QSM® Software Almanac

Development Research Series

Spring 2019 Edition

QSM® Software Almanac

Development Research Series

Spring 2019 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

Portions of this publication were previously published in journals and forums. They are reprinted here special arrangement with the original publishers and are acknowledged in the preface of the respective articles.

Fifth Edition

TABLE OF CONTENTS

EXECUTIVE SUMMARY

"The difficulty lies not so much in developing new ideas as in escaping from old ones."

– John Maynard Keynes,
British economist whose ideas fundamentally changed the theory and practice of macroeconomics and the economic policies of governments

"There are no new ideas, there are only new ways of making them felt."

– Audre Lorde,
American writer, feminist, womanist, librarian, and civil rights activist

"We're all shocked by new ideas, and we're less shocked when we hear them again. And less shocked when we hear them a third time."

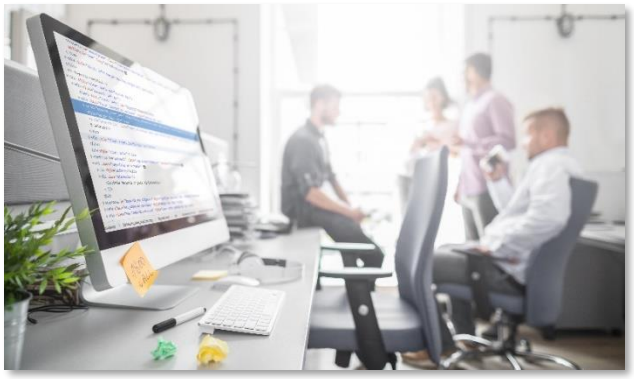
– Thomas Newman,
American composer best known for his many film scores; prolific Academy Award/Golden Globe nominee and multiple BAFTA/Grammy/Emmy award winner

"Change may not always bring growth, but there is no growth without change."

Roy T. Bennett,
inspirational author and thought leader

Executive Summary

Angela Maria Lungu, Editor

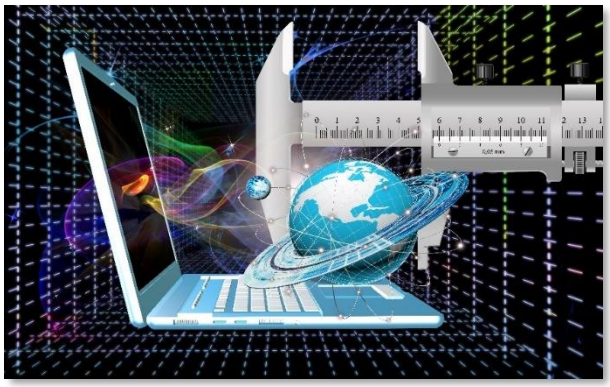


This year's almanac has a different focus from previous editions in that it focuses more on a specific development methodology: agile. We decided to spend time examining this topic as it dominates a great deal of the conversation with regard to estimation. Whether in our consulting practice or when assisting clients using SLIM® tools, we are constantly asked about the continued relevance and application of estimation and metrics with regard to agile projects. Is estimation really needed for agile projects? Do traditional metrics, sizing in particular, really apply when using agile? One of our consultants, Victor Fuster, loves telling his clients, "we were agile before agile was cool." In other words, our consulting and parametric tools have always been agile in both the ways they are used and designed – that we have always been frequently reassessing, adapting, and optimizing to meet changes in technologies, methodologies, customer cultures, or any other factor area.

The question of whether traditional software estimation maintains relevance within the agile environment is not voiced exclusively inside one industry or even one sector, for that matter. In fact, developers and program managers across the spectrum are pursuing the same lines of inquiry. This year's almanac presents a variety of articles from several perspectives, including both private and public. The intent is to show that there is indeed a compelling need to apply the basic principles of software estimation to projects, regardless of the methodology used, and that traditional metrics – yes, even sizing metrics – can and should be applied and used on agile projects.

To set the stage, Kate Armel presents a quick demographics overview of the "QSM Software Project Database." It provides a summary and offers noteworthy insights into the overall software development trends that we have been seeing over the last several years. This is a great lead-in to the first section on what QSM terms the five basic software performance metrics: size, staffing, effort, schedule, and quality. Sizing with regard to agile, in particular, is explored in Dr. Andy Berner's articles ("Sizing Agile Projects Consistently" and "Alternative Sizing Units for Agile Estimation"). He and Keith Ciocco also look at agile effort and productivity ("Measuring Effort and Productivity of Agile Products" and "Agile On-Time, But Is It Reliable?"). Andy's articles were the result of an interesting community forum hosted by QSM, which was established for the purpose of providing a platform to brainstorm

the role of estimation in agile environments and to chart a path toward better understanding for all stakeholders. In a broader application, pertaining to all development methodologies, Paul Below provides an elegant statistical validation of using peak staff rather than effort to derive a project's productivity ("Derived PI: Is PI from Peak Staff 'Good Enough'?"). Your own particular high school mathematics experience will determine how much enjoyment you derive from the explanation, but it offers a nice alternative to situations where project effort data may not be available.



The second section focuses on management and includes several articles that touch on all projects, regardless of their development methodologies. Previous editions of the almanac emphasized the importance of organizations leveraging their own historical data. Taylor Putnam-Majarian and John Staiger, Jr., take that a step further in this edition and talk through in more detail the process of collecting Department of Defense (DoD) data to establish a historical database ("Defense Trend Lines: Leveraging the Power of History"). The article is illuminating in terms of highlighting the differences between (or some might say the similarities of) the private and public sector processes. The DoD process for estimating cyber protection costs, or lack thereof, is the topic of the next article by Victor Fuster ("Estimating Cyber Protection"), who proposes a framework for estimating cyber protection costs using National Institute of Standards and Technology (NIST) Special Publication 800-53. Basic principles of sound management are eloquently reaffirmed as foundational skills in both Douglas Putnam's "Good Planning – Not Development Methodology – Is the Key to Successful Project Delivery" and Lawrence Putnam, Jr.'s "Common Ground Through PPM." Keith Ciocco adds the importance of using time boxing as a technique to estimate agile projects, noting that using this method can help attain promised functionality within time and money constraints, avoiding project failure ("Is Software Estimation Needed When the Cost and Schedule Are Fixed?"). Finally, this section finishes with Laura Zuber's very nice overview of the use of top-down estimating, also used by SLIM, and SAFe ("Role of Top-Down Estimating in SAFe").

The final section is a collection of estimation best practices. Taylor Putnam-Majarian and Victor Fuster ("Re-Discovering the Rosetta Stone: A Strategic Method to Software Sizing") put forth a great presentation on the "Rosetta Stone" gearing factor technique used to "translate" various software sizes, thereby allowing comparisons among different size perspectives. Since size is the most important factor in top-down parametric estimation, this is an invaluable technique and quite useful, particularly within the DoD. Victor Fuster adds a short piece calling for more data-driven consistency within DoD acquisition ("Data-Driven Approach for Defense Acquisition Accountability") and lays out a recommended approach for a horizontal data benchmarking capability. Lawrence Putnam, Jr., adds another compelling argument for using top-down estimation ("Top-Down Estimation Can Drive Efficient and Boundaryless Software Development"), and also give us pause when addressing the impact and importance of software quality on the areas of AI and automation ("AI and Automation Make Software Reliability More Important than Ever"). Doug Putnam ("How to Avoid the Three Top IT Project Risks") reaffirms the top three IT risks: inaccurate estimates, overstaffing projects, and denial of a project's true status, outlining recommendations for avoiding these common project failures. With that list in your pocket, Lawrence Putnam, Jr., and Donald Beckett ("Five Software Laws for

Smooth Product Development") add a complementary set of five laws that will help ensure smooth and successful projects:

- Law 1: Every software project has a minimum development time.
- Law 2: Schedule and cost do not trade off evenly.
- Law 3: Projects grow.
- Law 4: Small teams are better.
- Law 5: Allow sufficient time and effort for analysis and design.

All the articles offer research and insights into the foundational skills associated with parametric estimation. The focus this year has been, as stated initially, adapting those existing skills to account for changing conditions, specifically oriented toward agile development methodologies and across both the public and private sectors, to show where adaptation was made and how. In several cases, the importance of core metrics and best practices (or laws) was revalidated. Our hope is that these pieces provoke thought and discourse.

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

---

1. DEMOGRAPHICS

QSM Software Project Database

Kate Armel

Introduction

The QSM® software project database is the cornerstone of our business. We use validated metrics collected from over 13,000 completed software projects to keep our products current with the latest tools and methods, to support our benchmarking business, to inform our customers as they move into new areas, and to develop better predictive algorithms.

Data Sources

Since 1978, QSM has collected completed project data from licensed SLIM® suite users and trained QSM consulting staff. Consulting data is also collected by permission during productivity assessment, benchmark, software estimation, project audit, and cost-to-complete engagements. Many projects in our database are subject to non-disclosure agreements; but regardless of whether formal agreements are in place, it is our policy to guard the confidentiality and identity of all data contributors. To preclude identification of individual projects/companies or disclosure of sensitive business information, we release industry data in summary form only.

In 1994, QSM began collecting project data continuously, updating the database every two to three years. Over the last five years, we have added an average of 200 validated projects each year.

Data Quality

Only projects rated Medium or High confidence are used in QSM's industry trend lines and research. Before being added to the database, incoming projects are carefully screened. On average, we reject about one-third of the projects screened per update.

Data Metrics

Our basic metric set focuses on size, time, effort, and defects for the Feasibility, Requirements & Design, Construction & Test, and Maintenance phases. These core measurements are supplemented by nearly 300 other quantitative and qualitative metrics. Approximately 98% of our projects have time and effort data for the Construction & Test phase and 70% have time and effort data for both the Requirements & Design and Construction & Test phases.

Industry Data

QSM data is stratified into nine major application domains (Avionics, IT, Command & Control, Microcode, Process Control, Real Time, Scientific, System Software, and Telecom) and 45 sub-domains. Software projects predominate, but we have a growing number of hardware and infrastructure (non-software call center) projects, as well.

Data contributors include DoD; civilian commercial firms; and national, state, and local government entities. In addition to domain complexity bins, our data is also broken out by major industries and industry sectors. Major industries include the financial sector, banking, insurance, manufacturing, telecommunications, systems integration, medical, aerospace, utilities, defense, and government.

Methodology Data

The QSM database includes a variety of lifecycle and development methodologies (incremental, agile, RUP, spiral, waterfall, object-oriented) and standards (CMM/CMMI, DoD, ISO).

Language Data

Over 700 languages are represented with most projects recording multiple languages. Common primary languages are JAVA, COBOL, C, C++, C#, VISUAL BASIC, .NET, IEF / COOLGEN, PL/1, ABAP, SQL, ORACLE, POWERBUILDER, SABRETALK, JAVA SCRIPT, DATASTAGE, and HTML. Frequently used secondary languages include JCL, SQL, JAVA, COBOL, ASSEMBLER, C++, HTML, VISUAL BASIC, XML, ASP.NET, and JSP.

Country Data

QSM has collected and analyzed software projects from North America, Europe, Asia, Australia, and Africa. About 50% of our data is from the U.S., and another 35-40% is from India, Japan, the Netherlands, the United Kingdom, Germany, France, and other major European countries.

QSM Industry Trend Lines

The SLIM-Metrics® chart below shows Construction & Test effort for completed IT, Engineering, and Real-Time systems as system size increases. QSM stratifies project data into homogenous subsets to reduce variation and study the behavioral characteristics of different software application domains.

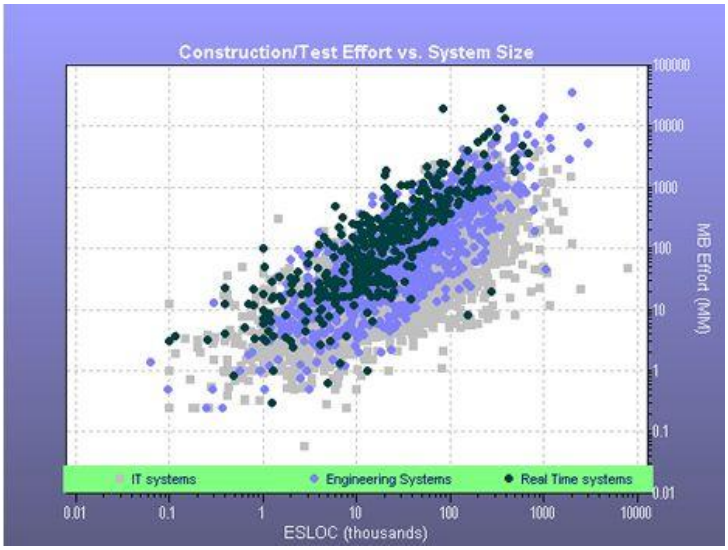


Figure 1. SLIM-Metrics chart showing Construction & Test effort for completed IT, Engineering, and Real-Time systems as system size increases.

