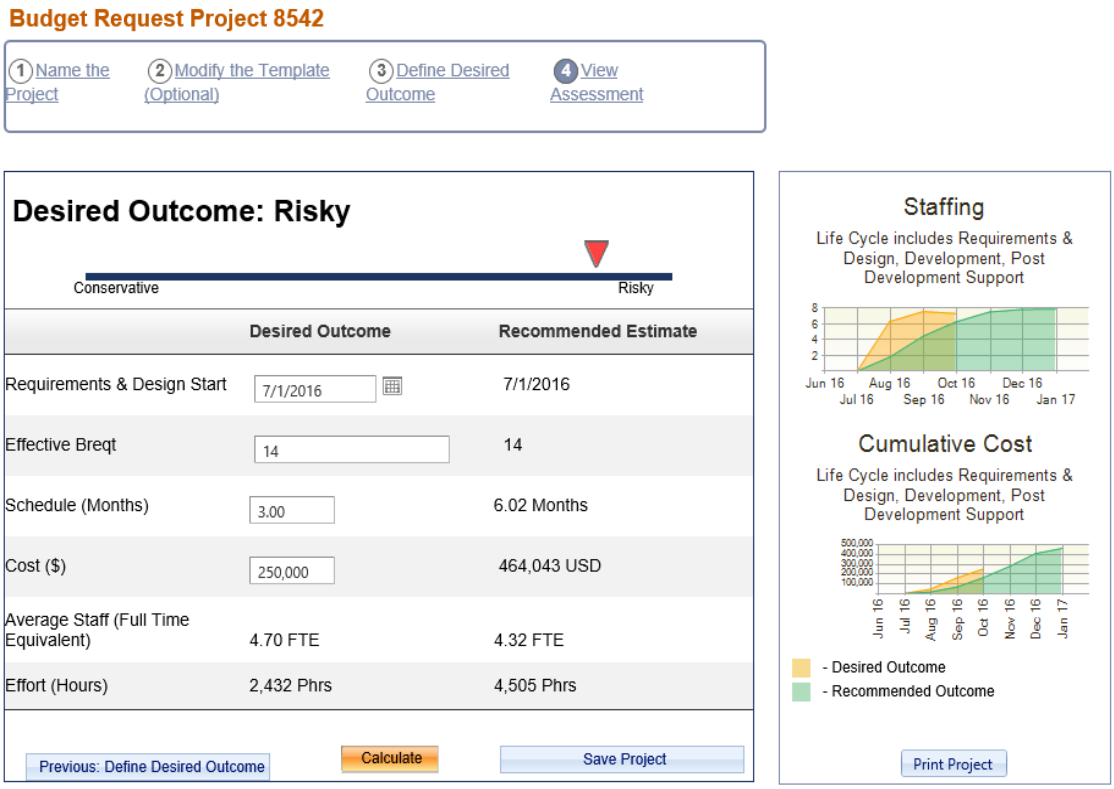


Figure 2. Feasibility assessment of original budget submission. Note that submission of desired outcome is assessed as High Risk and a recommended estimate is suggested that is more in line with historical data.

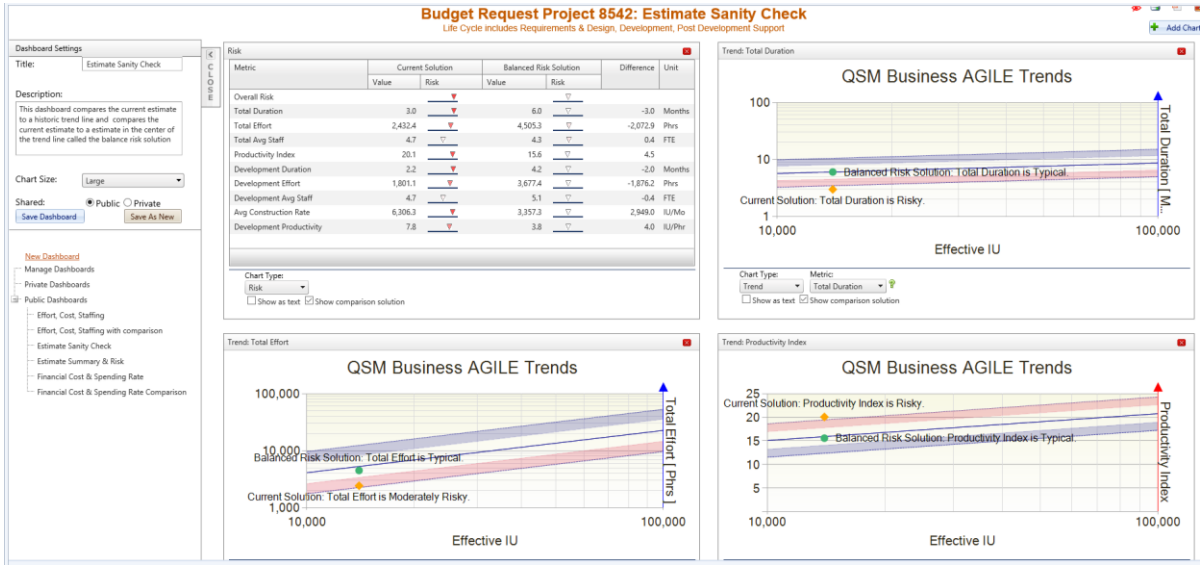


Figure 3. Budget submission (current solution shown as yellow diamond) positioned against historic trend lines.

Figure 3 shows the estimate positioned against the historical trend lines for schedule, effort, and productivity. The estimate, depicted by a yellow diamond, falls more than a standard deviation from the average trend line in each area and indicates that, historically, this has happened less than 16% of the time. In this case, the productivity, which is the driving factor, is assumed to be too high, which would result in a risky schedule and too little effort.

Figure 4 shows all the projects relative to schedule and budget (effort).

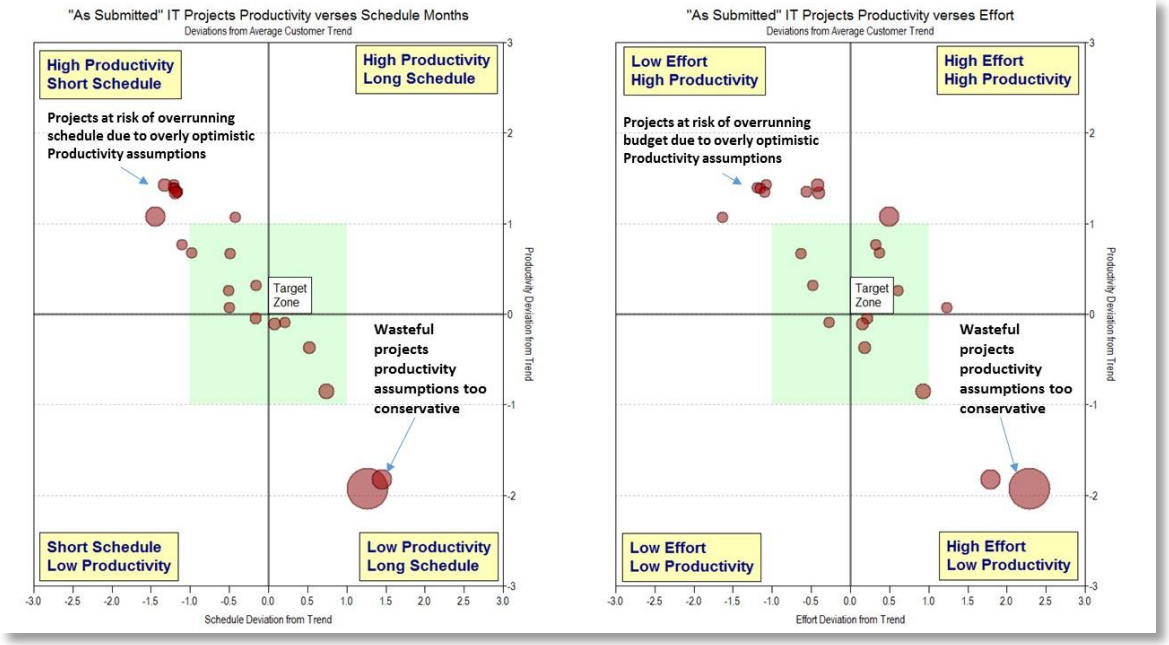


Figure 4. Quadrant chart showing every project's position relative to having enough schedule and budget.

Ideally, the perfect position would be for all projects to be within the green target zone at the center of each graph. In this case, there are several projects with high productivity assumptions and which are likely to require additional time and cost. Those projects are risky (top left quadrant). The projects that are in the bottom right quadrant are opportunities to save money. The productivity assumptions are very conservative for the projects in this quadrant.

Step 3, building the "as-planned" budget. The SLIM® suite of tools can be used to generate a staff, skill, and cash flow plan for each of the projects included in the budget submission process. It is possible to view how many people and respective effort hours would be needed to fill each role/skill category at different points in time. This information is useful as it allows the transformation of the data into peak demand points for various labor categories so that demand can be synchronized with organization capacity.

Additionally, several different templates may be needed to accommodate differences in methodology or project types. For example, there might be a group doing agile development, another that is implementing package solutions, and a third group building out infrastructure to support implementation. In this case, three templates with the appropriate skill categories and labor rates would be needed for each respective development environment.

Figure 5 shows how to identify each skill category.

Configuration Options | Skill Categories | Skill Allocations

Skill Category Name	Acronym	External ID (optional)	Labor Rate (USD per hour)
Project Management	PM		89
Architect	ARC		83
Technical Lead	TL		77
Business Analyst	BA		80
Developer	DEV		33
Application Support	APP		50
Enterprise Services	ENT		50

Figure 5. Skill category and labor rate configuration for example template in the portfolio.

Notice that Project Managers, Architects, and Business Analysts are the most expensive categories, while Developers are sourced off-shore and have the lowest rates.

Figure 6 shows how resources would ideally roll on and off one project over time.