QSM industry trend lines are available for nine high-level application domains, five application subgroups, and three application super groups. The nine application domains are:

Business	**Command & Control**	**Scientific**
System Software	**Telecommunications**	**Process Control**
Avionics	**Microcode/Firmware**	**Real-time Embedded**

Stratifying the data by application type reduces the variability at each size range and allows for more accurate curve fitting. One application domain, Business IT projects, has been further stratified into several sub-groupings:

Business Agile	**Package Implementation**
Government	**Web Systems**
Business Financial	

Three application super groups are also available to benchmark projects of mixed or unknown application domains:

Real Time Group	**Engineering Group**
All Systems	

2. FIVE CORE METRICS

"If the statistics are boring, you've got the wrong numbers."

– Edward Tufte,
American statistician and professor emeritus of political science, statistics, and computer science at Yale University; a pioneer in the field of data visualization

"That which cannot be measured cannot be proven."

– Anthony W. Richardson,
American author, speaker, and startup entrepreneur

"If you don't collect any metrics, you're flying blind. If you collect and focus on too many, they may be obstructing your field of view."

– Scott M. Graffius,
American consultant and author of Agile Scrum: Your Quick Start Guide with Step-by-Step Instructions

"Not everything that can be counted counts, and not everything that counts can be counted."

– Albert Einstein,
German-born theoretical physicist who developed the theory of relativity, one of the two pillars of modern physics

"Data! Data! Data! I can't make bricks without clay!"

– Sir Arthur Conan Doyle,
British writer best known for his detective fiction featuring the character Sherlock Holmes

Sizing Agile Projects Consistently

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 one in a series of articles resulting from this forum.

What Is a Story Point?

This often-asked question is really a trick—it doesn't have an answer, and more importantly, that doesn't matter! To see why, let's look at a more common unit of measure: the inch. While you probably don't know what it is, it would not surprise you that "inch" has a formal definition, an international standard. The international standard for an inch is exactly 0.0254 meters, which of course raises the question, what is the international standard definition of a meter? A meter is the length of the path travelled by light in a vacuum in1/299792458 seconds. Now you know!

But when you use inches for practical purposes, the definition of an inch does not usually matter. If I need to cut 10 pieces of wood that are each five inches long together with four pieces of wood that are each 11 inches long, an eight-foot-long board (96 total inches) is just about right, especially if my saw is sharp enough to make clean cuts.

Likewise, story points can be used to measure the total size of items on an agile backlog. Like the example of cutting wood, we do not need to answer the question, "what is a story point" to do that. A story assigned five story points is a smaller than one assigned 11 story points. If I have a 96-story-point epic, when I break it down to individual stories, I may end up with 10 small stories that are five story points each and four larger stories that are 11 story points each.

However, unlike measuring length in inches, there is no international standard to tell us how many story points to assign to any particular item. One team may assign a particular story five story points and, thusly, to another comparable story also five story points, while assigning to a third, larger story 11 story points. Another team, though, may assign to exactly the same stories three, three, and seven story points. Both teams rate the first two stories the same as each other, while the third story is larger. The two teams agree on the relative size, but not how it is quantitatively measured. Many

authors have explained why, when using story points for iteration planning on a particular project, this idea of relative measurement is enough. The only consistency that is needed for that purpose is within the team and the project. The members of the Agile Round Table found that some agile teams resented being asked to be consistent with other teams, wrongly (in our opinion) saying this goes against the agile principle that teams are self-governing.

Since agile teams are already using story points, the Agile Round Table looked at how that can be extended to get the consistency across projects needed for release level estimation without violating the self-governing nature of agile teams.

Choose a Set of Size Bins

One technique that promotes consistency is already in widespread use by agile teams. Many agile coaches recommend using a fixed set of size bins. Some use 1, 2, 4, 8, 16, 32 and so on. Some use a scale from 1 to 10. The Fibonacci series — 2, 3, 5, 8, 13, 21,… — is quite popular. This way, we do not waste time discussing whether a story is 5 or 5-1/2 story points; we can put slightly different sized stories in the same bin. The members of the Agile Round Table confirmed that most of their teams were doing this.

The SLIM-Estimate® and SLIM-Collaborate® template for Agile Story Point Estimation has an example of this (Figure 1):

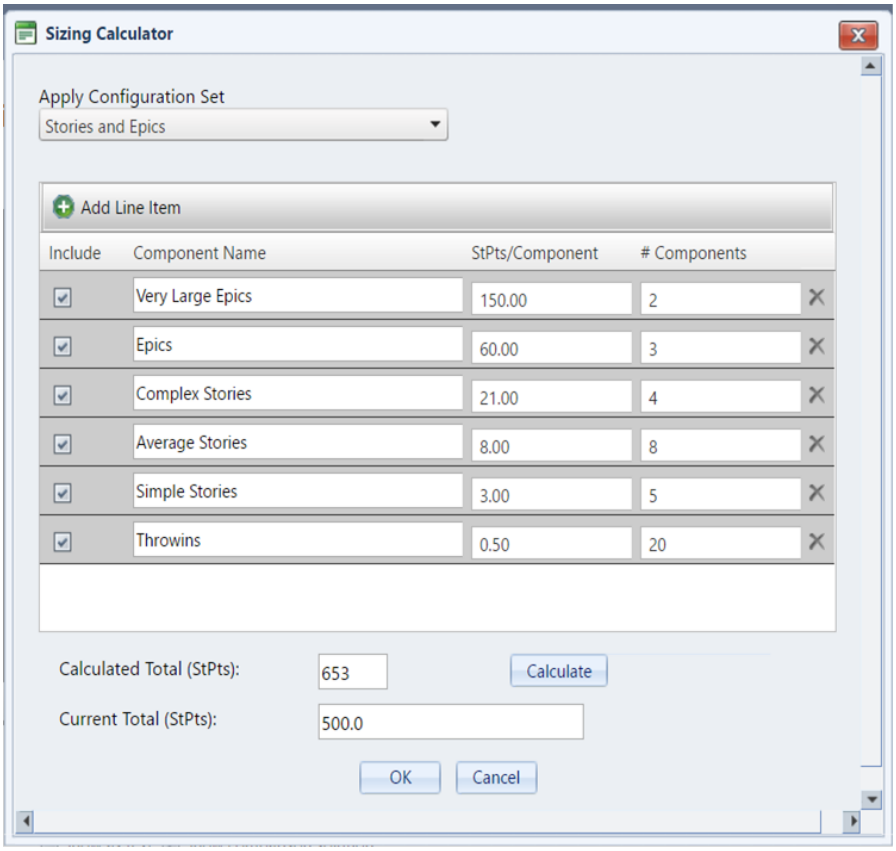


Figure 1. Agile story point estimation template.

In addition to bins for the detailed stories that get developed in each release, also settle on a set of bins for the "big rocks," as described in the previous article in this series, "Big Rock Estimation," in the section "Size Grounding" (available on the QSM® website at http://www.qsm.com). Since release-level estimation is often needed before many detailed stories have been defined, the items in the backlog you need to size may mostly fall into these larger bins.

Make sure the teams are clear that you are not asking for consistency just for the sake of consistency. Adopting a consistent set of size bins extends the work they are already doing for an additional needed purpose.

Once the advantages are explained, settling on a common scale and set of size bins throughout your organization promotes consistency without violating the principle of self-governing teams, and will likely be accepted by your organization.

Pick "Exemplar" Stories or Epics for Each Bin

For iteration planning, the goal of putting stories into the chosen size bins is that all stories from the project that are in the same bin are roughly the same size.

For release estimation, it is also important that stories in each size bin be roughly the same size, regardless of which team or project did the estimating. This kind of consistency can be difficult to achieve, since normally the backlogs on different projects are not compared to one another, nor estimated by the same people at the same time.

This consistency can be facilitated by picking example stories or epics to associate with each bin that will be familiar to the variety of teams that are doing the estimating. These "exemplars" will be used for estimating across projects and teams. When estimating the backlog of a particular project, instead of just comparing stories in the backlog to each other, you compare them to the exemplars to choose the appropriate bin. Your organization may be in a specific industry, like some companies represented in the Agile Round Table. In that case, the projects in your organization will likely have a common domain, such as insurance, travel, or automotive engineering. For each bin, you can find an example story or epic from your domain that will be understood by all teams to use as the exemplar for that bin. In other cases, like some other members of the Agile Round Table, you may be a systems integrator or other type of organization that is involved with projects from many different industries. In that case, you can still pick exemplars for each bin from areas that are commonly understood by many teams. While these are not exactly like the items in any particular backlog you are estimating, the teams can still use them for comparison.

Note that promoting consistency within a single organization is not the same as setting an international standard to promote consistency across organizations. We do not think it is likely there will be an effort to set such a standard for story points, though that could change as agile sizing methods become more popular.

Dealing with Emerging Requirements: Calibrate the Big Rock Bins with "Estimated Actuals"

Several Round Table participants said their organizations were consistently underestimating agile projects. A chief cause was the lack of detail on the requirements at the beginning of the project—the lack of a "big upfront requirements phase." Although epics were identified and sized, the size was underestimated because of the details that emerged only as those epics were refined from iteration to iteration.

Emergent requirements are a hallmark of agile methods, and using this technique is probably the most important factor in, well, agility. This strength of agile methods means that to estimate in advance, you must be able to size accurately enough with only very high-level requirements. This is the basis of the technique described in the earlier article in this series, "Big Rock Estimation." The bins chosen for the big rocks must be assigned story points in order to get a total size.

To promote consistency, the bins and exemplars should remain stable over time. Course corrections can be achieved by adjusting the number of story points assigned to each bin. Here is how this works:

An agile project may start off with a backlog consisting mostly of big rocks. As the project progresses, those big rocks are broken down into smaller, more detailed stories of various sizes. This forms a hierarchy of sorts, from the big rocks down to the developer-sized bites which actually get developed. Many agile planning tools support tracking this hierarchy, or you can track it manually. Typically, then, agile teams will assign story points to each of those developer-sized bites to plan the iterations in which they are developed.

There are, of course, other changes besides breaking large epics into small stories. Some stories are added to the backlog that were not broken out from the epics originally on the backlog, and other priority and scope changes mean that the hierarchy is not perfect. But if you keep this information from a variety of projects, the individual changes will "average out" and that is good enough for the calibration.

For each epic in the original scope, find all the developer-sized bites that were derived from that epic. That is, find the small stories that actually made it into development that were broken out from that epic. Each of these actual stories was assigned a number of story points during iteration planning. Keeping in mind the typical rule that you don't re-estimate after a story is developed, we will call these "estimated actuals." Total up these story points and track that total with the original epic or big rock, along with which bin to which that big rock was assigned (Figure 2).

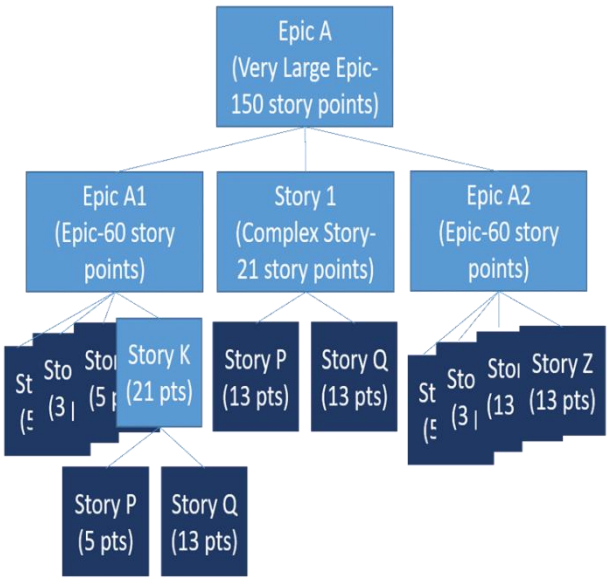


Figure 2. Add up the story points assigned to the detailed stories actually developed (illustrated in the darker color) and track that as the "estimated actuals" original "big rock."

Over time, you will collect data from multiple projects about a variety of epics within a single bin. While the bins you use stay the same over time, you can recalibrate the number of story points you assign to an epic in the bin for new estimates, perhaps, by taking the median or average of the estimated actuals for the completed epics already in the bin.

Summary

We have described some techniques you can use to measure size of agile projects consistently enough to use for release estimation. You may need to overcome some objections the team may raise regarding some of these techniques. To do this, you must be clear about the purpose and value of what you are asking the team to do.

Using story points to measure size for release estimation fits with what the team already does for iteration planning, but you have to achieve consistency across your organization. Some key techniques to gain this consistency are "Big Rock Sizing," getting agreement on a set of size bins to use and using "exemplar" stories to compare to stories and epics in your projects. However, there is likely not as much consistency from organization to organization, which presents challenges for using industry-wide data.

We presented a method to calibrate the size used for the larger bins by tracing the big rocks used for initial sizing to the actual stories derived from them, thus getting historical data about the total size in story points actually developed for epics in a particular bin. This calibration can improve the accuracy of your estimates as your organization gathers data and gains experience.

References

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

Derived PI: Is PI from Peak Staff "Good Enough"?

Paul Below

- Are you having a hard time collecting total effort for SLIM® Phase 3 on a completed project?
- Can you get a good handle on the peak staff?
- Maybe we can still determine PI!

It is difficult and often time consuming to collect historical metrics on completed software projects. However, some metrics are commonly easier to collect than others, namely, peak staff, start and end dates of Phase 3, and the size of the completed project. Asking these questions can get things started:

- So, how many people did you have at the peak?
- When did you start design and when was integration testing done?
- Can we measure the size of the software?

That gives us the minimum set of metrics to dig up. However, the PI (Productivity Index) formula also requires Phase 3 effort. Can we use SLIM® to generate a PI that is useful, using peak staff instead of total effort? A statistical test on historical metrics can answer this question.

What are we comparing?

- Projects used in this study had all four of the following: actual reported effort; size; peak staff; and duration.
- For each project, a derived effort is generated from peak staff, size and duration.
- A derived PI is generated from the derived effort, size and duration. This derived PI is then compared to the actual PI.

Definitions for terms:

- Derived PI is estimated PI from the reported size, duration and peak staff, using SLIM-Estimate® to do a "solve for PI" to fit a Rayleigh Curve to the duration and peak staff thereby generating a derived total effort.
- PI is the actual PI calculated by SLIM-DataManager®.

To generate the derived PI, use the "Solve for PI" wizard in SLIM-Estimate. For example, if the historical project had a 24-month duration, 40-person peak staff, and a size of 500,000 implementation units (Figure 1):

Figure 1. "Solve for PI" wizard screenshot from SLIM-Estimate®.

The "Solve for PI" wizard derives 122k person-hours and a Derived PI of 17.2 (using the default medium front load staffing curve shape). For comparison, the actual PI was 17.9. Full solution shown at Figure 2 below.

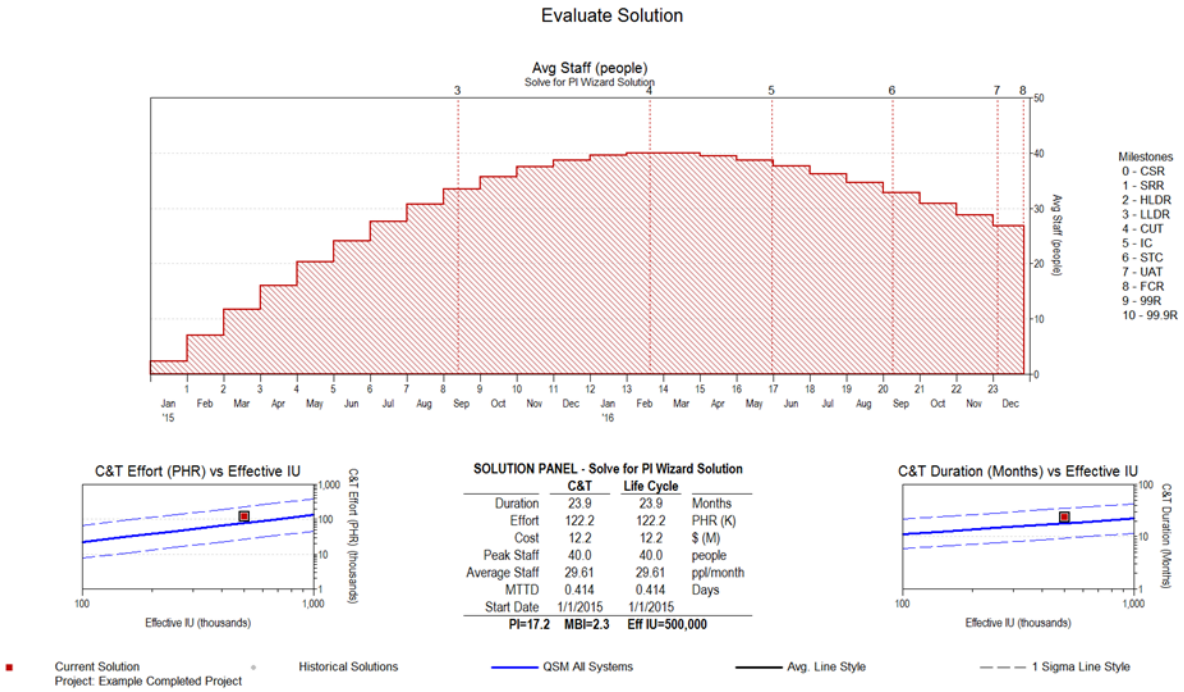


Figure 2. Derived solution from "Solve for PI" wizard for example.

For this statistical test, I used a sampling frame of projects in the QSM® Database that had all these:

- Effective SLOC
- Main Build Duration
- Main Build Effort
- Main Build Peak Staff
- PI

I took a random sample of 93 projects from the frame. (Why 93? Statistically, I needed at least a dozen, so I kept going until I got tired! But seriously, that is more than enough to provide a valuable answer to our question[1].)

Comparing the derived PI to the actual PI for each of the sample reveals a high correlation (0.971) (Figure 3). Which is promising, even though the means and medians are slightly different (blue ovals on Figure 4). We need to dig deeper.

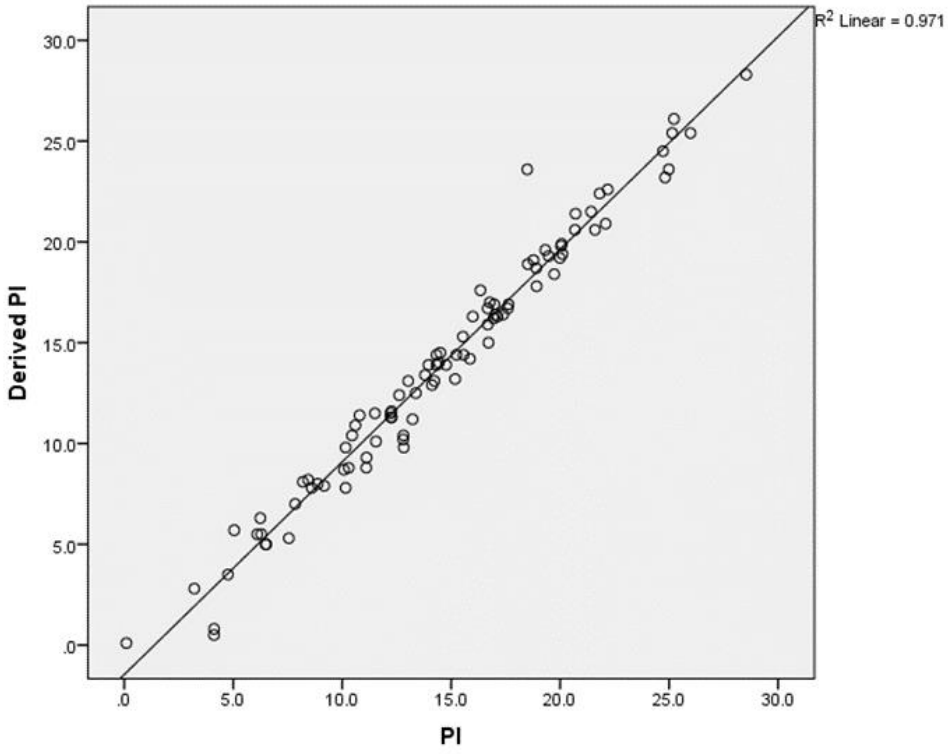


Figure 3. Comparison of derived PI to actual PI for each of the sample.

Statistics

		PI	Derived PI
N	Valid	93	93
	Missing	0	0
Mean		14.592	13.933
Std. Deviation		5.8284	6.2392
Skewness		-.014	-.014
Std. Error of Skewness		.250	.250
Percentiles	25	10.527	9.550
	50	14.432	13.900
	75	18.839	18.800

Figure 4. Mean actual PI and mean derived PI comparison.

First, a short digression or reminder about paired tests.

- In a **paired** test, samples are collected in pairs of values, such as before and after measurements, or, in this report, an actual and a derived PI for the same project.
- An independent-paired-samples Student's t-test **compares the mean value of the differences between the paired samples**.
- The null hypothesis is that the difference in means is zero.
- The test that rejects the null hypothesis indicates that we have evidence that the performances are different. We then conclude that the difference is unlikely to be due to random variation.

Got that? OK, onward to the numbers.

Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	PI	14.592	93	5.8284	.6044
	Derived PI	13.933	93	6.2392	.6470

Figure 5. Paired samples statistics.

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	PI - Derived PI	.6591	1.1069	.1148	.4311	.8871	5.742	92	.000

Figure 6. Difference between the mean actual PI and mean-derived PI (95% confidence interval of the difference).

The mean-actual PI of the sample was higher than the mean-derived PI (Figure 5 above). Statistically, we end up with a range of between 0.4 and 0.9 (blue oval on Figure 6, also above) for the difference between the mean-actual PI and the mean-derived PI (95% confidence interval of the difference).

Concluding that the actual PI and the derived PI are not equal, what should we do? The high correlation between the actual and derived PI means that we CAN use peak staff to estimate PI, with a correction.

One adjustment method would use a linear regression to improve the match. The adjusted R square of the model is a robust 0.971 (Figure 7).

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.985[a]	.971	.971	.9952

a. Predictors: (Constant), Derived PI

b. Dependent Variable: PI

Figure 7. Adjusted R square of samples.

The model provides this equation:

*Adjusted Derived PI = (Derived PI * .921) + 1.77*

Figure 8 contains the particulars.

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B	
		B	Std. Error	Beta			Lower Bound	Upper Bound
1	(Constant)	1.766	.254		6.961	.000	1.262	2.270
	Derived PI	.921	.017	.985	55.355	.000	.888	.954

a. Dependent Variable: PI

Figure 8. Model coefficients of samples.

If the equation is too complex, an easier adjustment would be to just increase the derived PI by the sample mean difference, which we have seen was 0.66.

In conclusion, we can derive a useful estimate of PI from peak staff where the actual effort is unknown. This will allow you to increase the number of historical projects with PI in your database even when you cannot collect actual effort.

References

1. Statistical Rules of Thumb, Gerald van Belle. Wiley, 2002. Pages 18-19.

Measuring Effort and Productivity of Agile Products

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 one in a series of articles resulting from this forum.

Measuring Effort

Measuring effort means the same thing in agile and non-agile methods: you need to know the hours spent by people working on the release.

There are two subtle issues, though:

a. Which people should track the effort they expend on the project?
b. Do you track effort based on the phases in the SLIM® methodology?

Which people? Agile methods refer to "the team," which usually means the people who write code and run test cases. For the purpose of estimating and tracking effort, the product manager and scrum master are included. But there are other people that expend effort on the project: specialists who share time on multiple products, perhaps specialized DBAs or Enterprise Architects, documentation and training materials developers, testers from an enterprise test group that must pass on every delivered product, and other such resources that contribute time and, thus, effort to a release. And don't forget subject matter experts and other stakeholders who participate in backlog definition, backlog grooming, and sprint reviews. Although they are not full time on the team, the effort of all these people is critical and contributes to the cost of the release.

What phases? The SLIM methodology divides a project into four phases. The term "phase" has a negative connotation for some people in the agile community. It's the same word that's used in "big upfront requirements phase" or "big upfront design phase" that is anathema to many agile methods. In the SLIM methodology, however, "phase" has a different meaning. The four phases in a SLIM project do not imply non-overlapping sequential activities. Instead, in the SLIM methodology, the phases represent different types of work that are carried out in a project, with different characteristics of how resources are used. The time periods during which those types of work are carried out can be configured for a particular project with what the SLIM method calls "phase tuning." In agile projects, the key types of work that matter for effort tracking are work on requirements, which corresponds to Phase 2 in the SLIM methods, plus work on development and testing of the code and

other needed materials, which corresponds to Phase 3. In an agile project, these types of work go on concurrently throughout the release, and SLIM phase tuning is configured to reflect that.

The SLIM methods rely on history of completed projects to calibrate estimates and forecasts, either from your own organization or industry wide data from the SLIM database. Effort is one of the key metrics. In the SLIM tools, you to track effort by phase (that is, by type of work).

The effort expended by team members and stakeholders on getting stories ready to develop, including writing the stories, grooming the backlog, and prioritizing should be tracked as effort expended in the Story Writing Phase (Phase 2 in the SLIM tools). Very importantly, this also includes the time spent on conversations among the stakeholders and development team to refine the details of the stories. Effort expended on detailed design, developing the code, refactoring, testing, debugging, and other related activities should be tracked in the Story Development Phase (Phase 3).

Since these activities occur concurrently, the tools you use to track effort may need to be used creatively to make this distinction. For example, a time-tracking tool that just lets a team member track which hours each day were spent on the project may not split out the time, since there's no set distinction of which days and which hours were spent on story writing, and which days and hours were spent on story development. On any given day, work was likely done on both. Many agile tracking tools let team members track the time spent on a particular story by having a sub-record of the Story for each team member to use for time tracking. Several participants in the Agile Estimation Round Table indicated that the members of agile teams use such tools. Most of these tools allow customization that would let team members to track two effort numbers for a story, splitting out the effort on story writing (including, most importantly, the conversation time about the details) and the effort on development.