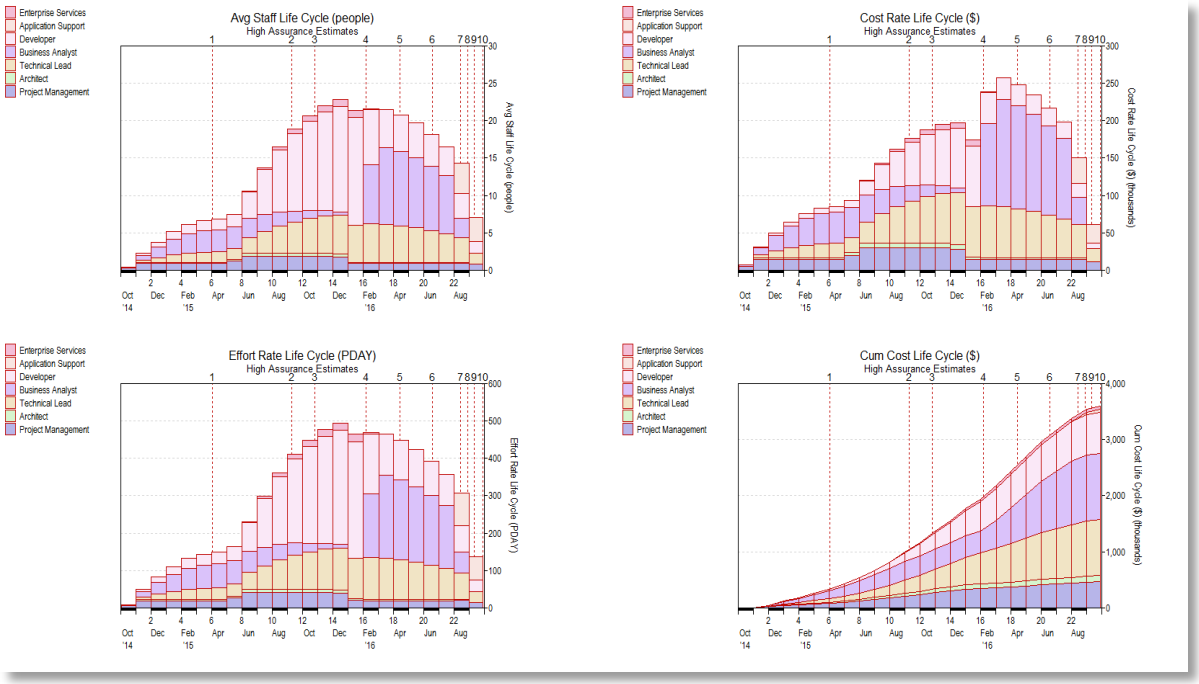


Figure 6. Resource estimate by skill type for a single project. The graphic shows staffing and effort by month, as well as skill category, and displays the monthly cash flow and cumulative cost.

When all of the projects are consolidated and rolled up across the organization, the result is an "as submitted" resource demand profile for the entire IT portfolio. The demand can be easily seen across the entire organization at once for various projects and resource types, the monthly spending profile by skill, and the cumulative cost – all useful business information.

To make this a little clearer, an example portfolio will be worked through. In this case, there is a medium-sized portfolio that consists of four main product areas. They are work flow automation, package implementations, back office capabilities, and an IT transformation program. There is also an area for miscellaneous projects. This portfolio has 22 funded software development projects. In this instance, approximately half of the project managers submitted their own estimates. The other half was estimated with SLIM® using a trend-based solution, which produces "typical" schedules and costs based on the company's historical data.

The start and end dates of each of the 22 projects are shown at Figure 7, in the Gantt chart on the left side. Notice that there are projects that started as early as mid-2014, which are well underway, while others will not even start until mid- to late-2016. In any portfolio, this would be quite normal. The graph on the right of the figure is a stacked bar chart showing the overall staffing requirements of each of the submitted projects. This proposed IT portfolio reaches its peak staffing demand in mid-2016, at approximately 225 FTE staff, and consumes approximately $55 million dollars. Notice also that a significant portion of the resources are consumed by 3-4 large projects.

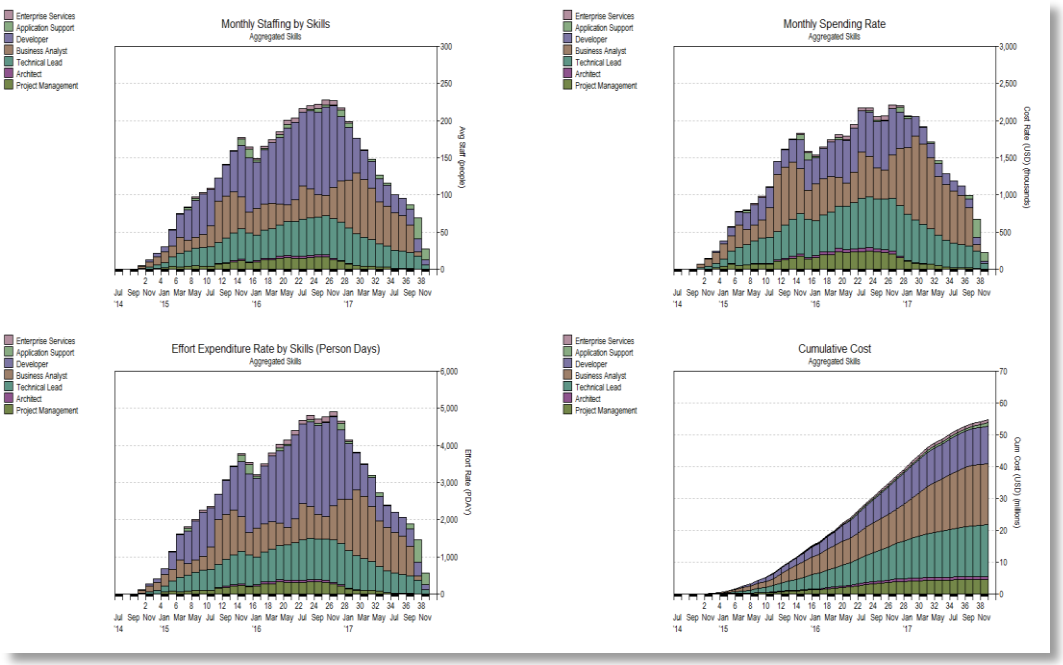


Figure 7. "As submitted" IT portfolio schedules and staffing by project – notice that the peak staffing occurrences in mid-2016 are approximately 225 FTE and that 3-4 large projects consume most of the resources.

Another view of the portfolio by skill categories and labor costs shows other important information. The peak spend rate will be $2.2 million per month and occurs in mid-2016, when the peak staffing occurs (see Figure 8, top right graph). Business Analysts are the most expensive labor category, at $19 million, followed by Technical Leads at $16 million, and Developers at $11 million.

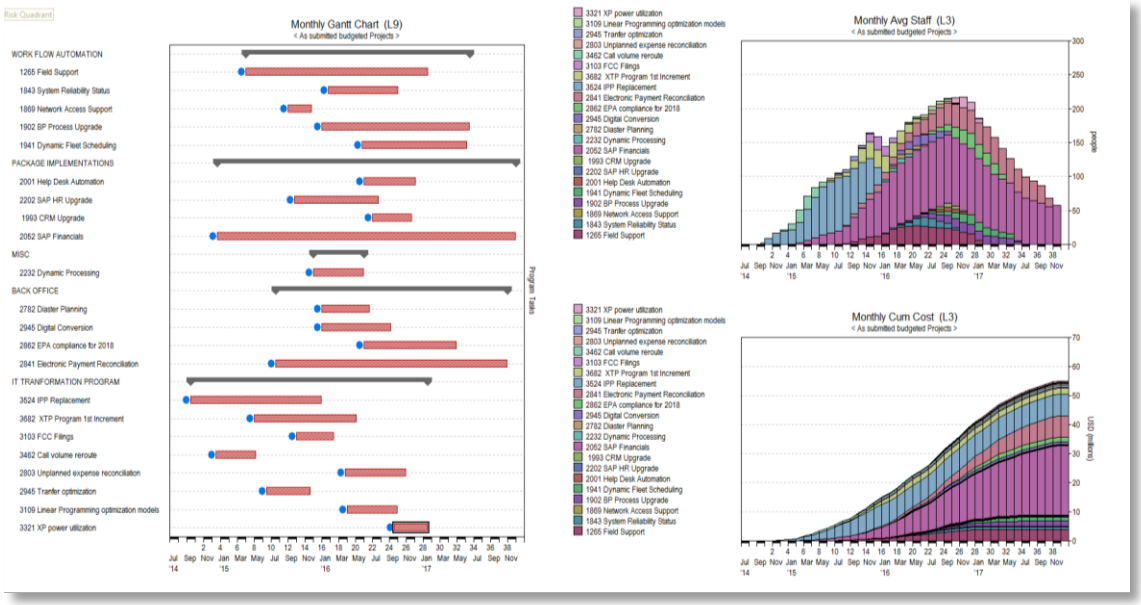


Figure 8. The example portfolio showing the IT resources and costs required by skill category. Each chart shows the aggregate view of a metric over time. The metrics displayed clockwise from the top right: Monthly Staffing, Cumulative Spending Rate, Monthly Effort, and Cumulative Cost.

Finally, notice that there is a big demand for Business Analysts for a nine-month period, between the end of 2016 and the autumn of 2017. Based on the staffing profile, no new projects could be started until December of 2016, at which point the staffing profile starts to decrease and resources would become available to begin new work.

Step 4, **building the optimized budget**. In almost all cases, there are certain constraints affecting the entire portfolio. In this case, the IT finance operation has requested that this portfolio not exceed a peak spending rate of $2.2M per month and the staffing capacity of the organization not exceed 225 people. The submitted budget meets these criteria, but it also has a large measure of risk because many of the projects have very optimistic productivity assumptions, while others are wasteful due to overly conservative estimates (i.e., low productivity and long schedules).

The first step in the optimization process is to adjust the "as submitted" portfolio to better reflect reality. Initially, the following adjustments are required:

- Projects that assumed a high productivity and resulted in a short schedule using low effort need to be adjusted so that they are more realistic. As budgeted, these programs carry a risk of significant cost and schedule overrun.
- The projects that assumed a low productivity and resulted in longer schedules and required a large amount of effort need to be adjusted because they are too conservative and, therefore, wasteful. These are opportunities to save money.