Note that very precise time tracking is usually not practical and making the distinction between story writing effort and story development effort makes that even more difficult. Fortunately, since SLIM develops trends from multiple projects, very precise tracking is not needed for the SLIM methods, so to be practical, team members can split the time they spend by their best guess of the percentage spent on story writing versus story development.

Tracking effort by story within the agile tracking tools is often not available to stakeholders who are not part of the core development team. In this case, a product owner may estimate the actual time spent by stakeholders on story writing (and development if applicable) and track that separately, perhaps as attributes of the story itself in the tracking tool.

No matter how you track the time, collating it and entering it in the SLIM tools can either be done manually, or may be automated through the use of the SLIM API (application programming interface) together with your tracking tool's API. Through such automation, and by appropriately sacrificing precision for ease of use, you can track the effort expended throughout an agile project on story writing and story development. This provides the historical basis for future estimates.

Measuring Productivity

Productivity in software development is surprisingly complicated to measure or even to define. We think we intuitively know what it means to be productive: a team can get a lot done in a short period of time. Or does it mean that a team can get a lot done without expending a lot of hours of effort? And what constitutes "getting a lot done"? Writing code? Testing the code? Efficiently defining requirements and building in quality so software is developed right the first time? With waterfall methods, producing an elegant, robust design may be very productive even though no actual code is running, but that's unlikely in an agile environment. And once you decide on what productivity means, how do you measure it?

Agile methods simplify some of these issues. Since working software is the measure of progress, and since the project progresses in time-boxed iterations, the number of stories developed per iteration, known as the Team Velocity on a project, provides some measure of productivity. Many agile teams use Team Velocity for iteration planning, where "number of stories developed" is the total Story Points for the stories completely developed in an iteration. The best guess of the Team Velocity you can achieve in the next iteration is that it will be about the same as the Team Velocity of the previous iteration, and that helps the team plan which stories they can tackle in an upcoming iteration.

However, when you switch from Iteration Planning on a specific project to Release Planning for a portfolio of projects, Team Velocity becomes quite problematic as a measure of productivity. There are many reasons for this including:

1. ***Velocity is not constant.*** The Team Velocity from iteration to iteration will change fairly slowly, especially if iterations are short, which is why it's useful for iteration planning. But Team Velocity does change through the course of projects in a systematic way. The fundamental research in project estimation, pioneered by Larry Putnam Sr., the founder of QSM®, showed that software development projects follow a basic shape, called the Putnam-Norden-Rayleigh curve, and there is an ebb and flow (or more precisely a "flow and ebb") to projects. Research by QSM verifies that this is true of agile projects as well as other methods. Most agile methodologists are implicitly aware of this; they suggest waiting a few iterations before starting to compute velocity because velocity typically starts low and grows for a while, then starts changing slowly (growing and then ebbing) in the middle of projects, and then, usually, velocity starts going down towards the end of a project as a release is "hardened." It's in that middle stage where the velocity changes slowly that it's useful for iteration

planning. This "grow, change slowly for a while, reduce" pattern is typical of the Rayleigh shape of projects. The cumulative effect is a familiar "S-shaped" curve, shown in Figure 1:

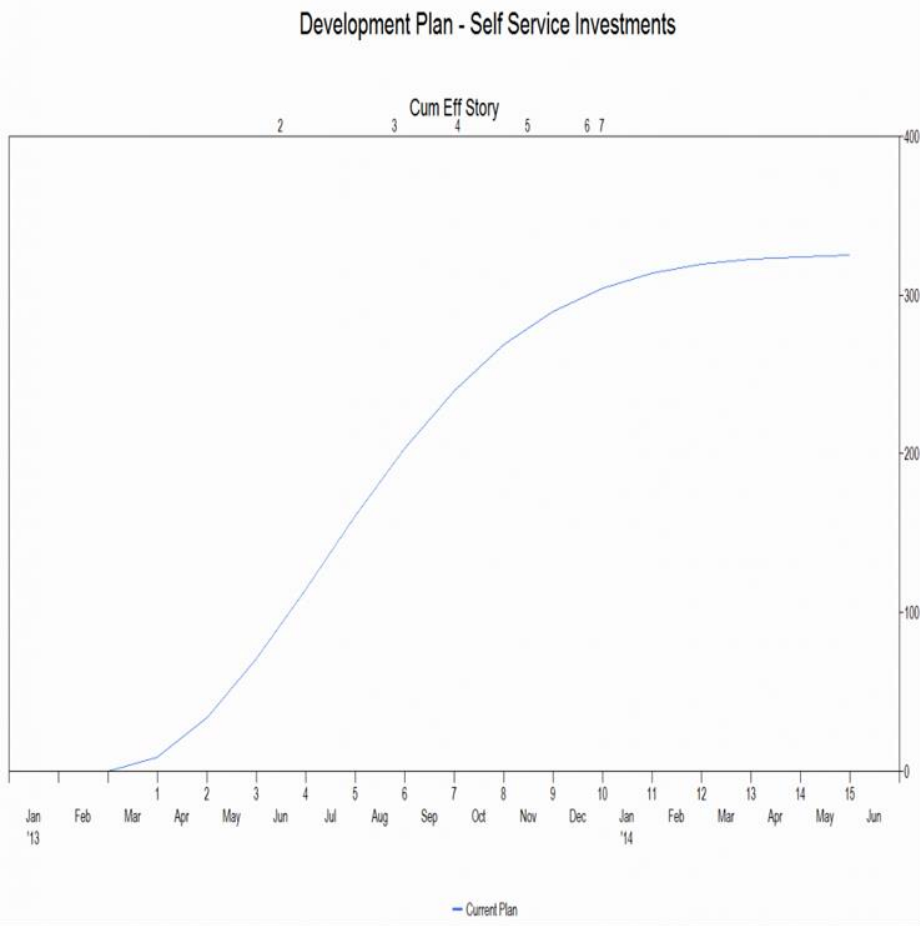


Figure 1. Reliability risk curve.

In particular, many companies are using "agile at scale" methods (e.g., SAFe or other large-scale agile methods). Several examples from members of the Agile Round Table showed the distinctive Rayleigh pattern when you look at development by teams of teams.

2. Velocity depends on team size. While most agile methodologists and the results of QSM's research recommend small team sizes, there is no single "ideal" size, and team sizes will vary from project to project, in part depending on whether time pressure or cost is paramount. Thus, the Team Velocity attained by a five-person team on a project cannot be used to predict the Team Velocity of a project that may be under more time pressure, so an eight-person team is used instead.

3. Velocity depends on the iteration length. If one team is working in two-week iterations, and another team is working in three-week iterations, their Team Velocities will be different, even though there may not be much difference in how long or how much effort comparable releases take.

4. Velocity is not linear. Remember "The Mythical Man Month"? The relationship between software size, duration of a project, and team size or effort expended is not a simple linear

relationship. A 10-person team, all else being equal, will NOT have twice the team velocity of a five-person team. And the amount of software a fixed team can develop in six months is not twice what the same team can develop in three months (it's likely to be more than twice!), so the Team Velocity (even assuming the same length iterations) is different. For these and other reasons, Team Velocity is not very successful as a predictive measure for Release Level estimation.

Instead of Team Velocity, the SLIM tools and methods use a more abstract measure of productivity called "Productivity Index." This is statistically derived from size, effort, and duration of completed projects, to produce a trend that compares Productivity Index to size of a project. This, when used with the Software Production Equation, compensates for the non-linear nature of software development and allows the SLIM tools and methods to adjust the Rayleigh shape of a project based on tradeoffs among size, effort, and schedule.

There are two ways to get a Productivity Index trend to use for Release Estimation:

a. You can use industry data. The SLIM tools come with a trend line based on projects from multiple customers doing agile development.
b. You can use your own data.

Most agile development shops recognize the importance of keeping metrics. The key metrics for computing Productivity Index trends are size, effort, and duration of projects. In the articles in the Agile Round Table series about size, we discuss that size measures in agile projects are often different for different companies. The computation of the trend for Productivity Index adjusts for that (as does Team Velocity, for that matter). So, if you use your own data, the size and Productivity Index form a matched pair.

Measuring Effort and Productivity Are Related

The Team Velocity of a project is derived from size and duration, but the Productivity Index of a completed project is derived from size, duration, and effort through the SLIM Software Production Equation. That is why this measure of productivity can be used for release estimation, including alternative estimates that understand "the mythical man-month" and the non-linear tradeoffs between effort and duration. Therefore, the effort tracking discussed in the first part of this article is important for measuring productivity, including the split of effort between Story Writing and Story Development.

By measuring effort for completed projects, as well as time and size, and using those metrics to derive a trend for the Productivity Index, you can meet business constraints with release estimates that can trade off cost and size of future projects, letting you plan how to best apply your resources to agile teams.

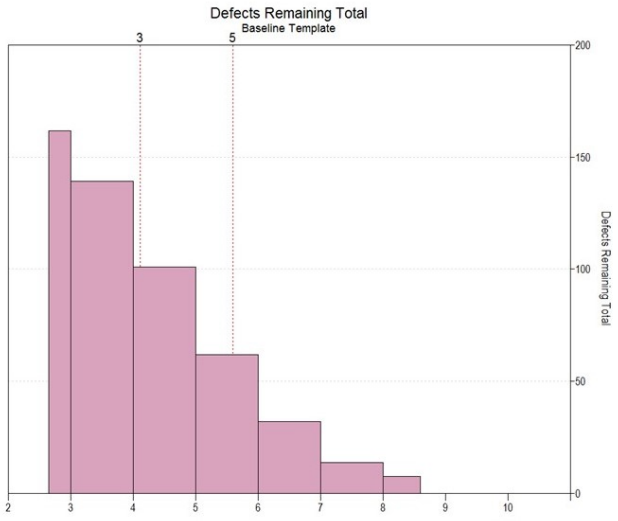
Agile On-Time, But Is It Reliable?

Keith Ciocco

With agile projects, we hear a lot about the planning benefits of having a fixed number of people with a fixed number of sprints. These are all great considerations when it comes to finishing on time and within budget. But one of the things we also need to focus on is the quality of the software. We often hear stories about functionality getting put on hold because of reliability goals not being met.

There are some agile estimation models available to help with this, and they can provide this information at the release level, before the project starts or during those early sprints. They provide this by leveraging historical data along with time-tested forecasting models that are built to support agile projects.

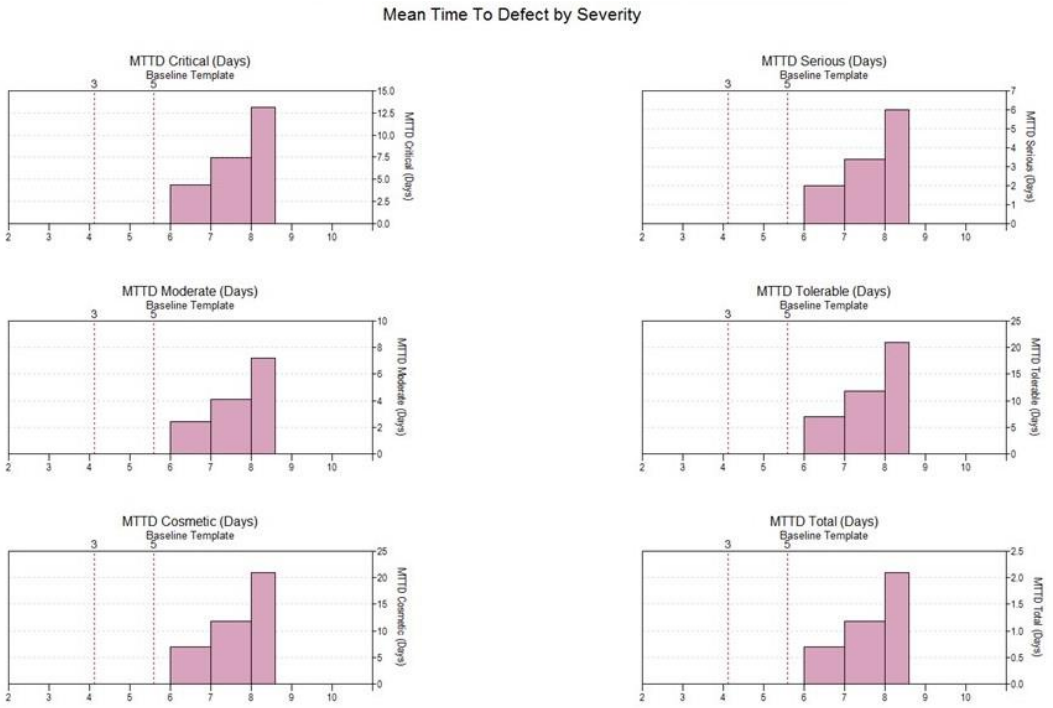
To better demonstrate this, I have examples of a project estimate. In Figure 1, you can see a project estimate showing the total number of defects remaining for the duration of the project development. This is a "big picture view" of the overall project release. Product managers and any other stakeholders invested in client satisfaction can leverage these models to predict when the software will be reliable enough for delivery to the customer.



MTTD over Time

Figure 1. Total defects remaining.