After the productivity adjustments are made, there is now a plan that is more realistic in terms of productivity, although it does not yet meet the staffing constraint or the monthly spending rate criteria. Figure 9 shows the revised portfolio.

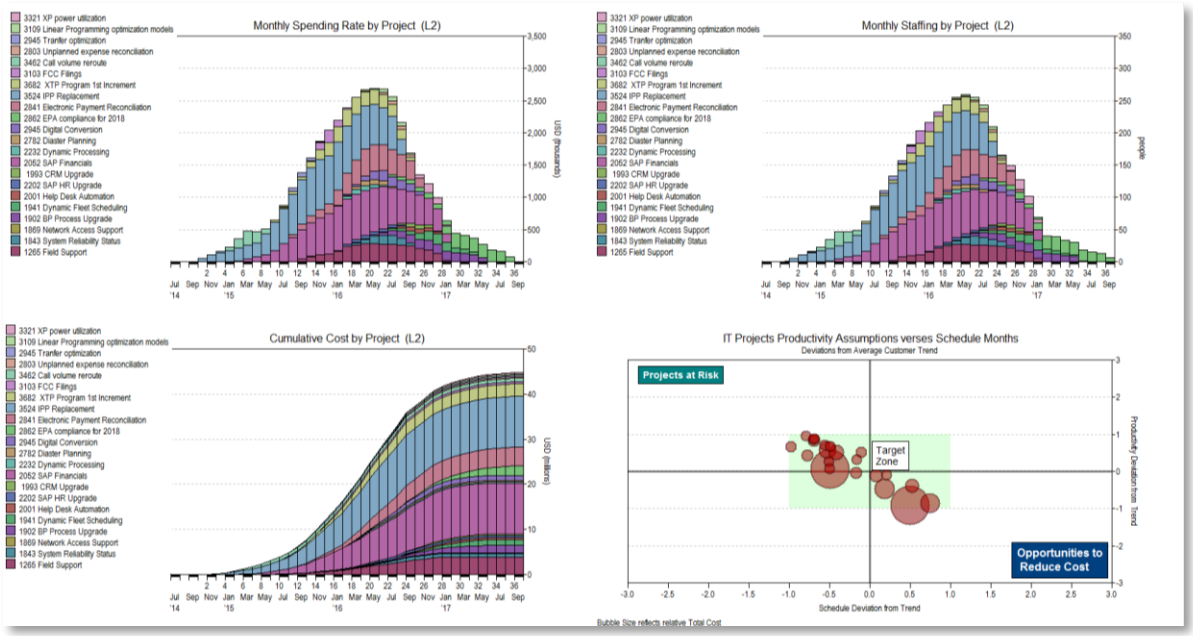


Figure 9. Productivity optimized IT portfolio showing that all the projects are now located in the target zone. This reduces the risk of cost overruns and uncovers opportunities to save money on projects that are too conservative.

The peak staffing is just over 250 people and the peak spending rate exceeds $2.5 million, so some adjustments are required to satisfy the IT finance constraints. However, through the initial stages of optimization, $10 million in savings has been identified, mostly driven by the ultra-conservative estimates of two outsourced projects.

There are two methods to achieve alignment for the staffing and cash flow constraints:

- Delay the start dates for projects that have not yet started, or
- Reduce the overall staffing on projects that have not yet reached their peak loading.

In this case, the peak staffing must be reduced by approximately 25 FTE staff, and there are seven candidate projects still to start where adjustments can be made. The start dates for each of these projects were delayed by two months. That alone was not enough to meet the constraints, however, so some modest downward staffing adjustments were made to five other projects with the largest staffs. The combination of these adjustments meets the staffing and cash flow requirements, and saved an additional $5 million by using smaller teams and modestly stretching the schedule. With the new budget in place, it can now be seen that the earliest that any new projects can be considered is May of 2016, when the staffing curve starts to "tail off," and the resource by skill level for this optimized solution are now also easily seen (see Figures 10 and 11).

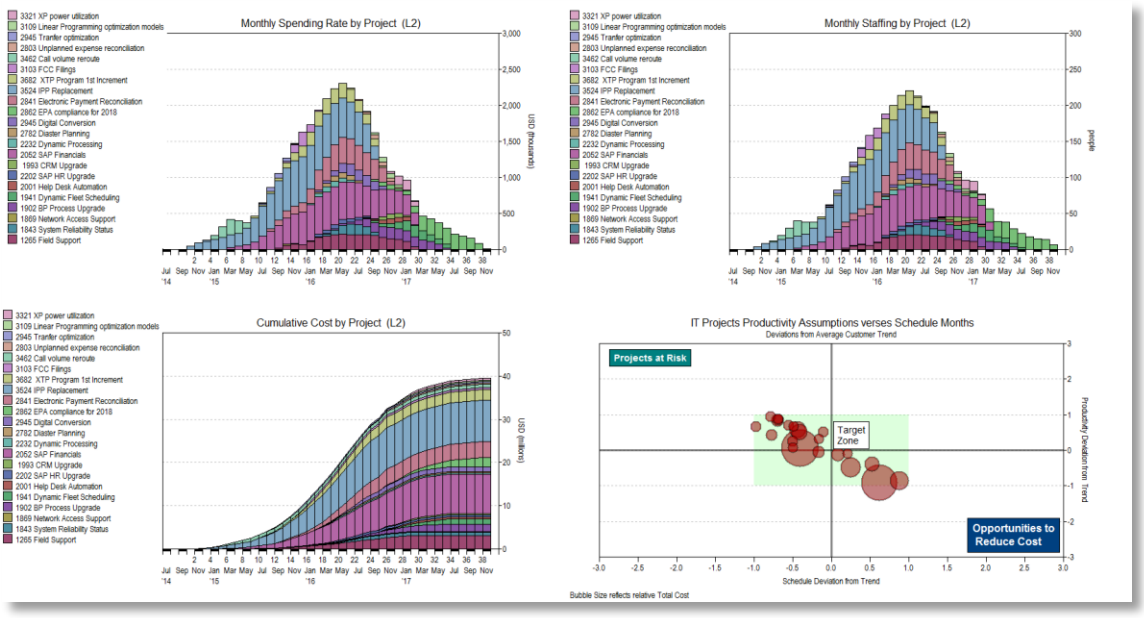


Figure 10. Optimized portfolio showing required project level staffing and optimized cash flow to meet the IT finance department budget constraints.

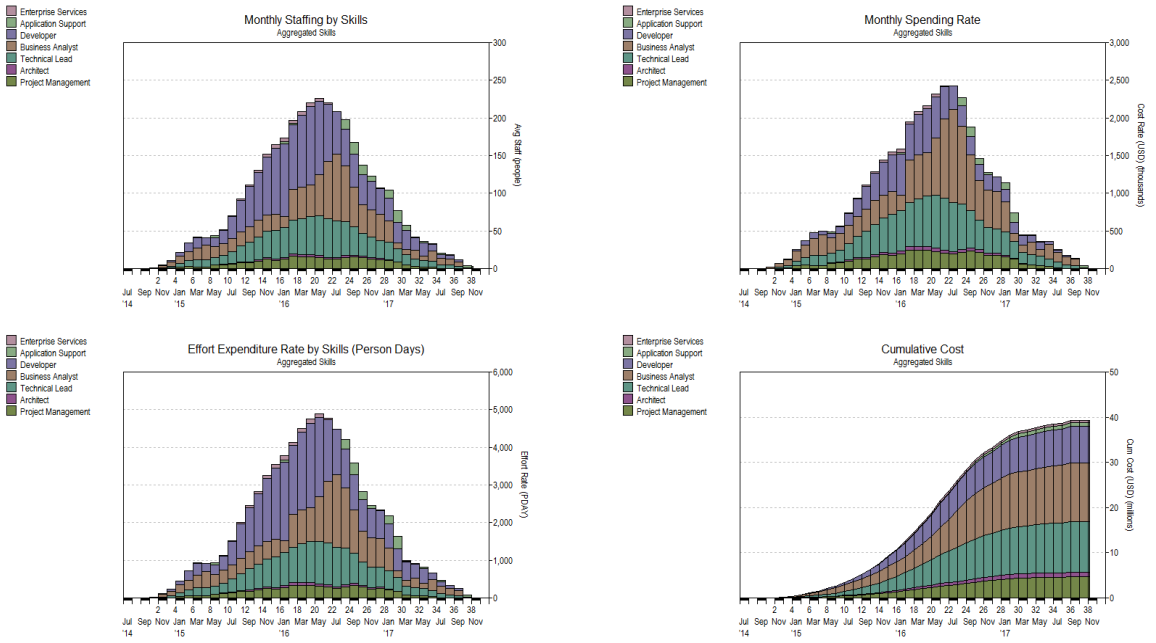
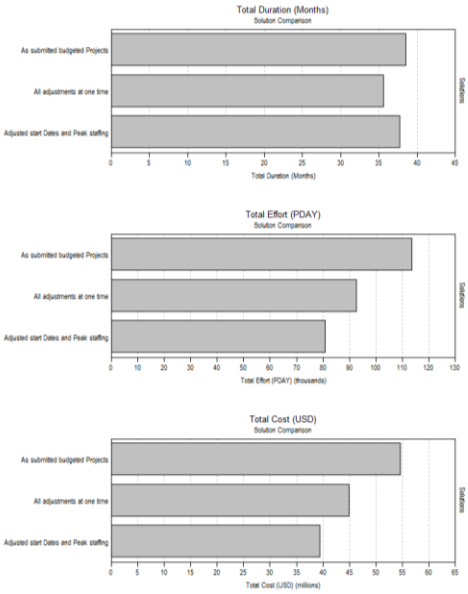


Figure 11. IT resources by skill level for the IT budget, which meets the budget constraints and has reasonable productivity assumptions.

A solution repository enables the collection and comparison of all noteworthy scenarios. It also provides configuration management for the entire process. Any potential solution can be added to the solution repository and reloaded for presentation or additional analysis. In this case, it is used to display the original budget submissions as well as the incremental steps to optimize for risk and waste and, finally, to optimize for cash flow and staffing constraints (see Figure 12).



Total Duration (Months)
Solution Comparison

Solution	Total Duration (Months)
As submitted budgeted Projects	38.57
All adjustments at one time	35.63
Adjusted start Dates and Peak staffing	37.77

Total Effort (PDAY)
Solution Comparison

Solution	Total Effort (PDAY)
As submitted budgeted Projects	113,522.92
All adjustments at one time	92,683.74
Adjusted start Dates and Peak staffing	80,809.55

Total Cost (USD)
Solution Comparison

Solution	Total Cost (USD)
As submitted budgeted Projects	54,651,480
All adjustments at one time	44,930,766
Adjusted start Dates and Peak staffing	39,421,485

Figure 12. Comparison of "as submitted" budget, risk-adjusted, and constraint-adjusted alternatives. Approximately $15 million in savings have been identified in the budget planning process.

This process identified approximately $15 million in potential savings, with little impact to the overall portfolio schedule.

Step 5, matching skills to specific corporate resources using the SLIM®-PPM integration framework. The SLIM® tooling contains a program and portfolio management (PPM) integration framework. This allows a smooth transition of skill, location, and labor rate data between SLIM® and a corporate PPM system, leveraging the advantages of each tool. For example, SLIM® is particularly good at quickly performing "what-if" analyses to support decision making. When a decision has been made, the skills and effort information can be passed to the PPM system for specific resource assignments, return on investment (ROI) business case analyses, and portfolio analyses. The following figure (Figure 13) shows the conceptual model of how the SLIM®-PPM integration works. This integration saves significant time and effort for project managers, who often must manually input and adjust this data.

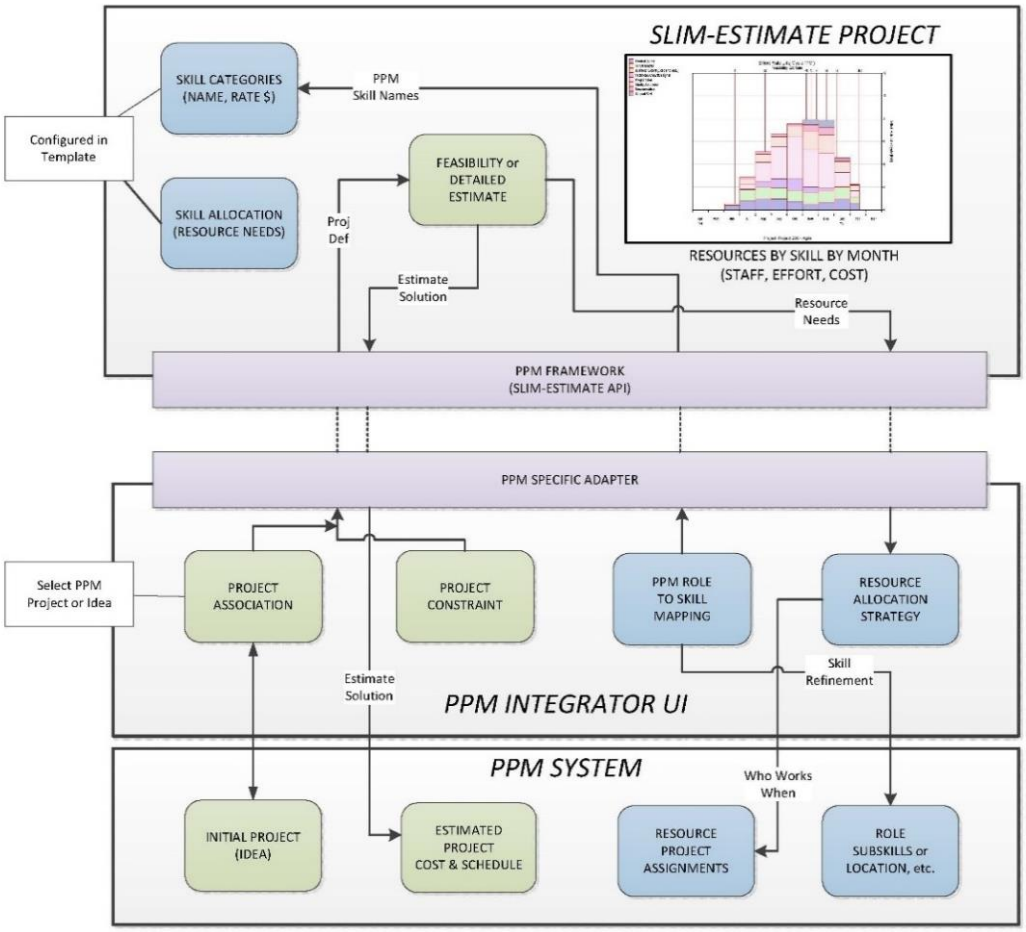


Figure 13. Conceptual diagram of the SLIM-Estimate® and PPM tool integration.

Summary of Benefits

A mature IT budgeting and demand management process can pay huge dividends and is worth considering if current methods are not fully successful, or if managers wish to maximize or improve the efficiency of their business. In short, implementing the processes and methods described in this article will help:

1. Identify and eliminate high risk projects that are likely to overrun and slip,
2. Identify ultra-conservative projects that are wasteful and opportunities to save big money,
3. Identify when to realistically take on new projects,
4. Establish realistic total budgets and cash flow projections,
5. Identify when specific skills are required to support normal development and be matched to capacity constraints,
6. Save time and money by interfacing with a corporate PPM system to streamline resource allocation and portfolio analysis, and
7. Quickly evaluate alternatives to meet specific objectives around constraints of money, staffing, or skill mix.

Better IT Budgeting with Integrated Capacity Planning and Demand Management

Keith Ciocco

A common theme among many "C-level" managers is how difficult it is and how long it takes to generate reliable resource plans at the enterprise level. Many organizations take months to generate their annual budgets, and the negotiated budgets often end up being unrealistic. One proven method is to combine good capacity planning with good demand management. There are several project portfolio management tools to help with the capacity planning. The problem, however, is that the numbers will not match if the demand management part is not correctly implemented.

There are many world-class demand management tools available that can be used by business decision makers. These tools provide empirically-based, reliable project- and enterprise-level resource plans. The SLIM-Estimate® view below (Figure 1) shows the forecasted effort of a hypothetical project, by role and by month.

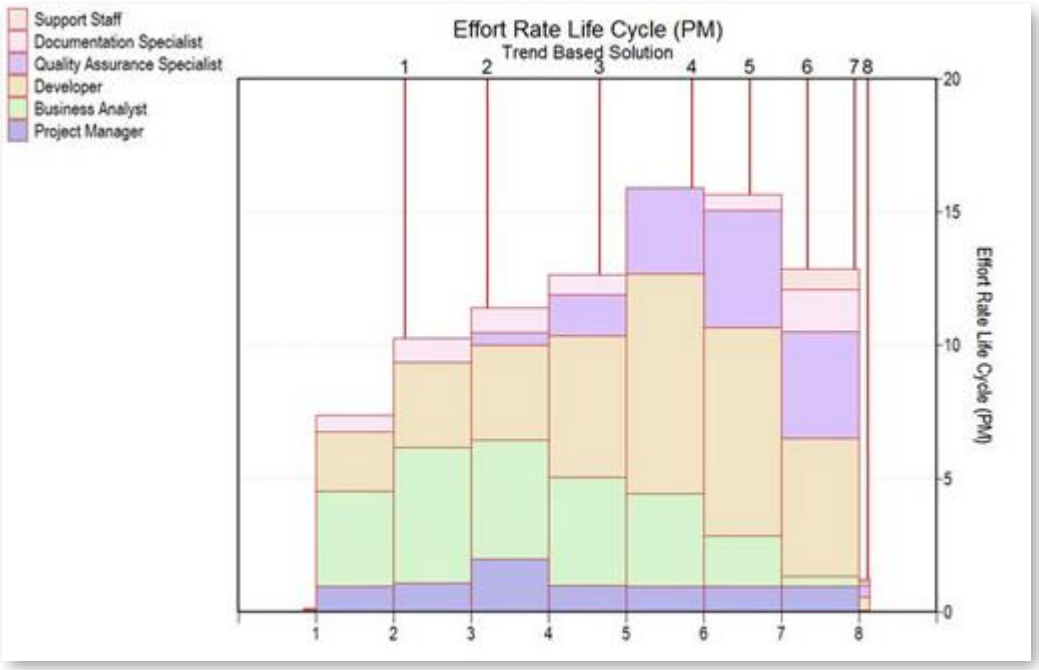


Figure 1. Forecasted effort of example project by role and by month.

The view below (Figure 2) shows the planned effort and schedule compared against historical industry data as a check to enable more confident negotiation of budgets.

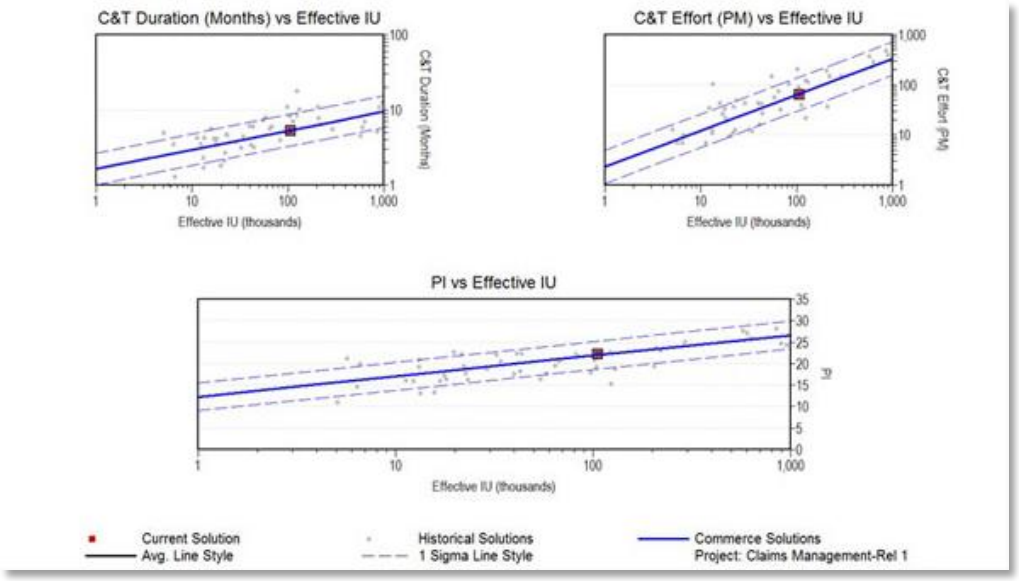


Figure 2. Planned effort and schedule of an example project compared against history.