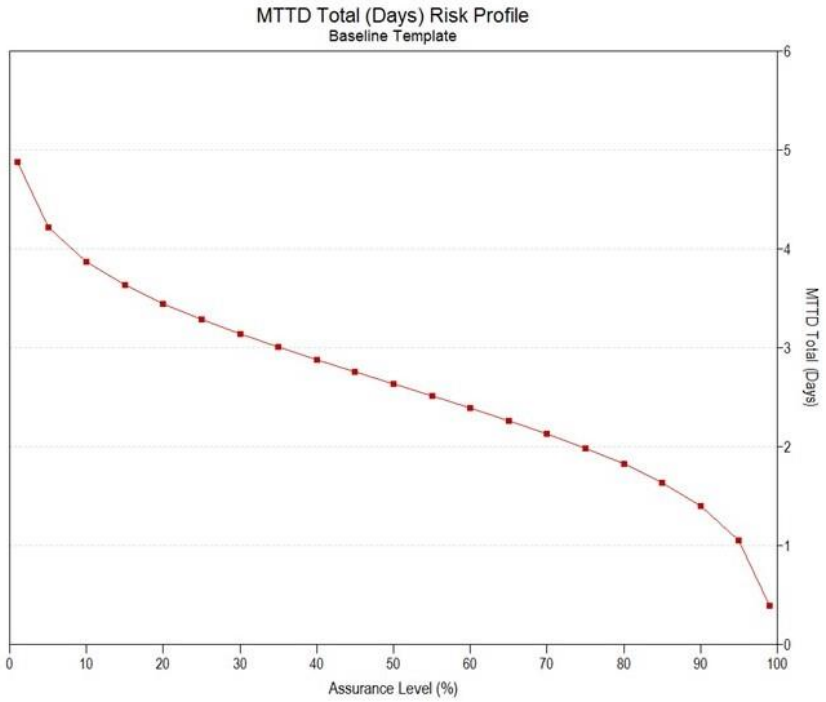
Figure 2 shows the total Mean Time to Defect (MTTD) and the MTTD by severity level, with the MTTD being the amount of time that elapses between discovered defects. Each chart shows the months progressing on the horizontal axis and the MTTD (in days) improve over time on the vertical axis.



Mean Time to Defect

Figure 2. Total Mean Time to Defect and Mean Time to Defect by severity level.

These models additionally provide a valuable view of the risk associated with the software reliability. Finally, in Figure 3 below (also sometimes referred to as an "S-curve" chart), you can see that there is a 99% chance that the MTTD will be less than half a day or, stated alternatively, that the software will only run part of a day without being impacted by a defect. If these are critical defects, or if the software is critical (such as aircraft or life-sustaining hospital equipment), then we have some serious concerns to deal with. So, this would definitely not be a good time to deliver the software, even though the fixed number of sprints is set to be completed – a red flag to a product manager.



Agile Project Reliability Risk Profile

Figure 3. Agile project reliability curve (or "S-curve") risk profile.

As you can see, defect estimation is extremely important with agile as well as any other type of software development method. We need to be able to see the release-level defect estimates before we agree to a delivery date and budget. What good would it be to deliver on time and within budget if the software is not going to work to a high enough standard of customer satisfaction?

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

Alternative Sizing Units for Agile Estimation

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 one in a series of articles resulting from this forum.

Function Points for Agile Estimation

Function points have been used as a measure of size for estimation for many years. There are benefits and challenges to using function points as the unit of measure for agile projects. Note that function points share with story points the characteristic that there's no such thing as "one of them." While you assign function points to scope items and add those up to get a total, there's no formal definition of a single function point.

There are several reasons function point counting has been successful for sizing projects for estimation. This includes the specific way the scope is decomposed for function point counting that helps account for the degree of shared code. Perhaps the biggest benefit is that function point counting has an internationally agreed upon standard, giving it the consistency needed for estimation. Of course, that's the reason the standard was developed and why many companies invest in training function point counters. Actually, there is more than one type of function point counting and more than one standard, but, for the purposes of this article, they are similar.

One challenge several of the Agile Round Table members found when using function points for estimating agile projects was gaining acceptance from the team. Some agile teams considered function points "old fashioned" and felt that, as a result, it was not applicable to agile projects. Related to this is that, unlike story points, the agile team probably will not use the function point counts for any purpose beyond the release estimate, so it is not considered part of the team's agile methods. In particular, epics and stories do not correspond directly to the decomposition of a project's scope that is needed for function point counting. You cannot count the function points in

each story and add them up, since the scope elements to which function points are assigned overlap multiple stories. This can be mitigated by doing the function point count in parallel to story points or other techniques for iteration planning, limiting the use of function points to release estimation and tracking overall progress. Although several members of the Agile Round Table did use function point counting extensively, there was less prevalence of recounting actual function points at the end of a project.

The biggest challenge, though, to using function point counting on agile projects is the agile technique of emergent requirements and the lack of scope detail at the time when we need to estimate the release or project. This is the issue addressed for story points by "Big Rock Sizing," and it creates some specific issues that must be overcome when using function points.

The standards for function point counting require a level of detail about the requirements not usually available early on in an agile project. For example, when you count function points, you count the data stores you expect to use and assign function points based on the approximate number of data elements. In an agile project, you will not have this detail early. In some cases, you can take a smart guess. If you are building an airline reservation site, you know you will have a data store for airports, for flights, for available seats, for reservations, and several others. But the complete list of data stores only emerges as the project progresses and the big rocks are refined iteration by iteration. And even for the ones you can guess at, you will not know the number of elements, even approximately, to use when assigning function points. Forcing the team to think about these details early goes against the main agile technique of emergent requirements. Even "taking a good guess" biases the team for later, when detailed decisions are needed.

The members of the Agile Round Table suggested two mitigations for this problem. The first was the use of "function points lite," as one member of the Round Table called it. Instead of formal function point counting, including assigning function points based on the detailed complexity of each of the scope elements, you assume each element has average complexity. You can also count the obvious elements without biasing future emergent requirements and buffer the count to deal with the less obvious elements.

The second was "counting on the side." This mitigates the team's resistance to using function points for their day-to-day activities and is particularly useful for tracking progress in a project and forecasting completion as the project progresses. An initial function point count is done based on the high-level scope (perhaps using "Function Point Lite"), and as the scope is refined by breaking epics down into stories, a separate tally of function points is maintained, modified as the backlog is groomed.

Source Lines of Code for Agile Estimation

Agile projects, like all other software development, produces code as an end product, and counting that code gives a measure of size. Counting code is perhaps the most concrete measure of size of a project. Unlike function points and story points, there is a standard definition of a single source line of code, even though it is surprisingly complex.

Counting source lines of code fell out of favor with many development teams long before agile methods became popular. There are many reasons for this, some good and some more emotional than logical. A more complete discussion of this is outside the scope of this article. But counting code

still remains a viable way to compare the size of projects, and perhaps the best way to get standardized data from many different sources, so it is particularly valuable to compare your organization's projects to industry data. And since it is independent of the way the scope of the project is expressed (e.g., as epics and stories versus the breakout used by function point counting), counting finished code allows you to compare projects developed with agile methods to projects developed in other ways.

The main disadvantage for using code counts for early estimation is not unique to agile methods. Since it is not widely used for any purposes anymore except estimation, the project team members are particularly inexperienced at counting total code. Ask most developers, "How many lines of code do you think that will take?" and you will be lucky if you just get a blank stare.

For counting code of finished projects, you can use automated code counters that work with the code files themselves or sometimes with the version control systems that your team may use. Thus, you can get code counts "after the fact" to use to calibrate your estimates using other size measures, as we have described earlier in this article.

Summary

As with projects built by other methods, function points can be applied to agile projects consistently because of international standards. Agile teams may be resistant, however, because of the perception that function points are old fashioned. More importantly, the lack of detailed requirements early in a project presents a challenge to rigorous function point counting.

Source lines of code is a concrete measure that can be used consistently and can help compare your projects to industry-wide data. You can automate counting the size of completed projects, thus getting a good historical base of data for future estimates (Figure 1). However, the lack of experience in this measure makes it difficult to use in advance for release estimation when estimating a new project.

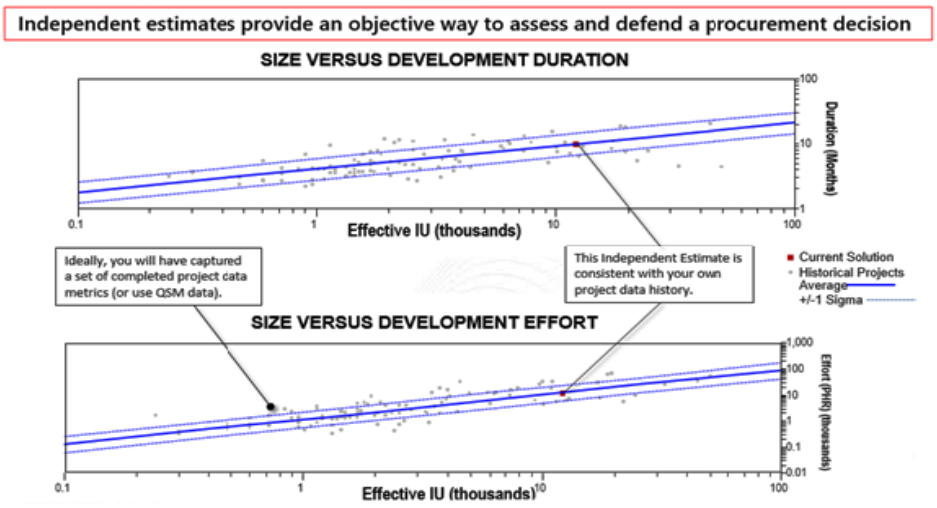


Figure 1. Scatterplot of independent estimate compared against historical data for schedule and effort to assess the proposed project plan.

3. MANAGEMENT

"Instead of freaking out about these constraints, embrace them. Let them guide you. Constraints drive innovation and force focus. Instead of trying to remove them, use them to your advantage."

–Basecamp (formerly 37Signals),
a privately held American web application company

"The goal is to turn data into information, and information into insight."

–Carly Fiorina,
American businesswoman and political figure, known primarily for her tenure as CEO of Hewlett-Packard

"A person who is gifted sees the essential point and leaves the rest as surplus."

– Thomas Carlyle,
Scottish philosopher, satirical writer, essayist, translator, historian, mathematician, and teacher

"The value of an idea lies in the using of it."

– Thomas A. Edison,
American inventor and businessman, described as America's greatest inventor

"Never confuse motion with action."

– Benjamin Franklin,
One of the Founding Fathers of the United States and leading author, printer, political theorist, politician, freemason, postmaster, scientist, inventor, humorist, civic activist, statesman, and diplomat

Defense Trend Lines: Leveraging the Power of Historical Data

Taylor Putnam-Majarian and
John A. Staiger, Jr.

Developing software within the Department of Defense (DoD) presents a unique set of challenges. The environment is such that many of the programs are unusually large, complex, and schedule-driven, and typically require several years to develop, field, and deploy. Often, they are maintained for over a decade, during which time several advancements in technology, unknown at the inception of the program, must be accommodated into the existing system. It should come as no surprise, therefore, that cost estimators have faced significant challenges when estimating systems in the Defense arena.

The QSM® team has supported a variety of levels within the DoD, from program management offices to the Secretary of Defense. One such organization's recent initiative was to improve its estimation process by leveraging historical data collected from forensic analyses of recently completed software development efforts. This article discusses (1) some of the challenges faced throughout this initiative, (2) the data collection process, and (3) how one can leverage that data to improve cost estimates.

Challenges

The Spanish philosopher, George Santayana, is credited with the observation that "… those who do not learn from their mistakes of history are doomed to repeat them…" This astute piece of wisdom has been paraphrased and quoted many times, but it is especially relevant in the DoD environment due to the unique challenges posed by the size and complexity of DoD systems. To learn from past mistakes, it is important to review the specific actions, assumptions, and information that were used and available. Thinking back to a completed project that was considered successful, what were some characteristics that allowed it to do well? Conversely, thinking back to a project that was considered less than optimal, what should have been done differently? Perhaps this type of information is readily

available informally, but it hasn't been captured in a quantifiable form. Hence, the need to collect historical data to objectively assess the process and quantify the actual performance.

Collecting historical data, however, can sometimes be a sensitive subject. It is not uncommon for the QSM team to face some resistance during the initial phases of the data collection process. One of the main challenges faced is a fear of measurement. Stated simply, people do not want to be viewed in a less than successful light, whether to their peers or senior leadership. Even when projects have performed well, some still may believe that any data provided could potentially be used against them. If the analysts can overcome this first challenge, they are often faced with a second, which is that no data is available. Finally, historical data is often competition-sensitive, usually the case within the DoD environment. These challenges are typically navigated by utilizing data from QSM's 13,000+ project database comprised of completed industry programs, rather than an organization's actual historical performance data. A subset of comparable projects is used in lieu of client data, if there is nothing better that is either available or provided by the organization (often due to contractual limitations).

Before discussing the process of how historical data is collected, it is important to first discuss why this information should even be collected. At QSM, historical data serves as the foundation for quantitative, data-driven estimating, tracking, and process improvement. First and foremost, collecting historical data promotes good record-keeping. While data-collection is often a standard practice among DoD systems, it is most often decentralized or has not been properly validated, therefore making it difficult to find the exact piece of information needed without searching through multiple documents or having a very low confidence level attached to it. Consolidating historical data into one location, with appropriate levels of protection consistent with their associated programs, makes it easier and more efficient to reference this information later. Secondly, historical data can be used to baseline and benchmark other historical projects to determine which projects were completed successfully and which ones could be improved. Finally, and very important for DoD "budgeteers," historical data can be leveraged for producing defensible, quantitative estimates for future projects. The following sections will discuss each of these points in greater detail.

Collecting Historical Data

How does one go about collecting historical data? Beginning this process can be intimidating, especially if historical data has not previously been collected. To simplify the process, one should instead focus initially on collecting just a few key pieces of information, adding more detailed metrics to the repository later.

For each phase of development, the start and end dates are collected, and the schedule duration is calculated. Most often, this is represented in months, but often weeks are used if the project is especially short or if agile development is used. Next, data on the effort expended is gathered, which is most often represented in person-hours or person-months (or man-month/staff-months), as well as the peak number of staff working on the program. The last necessary piece of information is project size. This is typically captured in the form of source lines of code, function points, business or technical requirements, agile stories, or RICEFW (Reports, Interface, Conversion, Enhancements, Forms, and Workflow) objects. Together, these pieces of information can be used to examine the project holistically.

Where can the required data be found? When beginning the data collection process, it is important to identify potential data sources or artifacts. This part of the data-collection process can

be like an archaeological dig. There will be a lot of data and information, but not all of it will be relevant or useful. However, it is necessary to sort through all of it to determine which pieces of data are relevant and necessary. In some cases, data must be acquired indirectly or derived using the artifacts and information that is available, particularly if the primary data is missing, unavailable, of questionable confidence, or if confirmation of the key metrics is needed. Typically, requirements documents contain some early sizing information, and code counts can be run at the end of the development effort to capture a more refined sizing measure. Software Resources Data Reports (SRDRs), Cost Analysis Requirements Descriptions (CARDs), and Automated Cost Estimating Integrated Tools® (ACEIT®) files, as well as program briefings, are all useful for collecting schedule and effort data. Moreover, if vendors are required to submit their data (via contract data requirements lists, or CDRLs), Microsoft Project® files, CA Project Portfolio Management® (formerly Clarity®), and other project portfolio management (PPM) exports can be sources of additional valuable data to verify staffing, schedule, and effort.

It is beneficial to have multiple source options for obtaining information. Many of these documents will undergo several iterations and may be updated at each milestone of the acquisition lifecycle. Unless the data appears questionable or invalid, it is generally best to use the data from the most recent version of the source document. Having multiple source options allows for validation, derivation of key metrics, and contextual information, all of which serve to maintain the integrity and increase the value of the historical performance database.

Putting Historical Data to Use

As an example, QSM's support team went through the process of collecting and validating a set of data from recently completed Defense Enterprise Resource Planning (ERP) projects. Within the DoD, ERP programs are those systems consisting of computer hardware, computer software, data, and/or telecommunications that perform functions such as collecting, processing, storing, transmitting, and displaying information, and are above specific cost thresholds or designated as a special interest program. With this data set, the team could create a set of ERP trend lines using the SLIM-Metrics® tool, as shown below (see Figure 1).

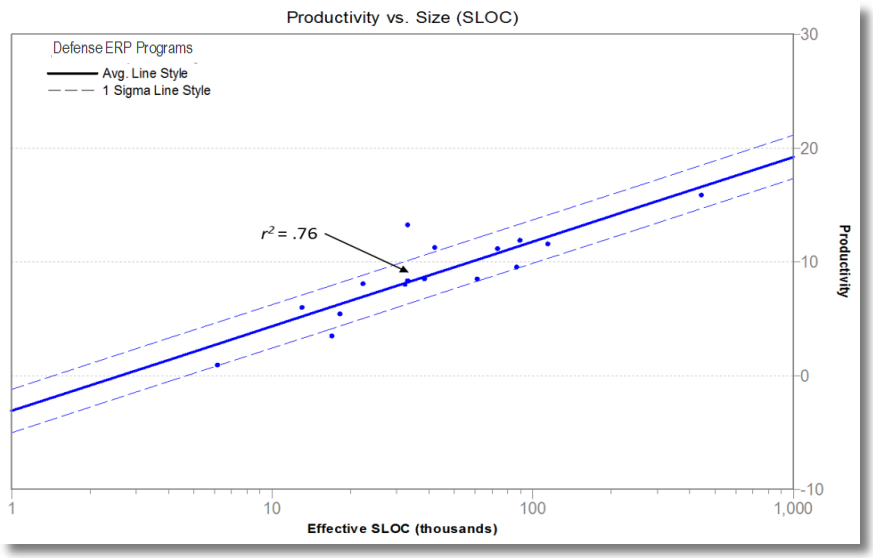


Figure 1. Defense ERP productivity trend lines.

Although these 16 data points came from three distinct vendors, notice that the r^2 value equals 0.76, indicating that the relationship between size and productivity for these releases was highly correlated. This type of correlation is typical when examining data points from a single organization. Seeing the strength of this correlation, particularly with three different vendors, indicates that this trend line would be appropriate for analytical use.

Note that Figure 1 demonstrates the correlation between size and productivity. Additional trend lines and correlations can be developed for numerous other metrics, such as effort, duration, defects, staffing levels. The development of trend lines is limited only by the types of data that are collected.

Using Historical Trend Lines

Once a set of custom trend lines has been developed, there are two primary ways in which they can be used: 1) for analyzing and benchmarking completed projects, and 2) for estimating future releases. Each of these will be discussed further in the sections below.

Assessing past performance. The use of historical trend lines can be instrumental in accurately assessing the performance of completed projects. Figure 2 shows the staffing trends for Defense ERP programs, plus or minus three standard deviations (dotted lines) from the mean average (solid line). The data points plotted on this chart represent notional completed projects submitted by a notional contractor.

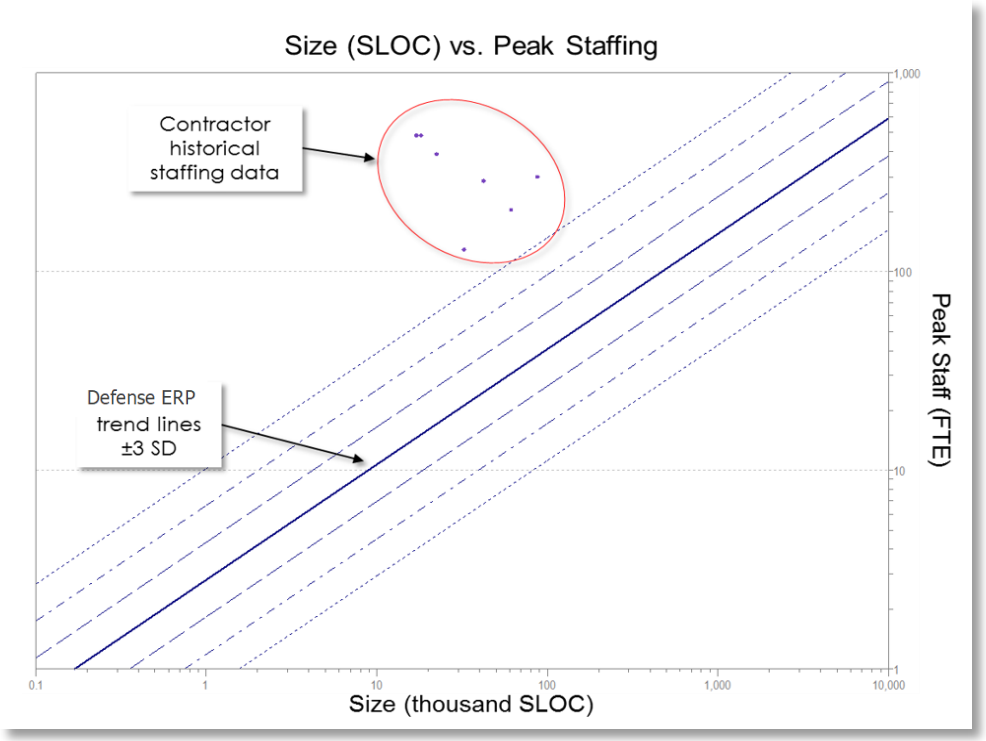


Figure 2. Contractor data reveals that historically completed projects use staffing levels three standard deviations higher than comparable Defense ERP programs.

When compared with other Defense ERP systems, this contractor is using significantly more people to complete its releases. Upon initial examination, this chart raises some questions. Why did this contractor need so many people when other vendors delivered the same functionality with a

fraction of the staff? Might there be a reason for this behavior? Perhaps the contractor was behind schedule and hoped that adding more people would help the program complete on time. Was this a time and materials or a fixed price contract? Perhaps this contractor used large teams as a means of billing more work if the contract type was favorable to this strategy. These questions are justifiable and pertinent, since actual, historical data is demonstrating a large variation from what might be expected (such as shown in an independent estimate).

When conducting benchmark analyses, there are at least two areas that should be examined. First, it is recommended that an assessment of the following core metrics be conducted: productivity, duration (time to market), effort expended, staffing, and reliability/defects (if data is available). Second, these metrics should be compared against industry trend lines and the distance between the two points should be quantified.

These types of analyses can shed light on the various factors that might have impacted project performance and can provide clues as to what adjustments can be made to improve processes in future projects. This same technique can also be applied to the vendor selection process. If all potential vendors were required to submit data from comparable completed projects, the acquisition or source selection office would be able to plot these data points against industry trend lines to validate the submitted bids. This would serve to determine the likelihood that the vendors can complete the work per their proposals and demonstrated historic performance, as well as to determine whether the bids are feasible against the government's independent (or "should cost") estimate and within its expected schedule and budget. Quite obviously, this serves a valuable purpose for acquisition, budget, and oversight offices alike.

Estimation. Aside from benchmarking, the other main use for historical trend lines is in estimating future releases. Before getting into specifics, it is important to first understand the method for how estimation is approached at QSM. The method is a top-down approach based on a proven production equation that is constantly adjusted and refreshed through the analysis of a very large set of actual, historical data. The software production equation is composed of four terms, namely: size, schedule, effort, and productivity. This equation is described below:

$$\text{Size} = \text{Productivity} \times \text{Time}^{4/3} \times \text{Effort}^{1/3}$$

The exponents attached to the time and effort variables differ significantly. The schedule variable, or time, is raised to the 4/3rd power, while the effort variable is raised to the 1/3rd power. This indicates that there is a diminishing return of schedule compression as staff are added to a software project. In extreme conditions, this indicates that consuming large amounts of effort does little to compress the project schedule. This equation can be rearranged algebraically to solve for any of the above variables.

Using QSM's proprietary SLIM® suite of tools (Software Lifecycle Management), it is possible to quickly create an estimate very early in the lifecycle, when little information is available. If the size and productivity assumptions are known (e.g., from similar past performances), then it is possible to calculate a schedule duration and the amount of necessary effort. Early sizing methods can be based upon analogous program efforts, requirements, or function points, and can be refined over time as more information becomes available. The size-adjusted productivity values can be taken from appropriate trend lines, based on historical performance. This very technique has been used to successfully estimate and model future releases of Defense ERP programs.

Refining the Cycle

One of the major advantages of having access to historical data is that it can be used at any point throughout the software development lifecycle. Figure 3 below describes this cycle:

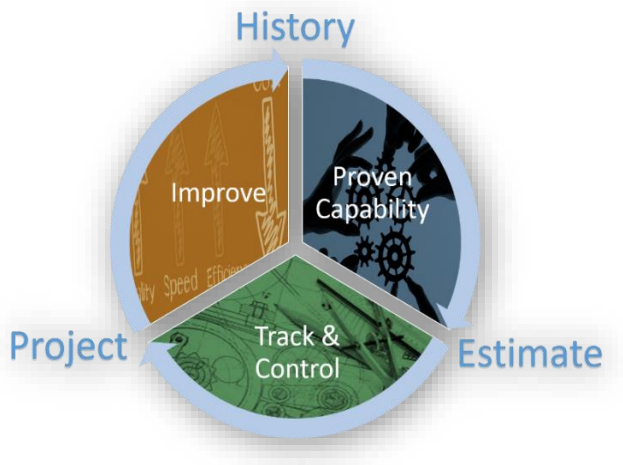


Figure 3. Historical software development data refinement lifecycle.

When a project is in its early phases, historical data can be leveraged to create an estimate based on the developer's demonstrated performance on other completed projects. Once accepted, this estimate can serve as the baseline plan for the program. When the project is underway, actual data can be collected at regular periodic intervals (e.g., monthly) and tracked against the estimate to assess and characterize actual performance. If the actual performance deviates from the estimated plan, it is possible to reforecast the schedule duration and necessary effort (i.e., staffing) values.