The even greater advantage is that empirically-based demand numbers can be automatically ported into project and portfolio management (PPM) tools (e.g., Clarity, Primavera, Rational), making the job of capacity planning much easier and more reliable. Figure 3 below shows two separate plans, displayed side by side in the Project Summary box in the upper left. The original PPM plan was automatically updated using an empirically-based and sanity-checked plan exported via an application programming interface (API) from SLIM-Estimate®.

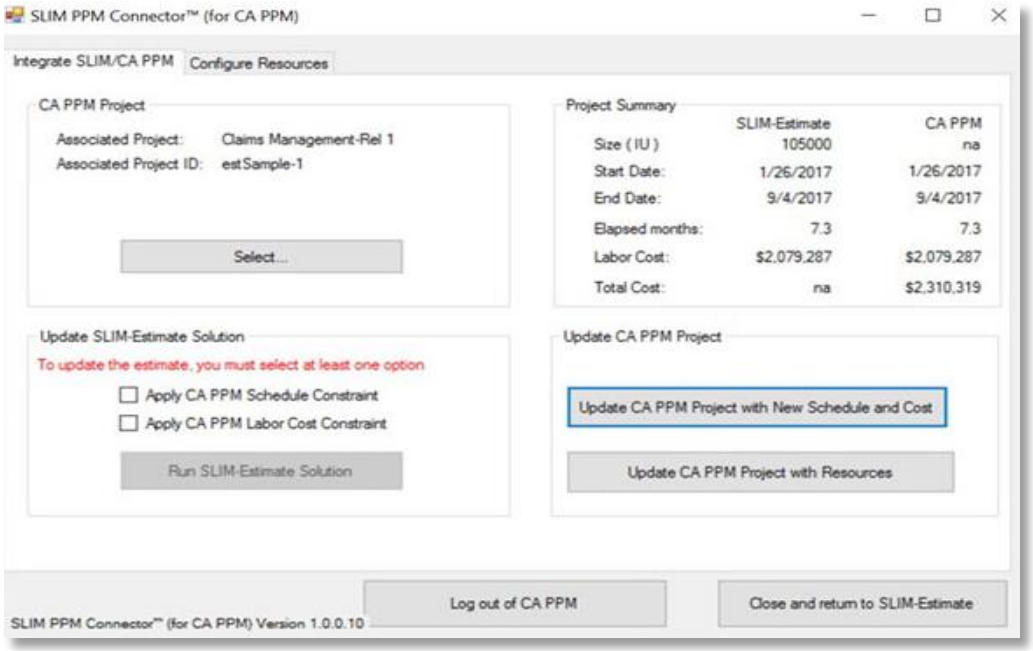


Figure 3. SLIM-Estimate® plan exported into PPM tool using SLIM PPM Connector™ (For CA PPM).

The view below shows the empty team section of the PPM tool. This is followed by a view of the same PPM tool team section now automatically populated by SLIM-Estimate®, showing the schedules of each resource by role and the number of allocated hours.

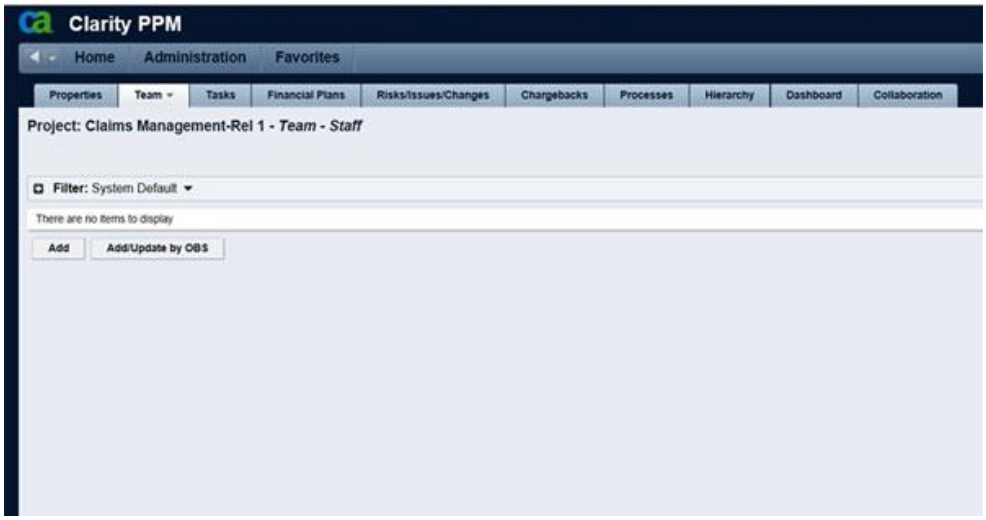


Figure 4. Empty PPM tool before the SLIM-Estimate® export.

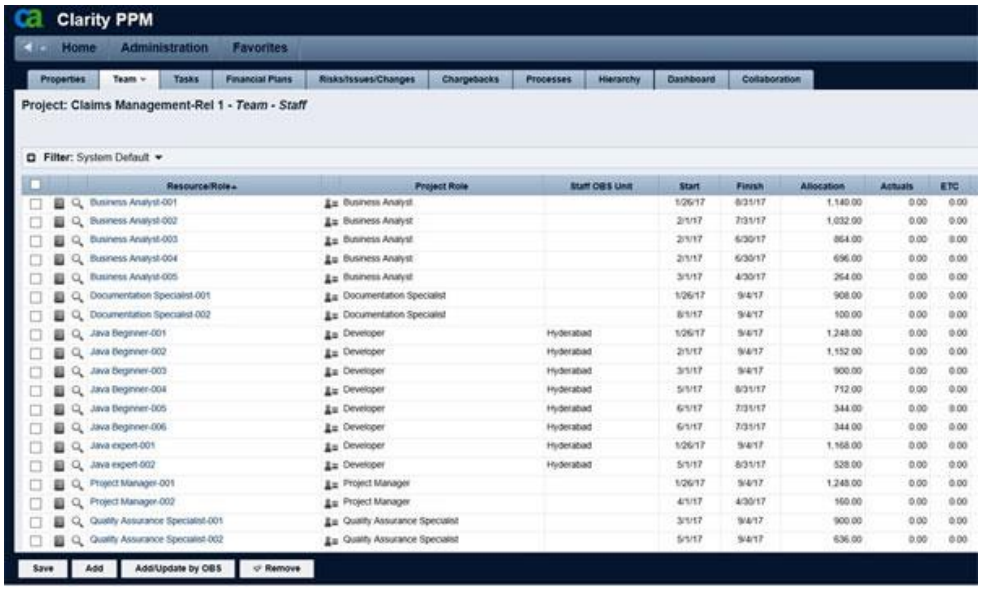


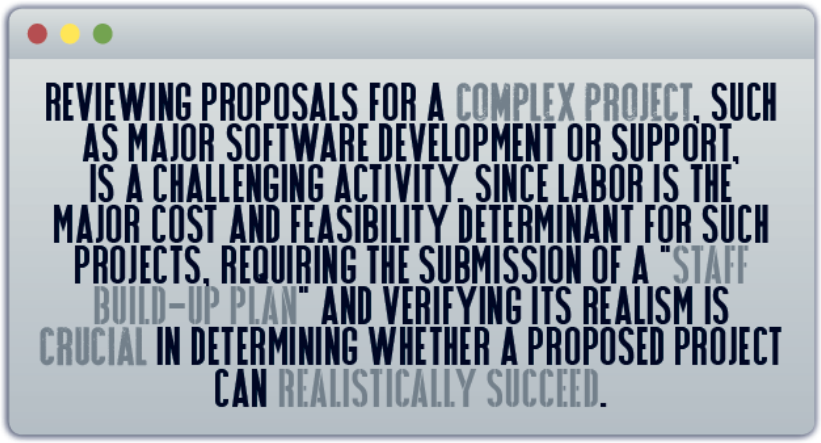
Figure 5. Populated PPM tool after the SLIM-Estimate® export.

Using these techniques, the long-bemoaned pain of IT budgeting can be alleviated, making it instead much more accurate and efficient, but only if the demand numbers are correct at the onset and if the appropriate export methods are used to port the data into the PPM tools. The more streamlined the process, the more accurate and less time consuming the planning will be, at both the project and enterprise levels.

Avoiding Doomed Software Projects by Checking the Staff Buildup Plan

Dr. Gene Shuman

This article originally appeared in the December 2015 (Vol 1/Issue 4) edition) of Contract Management and is reprinted here with permission.



The staff build-up plan defines how many, what kind, and when staff are needed for the entire project. Too many or too few, bringing them on too early or late, or employing the wrong mix of expertise or experience are red flags. Doomed projects can be avoided by rejecting proposals with unrealistic staff build-up plans.

Check the Staff Build-Up Plan

So, let's assume you've asked to review proposals responding to a big expensive contract your organization put out for bid. It's a multiyear effort involving outsourcing the development of a major new software application, and it is a major investment for your organization. You carefully read the eight proposals received in response to the solicitation. In deciding which proposal will win, several factors should be considered, including, but not limited to:

- Does the bidder understand the scope and technical complexity of the work?
- Is the proposed solution feasible?
- Is it as good as or better than the other proposals?
- Is their proposed project manager capable and experienced?
- Does the project manager—and organization—have a track record for success on projects like this?
- Are they proposing a team with appropriate skill sets?
- Will they meet your desired schedule?
- Is their price reasonable?

After reviewing the eight proposals, you manage to identify the best one that you believe answered the foregoing factors positively and its price is within your budget. On your recommendation, your organization awards the contract to that bidder. As reward for your hard and careful work, you're given the job of overseeing the outsourced contract—and the responsibility for ensuring delivery of the promised software application.

The Real Work Begins

Fast-forward three months. The winning company has begun work. You hold weekly meetings with their management team to review project status. You notice that you've received three monthly invoices covering a team that's been fully staffed since the second week. You also notice that they're still working on finalizing requirements and making early design decisions. You note that at this rate, you'll exhaust your budget before the software is completed and are on course for a cost-overrun.

Change the above scenario slightly. After three months of status reviews you notice that, while costs appear to be in line with the planned budget, product development is behind schedule. The contract project manager tells you the team is working hard and making progress, but it seems certain they won't be able to deliver the product on time. You review team staffing and discover they appear to have one-third fewer staff than needed to do the work. At this rate, the product will be late or you may be forced to increase the budget to add more contract staff. In all likelihood, both will occur and you'll have a late and over-budget project.

Change the scenario slightly again. Instead of software development, the contract calls for maintenance support of a major application. After three months, you notice a buildup of unanswered or unresolved trouble tickets resulting in a large backlog. Complaints from your user community have increased markedly. You review team staffing and it appears they have one-half the staff needed to provide the support. Your choices are either add budget so the contractor can add the appropriate number of staff or tolerate deteriorating support.

What Went Wrong?

In reviewing the proposals, you did your due diligence. The winner's proposal described the work to be done in enough detail to convince you that they understood the requirement. They have a track record in this particular area, and proposed an experienced project manager. Other proposed staff covered the needed skills. Why are you seeing problems after just a few months?

Evaluating one additional piece of information could have avoided these problems: A detailed staff build-up plan. The Project Management Institute's *Project Management Body of Knowledge* calls for a "staffing management plan" as part of the overall "project management plan," which is where this staff build-up plan would be found. Software development efforts, as well as many other types of IT projects, are service-oriented. Performing the work involves human labor, which is usually the major cost-driver of such projects. Too small a team risks late or poor delivery. Too large risks over-staffing, which can lead to cost overruns, increased defects, and unexpectedly late delivery. How would you use such a plan to avoid these problems?

Preparation is Key

Prior to awarding a contract that mostly involves services, a staff build-up plan for the proposed work needs to be reviewed. To do that, the following must be considered:

1. The organization putting out the contract solicitation must do its own internal estimate (i.e., cost, labor, and schedule) that includes a staff build-up plan for the work to be contracted. Note that bidders should not see this estimate and it should not be included in the solicitation.
2. The solicitation needs to require that proposals include a detailed staff buildup plan.
3. When reviewing proposals, the soliciting organization must compare its internal estimate with those in the proposals to ensure the proposal isn't over- or understaffing the resulting project.
4. The soliciting organization's evaluation team must have some understanding of the work to be done. The more expert they are the better. Also, they must have access to and understand the estimate and its relation to the proposed work.

What Does a Staff Buildup Plan Look Like?

Figure 1 below shows the minimum content of a staff build-up plan for a 24-month effort. It can be part of a larger staffing management plan and presented in several different ways, but this one is in the form of a resource histogram, where the resource shown is staff time. The x-axis is a period of time—days, weeks, or months (in this case, months). The y-axis is the staffing level—typically total number of staff, positions, or full-time equivalents. In Figure 1, the staffing level starts out at four in month one, then slowly climbs to its peak of 15 in month seven. The level begins to decrease in month 15, presumably because the work will begin to taper off, and ends with nine staff at the end of the period in month 24.

The time covered along the x-axis should be for the entire contract period, or a representative time period such as one year that is repeated for future year-long periods. The y-axis can be just the number of staff or full-time equivalents working in each month, as shown in Figure 1, or it can be stratified by type of labor. Figure 2 shows a more detailed chart with stratification by junior-, intermediate-, and senior-level staff. The distinction can be by skill level, cost, labor type, or other qualification that makes sense for the contract. In Figure 2, a project manager is shown for each of the 24 months in the bottom row. Two senior-level staff are shown through month 15, then only one thereafter. One intermediate staff is shown in month one, three in month two, and then varying numbers for the remainder of the schedule. Whether or not it's part of a larger staffing management plan, there should be some written description that relates the plan to the needs of the contract's work.

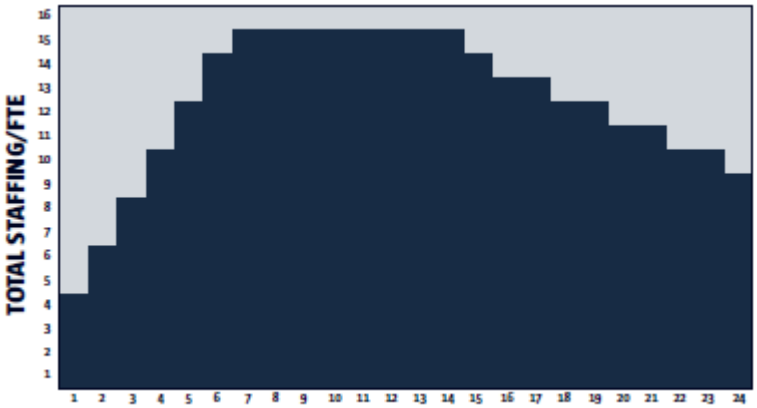


Figure 1. Basic staff build-up plan.

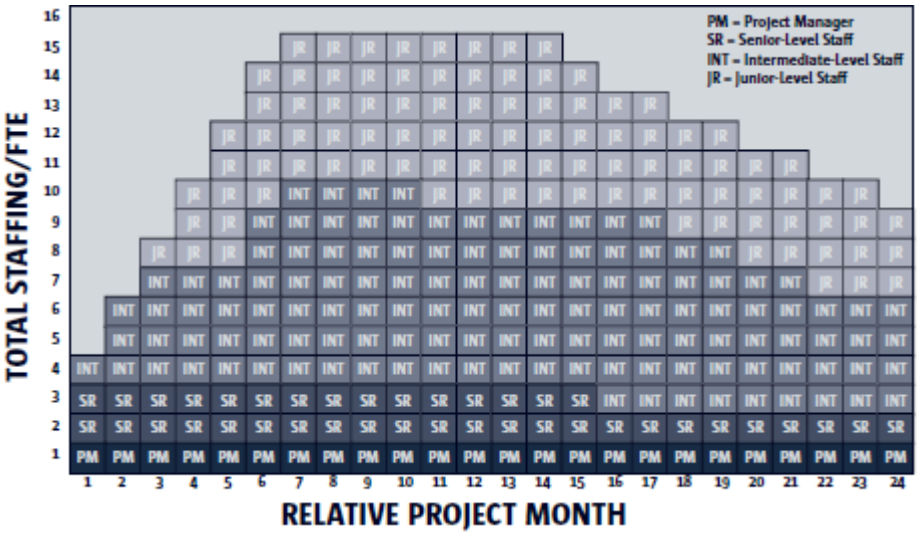


Figure 2. Staff build-up plan stratified by staff level.

What Does a Staff Build-Up Plan Tell Us?

Item three in the numbered list earlier in this article compares proposal staff build-up plans with the solicitor's internal estimate. What should proposal evaluators look for, and why? The following are several scenarios that could indicate one or more problems.

Proposal staffing level is significantly lower than the estimate. Unless the proposal makes a convincing argument for a lower-than-expected staffing level, it probably indicates one of two things:

- The bidder doesn't understand the full scope or complexity of the work, or
- They're deliberately trying to low-ball the bid.

If the first is true, the bidder shouldn't be awarded the contract. In the case of the second, the bidder may be trying to win by undercutting the competition's price with the expectation that, once they win, they can convince the soliciting organization to increase the contract budget.

Contracting processes can be long and difficult, especially in government settings. Once awarded, the solicitor may prefer to "bite the bullet" and increase the budget rather than ending the just-started contract, wasting a substantial amount of internal work, and starting over. At a minimum, this is unfair to the other bidders and, worse, the soliciting organization may end up paying more than if someone else had won. Unfortunately, this too often plays out in practice.

In highly structured evaluation situations, as is usually the case in U.S. federal government contracting, technical and cost proposal evaluations are often split between two separate teams:

- The technical team, and
- The cost team.

The technical team might not have access to costs and, if no detailed staffing plan is evaluated, may declare an under-staffed proposal to be technically acceptable. Later, that proposal may end up

as the cost winner and win the award because the price is low and the technical part was deemed acceptable. This practice—i.e., "lowest price technically acceptable" (LPTA)—is not a good model for high-complexity work like software development or support, but is often used in practice. LPTA errors can be avoided if the technical team spots the under-staffed proposal during its review and declares it "not acceptable." In that case, it won't win the award no matter how low its cost because the staffing was found to be completely inadequate for the job.

Proposal staffing level is significantly higher than the estimate. As with the low staffing scenario, unless the proposal provides a strong justification for a larger-than-expected staff, it should be rejected. As before, either the bidder doesn't understand the requirement or they're padding the proposal. In practice, this case is easy to reject since the cost will be in the higher range. Projects with high complexity may call for a "best value" type award in which a higher cost can be justified by a truly superior proposal. This must be carefully considered and the technical evaluation team thoroughly convinced of its superiority if it declares such a proposal as "acceptable." In such cases, the evaluation team must make a strong justification for award on technical merits since the proposal's cost will very likely be high.

Proposal staffing level is close to the estimate, but the skill mix is biased toward junior or senior staff. If provided in the proposal, does the skill mix match the solicitor's estimate or the technical evaluation team's assessment of the project's need? The mix could be junior/senior or a specific labor category breakdown (e.g., systems analyst, lead developer, database expert, etc.). Proposing the right skill mix is another clue as to the bidder's understanding of the required work. Also, a mix consisting of too many senior staff could indicate price padding, while too many junior staff indicates an attempt to low-ball the price.