Upon completion, the final data is collected and added to the historical data repository. An updated set of historical trend lines is produced and can then be used for estimating future releases. Over time, there will be less discrepancy between the estimates and the actual outcomes (excluding the impact of external factors, such as changed or additional requirements, budget cuts, Congressionally-mandated changes, or other factors unique to the Defense environment), thus refining the cycle to improve processes across the entire software development lifecycle. However, more accurate estimation of what can be controlled certainly helps mitigate the overall impact of those factors that are beyond a Defense organization's control, and can be done at an early stage, when the cost of such adjustments is far less than later in the development process.

Summary

The British mathematician, Charles Babbage, stated that "errors using inadequate data are much less than those using no data at all." This quote really speaks to the advantage that can be found in collecting an organization's own data or in the use of data available in QSM-provided trend lines. As more programs are completed, an organization can continue to add to its historical database, refine their estimates, and improve processes throughout the entire software development lifecycle. With QSM's assistance, these process improvements, particularly as demonstrated within the DoD, can lead to significant savings in effort and schedule.

Estimating Cyber Protection

Victoriano Fuster III

Despite the best efforts of federal agencies and the near constant media coverage of threats, most government cybersecurity initiatives remain reactive. Once a threat is detected, agency teams typically scramble to identify the source of the intrusion and take necessary steps to mitigate its impact. The nature of the business can make planning and, therefore, budgeting a seemingly impossible task.

Unfortunately, federal IT security professionals' and program managers' hands are tied, thanks to limited budgets and time, with most of their time spent worrying about the costs and schedules involved in proactively creating a compelling cybersecurity program. Beyond that, they traditionally have not had the necessary tools to develop accurate estimates of what it will take to create these programs. In turn, this leaves them only able to make educated guesses, trapping them, in essence, in a reactive mode.

Agency project managers need to be able to build and develop their cybersecurity systems just as they would a software project. They need accurate planning and estimation that will allow them to consider timeframes, appropriate staff, potential costs, quality, risk, and other key factors.

Applying a Proven Parametric Estimation Approach to Cybersecurity

There are proven benefits provided by parametric estimation tools, which use parametric estimation algorithms and are based on detailed analytics, historical data, and methodologies for IT project estimation. These tools and methods are particularly beneficial for large-scale, complex projects that require a great amount of estimation, tracking, and analysis to achieve the right mixture of staff and effort for on-time, on-budget delivery.

Although historically such parametric estimation was designed for estimating the development and maintenance of software projects, it can do much more. In addition to being beneficial for planning complex IT infrastructure projects, the approach can also be effectively used to accurately estimate the costs and schedules associated with the development of large-scale cybersecurity initiatives. Indeed, parametric analysis methodology can easily be adapted to size the

actual work effort required not only to develop overall federal information systems, but also the specific cybersecurity components of those systems.

Using the NIST Blueprint

One suggestion is to start with an already-existing government blueprint – NIST Special Publication 800-53. This National Institute of Standards and Technology (NIST) standard is the most prescriptive guidance of security controls required by federal agencies. It provides detailed recommendations for the various low-, medium-, and high-impact security controls that government agencies need to implement to achieve minimal security postures.

Within the parametric estimation toolset, we can create a work breakdown structure (WBS) that maps to the security controls category requirements contained within NIST Special Publication 800-53. This allows US Government agencies to gain an accurate representation of the costs and effort required to implement many of these categories. Through such an approach and balance of fixed costs and parametric estimates of systems development, agency IT program and security managers can receive easy-to-understand graphic depictions of the estimated size, cost, schedule, and other cost factors, of their information systems. Historical comparison of information system project sizing furthers enables insights into what is necessary to accomplish an agency's cybersecurity goals. Based on the analysis, organizations gain data-driven and more realistic estimates of how much it will cost to develop the mandated security controls outlined within each category of the NIST publication.

This solves multiple challenges. First, it provides program managers with the knowledge and insight necessary to build a cybersecurity posture based on the requirements outlined in the NIST guidance. More importantly, it allows them to do so based on a quantitative analysis of trends and evidence regarding the specific historic and analogous costs associated with these activities. No one will ever be able to predict the costs incurred by reacting to unexpected intrusions. However, teams will be able to perform upfront cost analysis for adherence to the minimal security standards called for by NIST – all through the use of a proven parametric estimation methodology.

Good Planning – Not Development Methodology – Is Key to Successful Project Delivery

Douglas T. Putnam

IT development teams have long been searching for a magic formula of success in software development. How can teams find the greatest efficiency and the least cost, without sacrificing quality?

Teams have tried a myriad of methodologies to achieve these goals. Twenty years ago, developers looked to waterfall development as the answer. Waterfall then gave rise to object-oriented incremental or spiral, Rational Unified Development (RUP) practices.

Today, it's all about agile development. Companies are investing lots of capital to develop agile methodologies and committing significant resources to train employees to work within agile frameworks. Despite this investment in agile methodologies, many projects still fail, clients are unsatisfied, and IT departments often miss deadlines. Why?

The method doesn't matter: It's all about planning and how teams use resources.

When projects get behind schedule, team leaders almost automatically think that adding more staff will help. After all, many hands make light work, right? The more staff a team has working on a project, the faster and shorter development time needed.

The old adage doesn't necessarily apply in software development. In fact, the opposite effect can occur: adding more people to a project can make more work or even slow things down further. The additional person-hours may give the team a short-term boost, but over time, the team will have to manage higher costs and more connection points—each of which creates an opportunity for a mistake or defect, bringing additional risk to the project.

What's more, how teams allocate resources is just as important, if not more, to successful projects as which development method teams use. While methods have changed, the allocation of resources has remained paramount over the last two decades. This finding was underscored in QSM®'s latest analysis, which leveraged data from the most recent QSM Software Project Database update. The update includes new insights into agile development processes and the staffing models that agile teams are employing for their projects.

Table 1 below provides a glimpse into how adding staff affects a project's outcome. It compares 390 applications of the same size featuring both 10,000 and 20,000 lines of newly developed code, with a significant portion using agile methods and tools. One sample uses an average of less than four people; the other, nine people or more. While the additional staff reduced the schedule by approximately 30 percent, the project cost actually increased by 350 percent. The additional staff also created 500 percent more defects that had to be fixed during testing.

Table 1. Effects of adding staff on a project's outcome.

Staffing Scenario	Size IU's	Schedule (months)	Effort (hours)	Cost @ 110 per hour	Peak Staffing	Defect found in Testing
Average staff is greater that 9 people	10,000	3.85	8,790	$966,900	17.0	122
Average staff is less than 4 people	10,000	5.33	1,920	$211,200	2.7	18
Difference Absolute Value	None	1.48	6,870	$755,700	14.3	104
Percentage change	None	-27.7%	357.0%	357.0%	529.0%	577.0%
Average staff is greater that 9 people	20,000	4.71	11,230	$1,235,300	18.0	153
Average staff is less than 4 people	20,000	6.80	2,670	$293,700	3.0	29
Difference Absolute Value	None	2.09	8,560	$941,600	15.0	124
Percentage change	None	-30.7%	321.0%	321.0%	500.0%	427.0%

In Figure 1 below, the lines represent the average behavior for size developed vs. schedule, effort, defect, and average staffing. Notice the large variation in effort and defects and small variations in schedule.

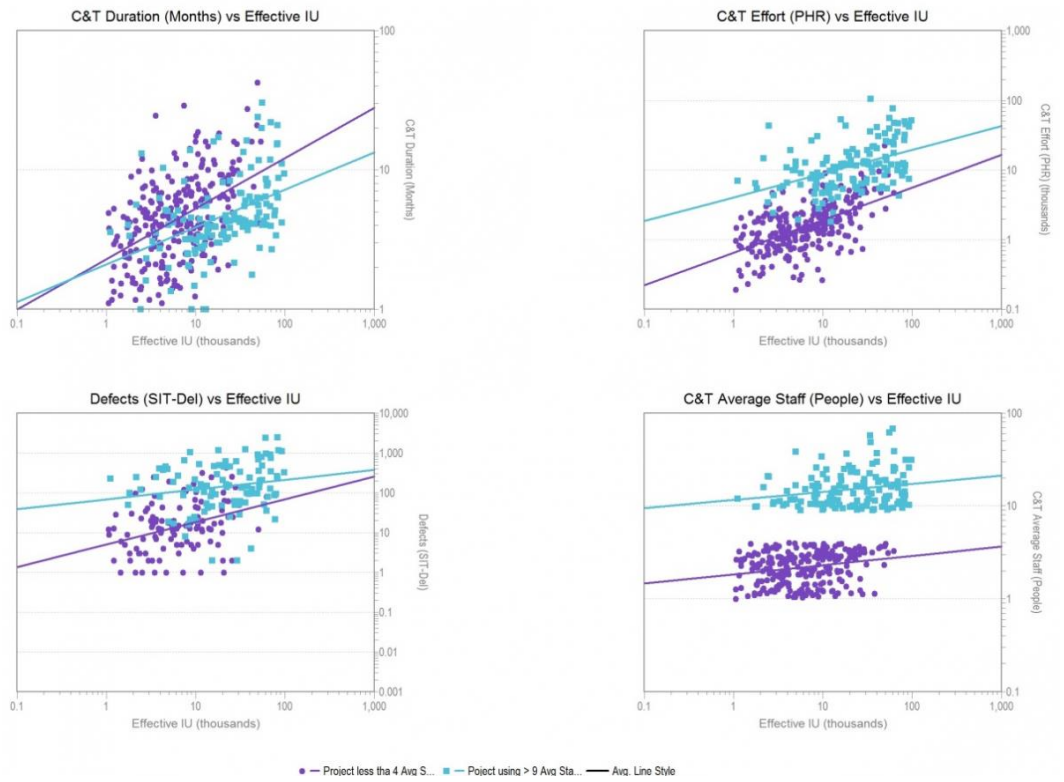


Figure 1. Time-effort-reliability trade-off between average staff of four people or less versus average staff of nine people or more.

Over the past 15 years, QSM has performed this same study in five-year increments and has found the same results -- staffing decisions have more of an impact on project success than any development methodology.

Instead of relying on a particular development method, project managers should estimate early and use predictive analysis to staff their projects. Predictive analysis and early estimation can help C-suite executives more accurately plan for staffing as well as set projects and developers up for success. These techniques provide an accurate sense of how many people should be assigned to a project. Additionally, executives who use them gain actionable insights that help with course correction throughout a given project's lifecycle.

Individual projects may differ, but all projects can benefit from a step-by-step process that uses predictive analysis and early estimation to more accurately account for staffing needs.

1. Gather all current project data in one place.

It's important to know the status of current resources before beginning a new project. Creating a central place that allows team leaders to see all project data at-a-glance – including start and end dates, resources currently deployed, and so forth – can provide a snapshot of how many resources are currently available to work on a given project and where that project fits into an organization's overall development cycle.

Software lifecycle management solutions provide the tools through which team leaders can monitor projects that are in various stages of development. These tools store data about each project an organization completes and allow users to generate subsequently more accurate forecasts with each project entered and tracked.

2. Examine historical data.

Once team leaders have current project data in place, they should run a forecasting analysis to see if they have the right number of resources available to complete a project in the timeframe proposed. A forecasting analysis is based on historical data from past projects that were similar in scope and size. Data from these past projects give a good idea of how many resources may be needed to complete the project within the budget and time allotted.

Unfortunately, deadlines and client expectations don't always match. It is up to team members to use available historical data and make decisions whether proposed deadlines are achievable. Members who use historical data to suggest extended delivery dates (to meet goals or save money) do so from a position of strength – they have the numbers to back up their projections.

3. Compare project data to industry trends.

Teams can compare forecasts to industry-wide trends to see if their numbers are tracking similarly. A software lifecycle management tool can help by compiling trend data, or teams can gather it themselves. Either way, they should graph trend data against their own data points to see how their staffing and resource levels compare with other, similar projects.

Consider the following example (Figure 2 below): PI represents a project's "Productivity Index," a measurement taken from QSM founder Larry Putnam's Putnam Model, which has been the standard in software estimation for nearly 40 years. The red dots represent a company's current resource level, while the blue lines indicate industry averages.