Proposal staffing level does not match the rhythm and pace of the work. The staff build-up plan shows not just "how many," but also "when" staff is needed. Support contracts tend to be operational and call for the provision of a certain number of staff for the project's duration. In this case, rapid staff build-up would be expected almost immediately upon contract award. For software development projects, however, a lot of front-end analysis may be needed in the early stages and a more modest and gradual staff build-up pace would be expected. One potential trouble scenario is where the full staff is quickly brought onto a project in which front-end work will dominate the early going. For example, a full team of 20 staff is brought on board by the end of month one, but only five will be engaged in planning, requirements, and analysis until the end of month six. The soliciting organization would be paying the vendor for up to 75 staff months of low- or no-value effort in month two through six for the excess staff.

Proposal staffing level and skill mix is close to the estimate. This is the hoped-for scenario. Proposals that match the estimated staff build-up levels and pace with the right skill mix (when provided and evaluated) indicate an acceptable proposal, at least for this evaluation factor (other factors are also evaluated).

By matching the estimate, we mean the proposed staff build-up is not significantly above or below it. But what does "significantly above or below" mean? The answer to this question should be based on both a trend line of historical data and the evaluation team's judgment. The trend line of historical data should show size of the software functionality versus effort and/ or staffing levels. If the proposed staffing is more than a standard deviation above or below average, the evaluation team should consider rejecting the proposal. The evaluation team should also apply some expert judgment. Does anything else in the proposal clearly justify why fewer staff than expected can do the job or that

more are needed? Perhaps the bidder has some kind of process, tool, or other "secret sauce" that allows them to be more efficient than expected, leading to lean staffing. Or perhaps the complexity of the work or high quality of the proposed solution justifies more than the estimated number of staff? Having a highly expert evaluation team will help sort out these issues and lead to a good award decision.

Staffing Model for Engineering Projects

In the early 1960s, Peter Norden of IBM discovered that engineering project staffing tended to follow a statistical pattern called the "Rayleigh curve," which is part of the family of Weibull distributions. Figure 3 below shows a typical Rayleigh curve—a skewed-left version of the normal or bell curve. In the early going of a project, staff is added gradually to keep the team small since only a relatively small team is required for planning and analysis activities. As work picks up, staff is added rapidly until a peak is reached. The peak staffing continues until most of the work is accomplished, followed by a gradual decrease in staffing while the project is implemented and supported in production.

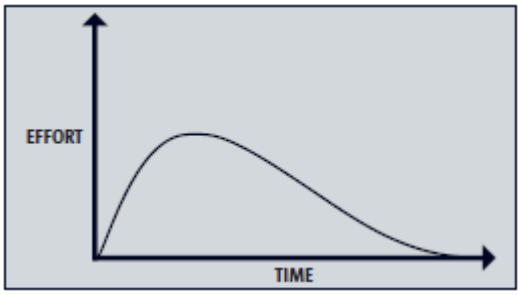


Figure 3. Typical Rayleigh curve.

In the 1970s, Lawrence Putnam, Sr., discovered that the curve also appeared in software development projects. He developed the "Software Lifecycle Management" (SLIM®) estimation tool, which uses the Rayleigh curve to model an estimated staffing curve. The SLIM® tool is calibrated with over 10,000 industry and government projects and has been refined since then to accommodate innovations in development practices, such as agile acquisition. SLIM® can be used to estimate project cost, schedule, and defects. It can also show how a proposal's estimate compares with a historical trend line and whether that estimate is more than a standard deviation above or below, as previously described. It can model various cost/schedule tradeoffs, showing the impact of staffing level on the schedule, or, conversely, of shortening or lengthening the schedule on needed staff. The relationship among those factors in engineering projects is not linear, as in the case of manual labor projects, but is more complicated. The SLIM® model captures the interplay and allows for reasonably accurate estimates of such projects.

When evaluating proposal staff build-up plans for software development projects, ask how closely the plan follows the Rayleigh curve. Is there a gradual staff build-up to a peak, sustained for a period of time, then a more gradual reduction? If yes, the plan is consistent with this well-known staffing model. If not, why not? Figure 4 shows a Rayleigh curve superimposed on the staff build-up histogram from Figure 2. Here, the histogram shows the curve's characteristic shape and is an indicator of a valid staff build-up plan.

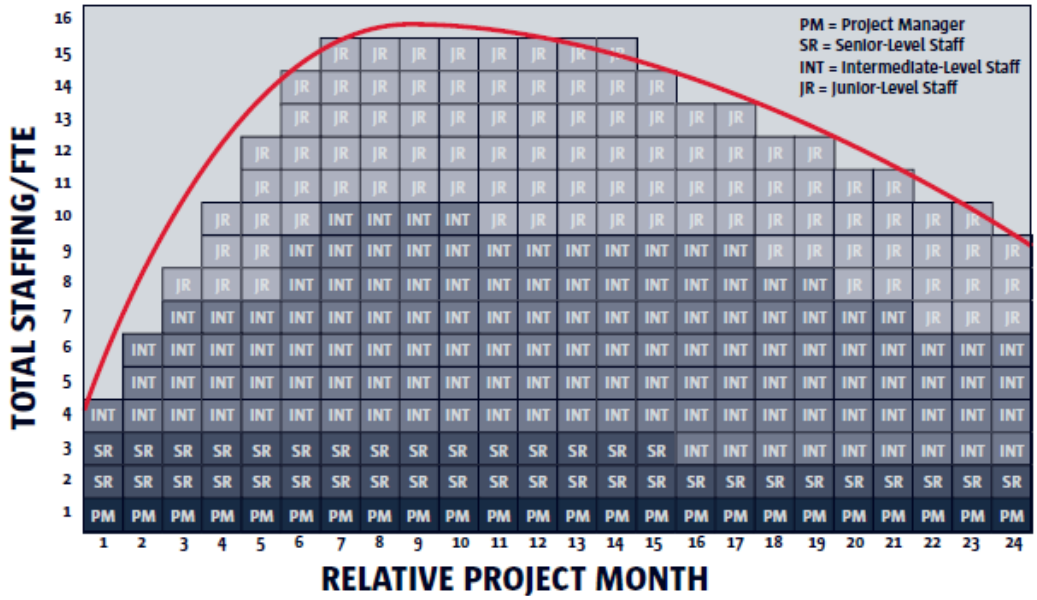


Figure 4. Rayleigh curve superimposed on staff build-up histogram.

Evaluating Proposals – Art and Science

As described in this article, reviewing the staff build-up plan gives the evaluation team an objective way of evaluating a service-oriented project proposal—whether it's part of a contract bid, a proposed internal project, or a project review. Comparing the number of staff, their mix, and when they're needed to an internal estimate helps the evaluator decide whether the proposing company understands the work and is making a good-faith bid. It can help weed out low-ball proposals and avoid cost overruns and failed or cancelled projects. Staffing that significantly differs from the estimate is a red flag and needs to be examined closely. Significant differences that aren't credibly explained in the proposal should lead to rejecting the proposal, even if its price is low.

Staff build-up, however, is only one of a number of factors to be evaluated. In software development projects, for example, other factors to evaluate include the following:

- Are the proposed solution's architecture, feature set, and nonfunctional requirements consistent with what the contract calls for?
- Does the bidding company have a history of successfully performing this type of work?
- Are the project manager and named team members experienced in this type of work, available, and capable of delivering the required product or service?

These factors are interrelated in various ways, and improving upon some can mean weakening others. Judging them requires that each not only be assessed on its own, but also their relationship to each other and what kinds of tradeoffs can and should be made.

This is a complex process and is as much art as science. While objective measures can inform these kinds of complex decisions, in the end the evaluators need to use their own expertise and judgment to get a complete picture and make a sound decision.

How a Center of Excellence Can Help Teams Develop Excellent Software

Douglas T. Putnam

This article originally appeared in the June 2016 edition (Issue 326) of Software Development Times and is reprinted here with permission.

The ways that enterprises handle software development have changed immensely over the past couple of years. Today, many organizations are upending traditional business cultures as they strive for greater collaboration through things like DevOps and other agile development methods.

But even as things change, some core principles must remain the same. Business stakeholder requirements need to be delivered within a reasonable timeframe and budget, with a good user experience and solid return on investment.

Software development estimation can check off all of these "must have" boxes. Estimation helps development teams and organizations ensure that their projects remain on track, even (or perhaps especially) in highly agile environments.

The problem is that not every organization does software estimation correctly—or at all. However, enterprises that are already undergoing transformation are primed to integrate software estimation practices that offer long-term benefits.

Most of these organizations would benefit from creating an "Estimation Center of Excellence," or ECoE. That may sound aspirational (and it is), but the concept is based on a proven methodology focused on a combination of people, process and tools.

An ECoE is a separate entity within an enterprise. It's comprised of employees whose primary functions are to ensure the development and implementation of sound software estimation processes. An ECoE can help businesses avoid potential mistakes and software failures. It can also

provide them with a way to deliver better solutions more efficiently and productively, saving time and money.

Like DevOps, establishment of an ECoE is more a shift in culture, rather than technology. Thus, it's important for managers to first figure out if such an entity will be accepted, and whom it will comprise. Some organizations may need to hire estimation experts, while others might be able to pull from existing internal resources. Others may need to overcome resistance to change, which is natural when anything new comes along that could upset the status quo. And everyone will need to determine their anticipated ROI, which will change based on each organization's unique goals.

Once those baselines have been established and it's been determined that software estimation and an ECoE is the correct way to go, enterprises can begin implementing their estimation solutions. The best way to do this is to follow a four-stage model that takes organizations from creating an initial ECoE blueprint to "flipping the lights on" and keeping things running.

Defining Requirements

The first stage involves defining the requirements for the ECoE. During this stage, managers identify their organizations' specific pain points and goals, and begin building a business case for addressing each of them. Management should assess the current state of software estimation within their organizations, and tailor the ECoE as applicable. They should also begin identifying the major stakeholders who will become part of the ECoE and advertise their role in a successful implementation.

At this point, it's also important for management to set forth ground rules for ECoE stakeholders to follow. Everyone should be on the same page in regards to the estimation process the organization has chosen to adopt. Disagreement or lack of buy-in poses the risk that projects will be derailed, thereby undermining the ECoE's purpose.

Design and Development

The second phase may be the most intense part of the process. A lot should happen here: Staff should be hired and trained, business processes defined, and service-level agreements reached.

There's a lot of additional detail that should be attended to, particularly related to establishing business ROI. First, historical data repositories should be established to support estimation and business-case ROI analyses. Then, several pilot estimates should be performed to help the staff gain experience and fine-tune the estimation methods so they may be able to deliver the expected ROI.

This is also another chance for management to gain further support behind the ECoE effort. They should be prepared to communicate with key individuals throughout the organization to gather support and build the value of the ECoE. It's an opportunity to unify the organization to support the program. That's important as, like DevOps, software estimation requires buy-in from everyone to be truly successful.

Launch and pilot operation

Finally, the moment everyone's been preparing for: the ECoE becomes operational. During this stage, staff should be mentored and trained as they gain experience. New users should be brought on board as necessary, complementing the individuals who are already in place. These teams should

develop case studies of successes achieved showing the efficacy and value of the ECoE's operations so far.

The people working in an operational ECoE have now become part of something more than just their company: They are members of a community of stakeholders practicing successful software estimation techniques. Thus, during the third stage, some organizations may choose to implement sharing circles that allow these stakeholders and their colleagues to share knowledge and practical experience with the rest of the estimation community. Ideally, this will help them and their peers enhance their software development efforts.

Full and ongoing operation

This final stage never really ends. In fact, over time, ECoE teams should strive to improve the processes they employ based on lessons learned from completed projects and their overall experiences. Hiring and training continues as needed. Skillsets will change and evolve. In other words, much like in agile development, the ECoE will adapt over time to continue to adequately address the business's needs.

It's true that developing an ECoE is not a quick fix. It can take many months for a completely operational ECoE to go online, and even then, it becomes an iterative process that managers will continue to refine over time.

But that's OK, because an ECoE offers long-term benefits and will pay for itself by reducing risk, maximizing business stakeholder demand, and optimizing existing resource capacity. Organizations will gain greater productivity and better insight into the entire software development process and ensure better outcomes.

It's well worth the time and effort.

INDEX

CONTRIBUTING AUTHORS

Ethan Avery has been involved in information technology for the last 18 years, supporting the software project measurement community in sales, customer service, education, and support. Paramount to his professional purpose, Ethan strives to assist clients in building solutions to solve their software measurement challenges. He has a bachelor's degree in communications from George Mason University, and is a regular contributor to the QSM® blog. Ethan has been with QSM® for more than 17 years.

Don Beckett has been active in software as a developer, manager, trainer, researcher, analyst, and consultant for 30 years. Since 1995, the focus of his work has been software measurement, analysis, and estimation; first with EDS (now HP) and, since 2004, with QSM®. He has worked for many years with parametric models and tools to estimate and create forecasts to completion for software projects, and has created estimates for over 2,000 projects. In recent years, Don has worked extensively for the Department of Defense to evaluate requests for proposals and monitor the progress of large ERP implementations. More recently, he has studied the productivity and quality of software projects that follow the agile methodology.

Paul Below has over 30 years of experience in technology measurement, statistical analysis, estimating, Six Sigma, and data mining. As a Principal Consultant with QSM®, he provides clients with statistical analysis of operational performance, process improvement, and predictability. He has written numerous articles for industry journals, is co-author of the *2012 IFPUG Guide to IT and Software Measurement*, and regularly presents his technical papers at industry conferences. He has developed courses and been an instructor for software estimation, Lean Six Sigma, metrics analysis, and function point analysis, and has taught metrics for two years in the Masters of Software Engineering Program at Seattle University. Paul is a Certified SLIM® Estimation Professional, and has been a Certified Software Quality Analyst and a Certified Function Point Analyst. He is a Six Sigma Black Belt and has one registered U.S. patent. He has a bachelor's degree in geological engineering from the South Dakota School of Mines and Technology and graduate studies in economics and systems engineering at Arizona University.

Dr. Andy Berner has helped organizations improve their software development processes for over 20 years. He has "hands-on" experience with almost every role in software development. He is on the QSM® software development team and is leading the work at QSM® to incorporate agile techniques into and enhance the resource demand management capabilities of the SLIM® suite. He has recently published several articles on agile methods and practices, focusing on planning projects to set realistic expectations. He has spoken at numerous conferences on software tools and methods, often with an emphasis on how to make sure that tools serve the team, rather than the other way around. He has an A.B. *cum laude* in mathematics from Harvard University and a Ph.D. in mathematics from the University of Wisconsin-Madison. He has seven registered U.S. patents.

Keith Ciocco has more than 29 years of experience working in sales and customer service, with 21 of those years spent at QSM®. As Vice President of Sales, his primary responsibilities include managing business development, existing client relations, customer retention, and response with a major focus on supporting the QSM® client base with their software estimation and measurement goals. He has developed and directed the implementation of the existing sales process within QSM® and has played a leading role in increasing the size of its customer base. Since he started with QSM®, the company has experienced a growth rate of more than 400 percent.

Carol Dekkers, PMP, CFPS, P.Eng (Canada), is a consultant, author, and speaker who has presented and taught in over 30 countries. She has been the primary U.S. expert for IFPUG and ISO software engineering standards for 20 years, and is a technical advisor to the International Software Benchmarking Standards Group (ISBSG.) She was recently re-elected to the IFPUG Board of Directors in Oct. 2015 (having served as its President in 1998-99). Carol is the co-author of two books: *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement* and *Program Management Toolkit for Software and Systems Development*, and is a contributor to a dozen more, including both IFPUG textbooks on software metrics. Carol also holds designations as an Agile Expert Certified (AEC) and Certified Scrum Master (CSM). She earned a master's degree in mechanical engineering from the University of Calgary, Canada.

Jay Daniel is an IT professional that has served in software development and consulting, ranging from providing Independent Verification & Validation (IV&V) to program and project management. For the past seven years, Jay has focused on agile methodologies in the implementation of software development efforts. He is a certified project manager (PMP), scrum master (CSM), product owner (CSPO), and SAFe Program Consultant (SPC). He has a bachelor's degree in computer science from the New Jersey Institute of Technology, and a master's degree in business administration from The George Washington University School of Business, focusing on finance and international business.

Capers Jones is currently the President and CEO of Capers Jones & Associates, LLC. He is also the founder, former chairman, and Chief Scientist Emeritus of Software Productivity Research (SPR), LLC, founded in 1984. Capers is a well-known, prolific author and international public speaker, with his works translated into six languages. Capers and his colleagues have collected historical data from more than 600 corporations and 30 government organizations. His research studies include quality estimating, quality measurement, software cost and schedule estimation, software metrics, and risk analysis. Capers has consulted at more than 150 large corporations and U.S. government organizations such as NASA, Air Force, Navy, Internal Revenue Service, judicial courts, and several state governments.

Joseph A. Madden leads the QSM® consulting division, which offers a wide range of professional services to clients in both the public and private sectors. He has more than 23 years of experience in IT management and consulting. This includes more than six years as an officer in the U.S. Air Force and 10 years with consulting firms KPMG and BearingPoint. At those firms, he led many high-visibility projects and played a key role in successful SW-CMM and CMMI process improvement efforts. Joe is a graduate of the Yale School of Management. He earned a bachelor's degree in computer science from Marquette University and a master's degree in software systems engineering from George Mason University.

Douglas T. Putnam is Co-CEO for Quantitative Software Management® (QSM®) Inc. He has 35 years of experience in the software measurement field and is considered a pioneer in the development of this industry. Mr. Putnam has been instrumental in directing the development of the industry leading SLIM® suite of software estimation and measurement tools, and is a sought-after international author, speaker, and consultant. His responsibilities include management and delivery of QSM® software measurement services, defining requirements for the SLIM® product suite of tools, and overseeing the research activities derived from the QSM® benchmark database.

Lawrence H. Putnam, Jr., has 30 years of experience using the Putnam-SLIM® methodology. He has participated in hundreds of estimation and oversight service engagements, and is responsible for product management of the SLIM® suite of measurement tools and customer care programs. Since becoming Co-CEO, Larry has built QSM®'s capabilities in sales, customer support, product requirements, and, most recently, in the creation of a world-class consulting organization. He has been instrumental in getting QSM® product integrations validated as "Ready for IBM Rational" as an IBM Business partner. Larry has delivered numerous speeches at conferences on software estimation and measurement, and has trained more than 1,000 software professionals on industry best practice measurement, estimation and control techniques, and the use of the QSM® SLIM® tools and methods.

Taylor Putnam-Majarian has over ten years of specialized data analysis, testing, and research experience. In addition to providing consulting support in software estimation and benchmarking engagements to clients from both the commercial and government sectors, Taylor has authored numerous publications on agile development, software estimation, and process improvement, and is a regular blog contributor for QSM®. Taylor also presented research titled *Does Agile Scale? A Quantitative Look at Agile Projects* at the 2014 Agile in Government conference in Washington, DC. Taylor is a Certified SLIM® Estimation Professional, and holds a bachelor's degree from Dickinson College.

Dr. Gene Shuman is a Senior Consultant at QSM® and a member of the George Mason University (GMU) Computer Science adjunct faculty. He previously worked at the U.S. Department of State, spanning the tenures of Secretaries Kissinger and Kerry. During that time, he held a variety of information technology-related positions, including Software Developer, Systems Programmer, Associate Director of the Paris Regional Financial Services Center, and Director for Consular Systems Development and Support. He holds a doctorate in computer science from GMU, a master's degree in computer science from the University of Maryland, and a dual-major bachelor's degree in mathematics and computer science from Penn State University. He is a certified Project Management Professional. His professional and research interests include predictive models, machine learning, data mining, software engineering, project management, and IT-oriented management areas such as contracting for services and customer relationship management.

John A. Staiger, Jr., has worked with SLIM® tools since 1985. He has extensive experience in data analysis and parametric methods. His responsibilities at QSM® include consulting and mentoring support to both government and commercial clients, and he is a senior QSM® classroom trainer and onsite mentor. He earned a bachelor's degree in economics from the University of Michigan, a master's degree in business administration (MBA), a post-MBA certificate in management science from Seattle University, and a master's degree in project management from The George Washington University. He is also a Distinguished Graduate of the Navy War College.