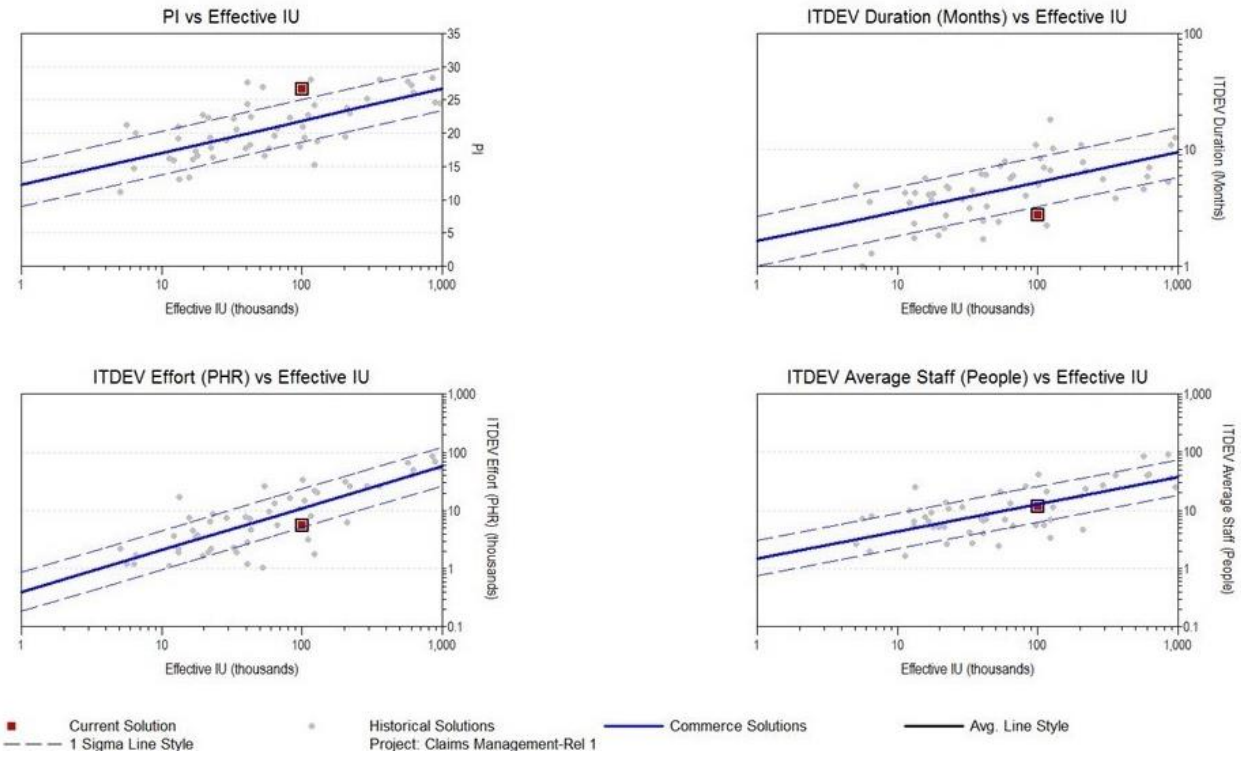


Figure 2. Example "Compare to History;" note that the productivity assumption is much higher than normal, resulting in a shorter than usual schedule and effort – a recipe for disaster at delivery time.

A visual representation makes it easier to see where adjustments need to be made to bring a project more in line with the rest of the industry. Tech teams can tell if they are falling behind the curve, ahead of it, or right where they need to be.

4. Determine a project's impact on an organization's overall budget.

No matter which project they work on, teams should be able to see the potential impact an organization's projects have on one another as well as on the overall development budget and resources. A dashboard can provide an at-a-glance view of all projects and the data relevant to each, such as scheduling, cost, and resources. Team leaders and C-suite executives will appreciate this overall view, as it will help them track when specific projects may be deviating from estimates and threatening to throw an entire IT budget out of whack.

Figure 3 shows another example of an at-a-glance view. The red dots represent given projects. The goal is to get them within the green rectangle – the "sweet spot."

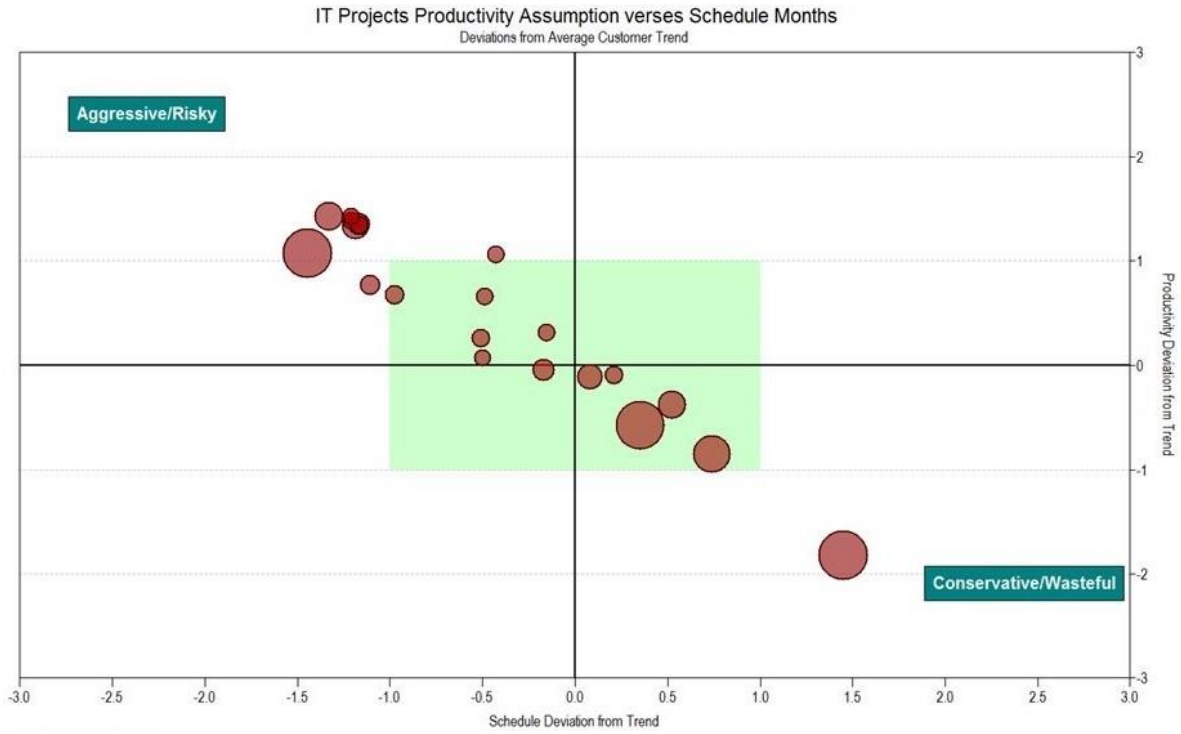


Figure 3. IT projects' productivity assumption versus schedule months.

Having this information at the ready will help teams identify projects that may be in danger of becoming risky or wasteful. Once again, predictive data helps teams course correct and get back in the green zone.

5. Use available resources fully.

Perfecting the way staff are allocated increases efficiency and minimizes the risk of project failure. In this final example, shown below at Figure 4, staffing and cost charts are used to accurately plot staffing resources across the entire project portfolio. With a view of all of an organization's projects, executives can adjust parameters on risky projects and eliminate overstaffing (which reduces cost and improves quality) from others. Doing so can then quantify the benefits of the optimization process and match demand with current capacity, which ultimately leads to a realistic portfolio plan.

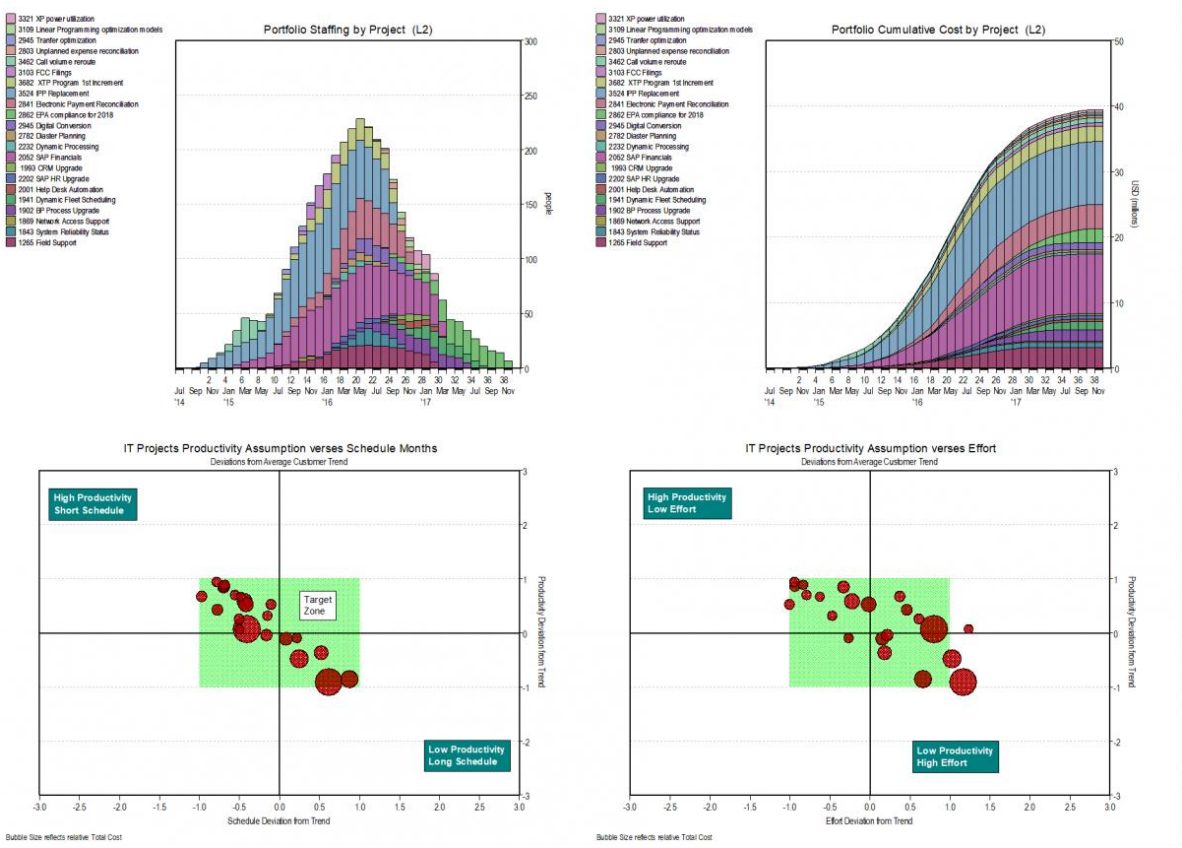


Figure 4. Portfolio resource and risk quadrant view of example project.

With an entire portfolio of staffing needs captured at a glance, executives can more accurately determine how to allocate sufficient resources to individual projects. Will current resources be enough? Will they need to move team members around to complete a given project on time and within budget? Are new personnel needed? If so, how many should be hired to get to "peak staff?"

Again, methodology clearly doesn't matter here. Knowing which resources are available and optimizing them makes answering those and other staffing questions that much easier. Teams can then complete projects that deliver on clients' expectations in a realistic timeframe and cost-effective manner – and thus be one step closer to that "magic formula of success."

Common Ground Through PPM

Lawrence Putnam, Jr.

*This article originally appeared in the February 13, 2017, edition of the **Projects at Work** online journal, and is reprinted here with permission.*

What do an IT project manager and a top-ranked business executive have in common? Aside from the occasional fondness for caffeine (all those late work nights!), perhaps more than one might originally suspect. For while both IT and senior management have usually operated in their own separate spheres, both groups are tasked with establishing goals that will help move their businesses forward.

Traditionally, these two groups go about achieving these goals in different yet somewhat similar ways. Executives engage in strategic portfolio planning, the business process by which organizations determine the set of new product development initiatives they will fund (and which ones they will not) to help achieve their overall business objectives. Meanwhile, IT project managers embark on IT portfolio management, which focuses on the investments, projects, and activities of IT departments.

Common ground through collaboration

Wouldn't it make more sense — and be more efficient — to combine those two practices? After all, both sides are committed to seeing their organizations succeed. And in today's business climate, success is often the result of collaboration among traditionally siloed teams. In IT, this most commonly manifests itself through DevOps, where operations and development teams pool their resources to accelerate application development.

But let's extend that philosophy to connect IT with senior business leadership. IT is increasingly becoming a significant factor in the success of today's enterprises, and we are seeing more IT administrators have a seat at the executive management table. If a business is to succeed, senior executives need input from their organizations' technology teams to truly gauge the feasibility of developing capabilities that are under consideration. At the same time,

IT needs input and insight into what those capabilities are, so they can produce the appropriate solutions to bring them to fruition and satisfy the needs of their business leader colleagues.

For this to occur, business and IT teams must break down the barriers that may exist between their groups — and that's where project portfolio planning comes into play. Project portfolio planning creates a platform where both business and technology teams can communicate with each other and learn about their needs and expectations, so that each can properly plan the development of solutions that will help their organizations. It also lays the groundwork for senior business leaders and IT executives to be able to prioritize upcoming projects and determine whether or not the budget and resources exist to make them happen.

The project portfolio planning process

The main goal behind project portfolio planning is to determine the maximum capabilities that can be delivered within the confines of budgets, resources, and time. Prioritization of projects is a large part of the process, as is alignment of those projects with strategic business goals. Business and IT teams work together to build business cases and return on investment scenarios that justify the effort and cost involved in project development. This research and discussion help determine which projects ultimately get green lights, while others get delayed or shelved.

The first phase of project portfolio planning should be the prioritization phase, which requires the development of detailed scope estimates for all potential projects. In this phase, IT teams must acquire as much insight as possible into the expectations of their business executives and understand what type of functionality will be necessary to realize those expectations. This is critically important, as it helps teams gain a clear sense of exactly what they wish to accomplish for each project in the portfolio. Ultimately, this phase helps determine the feasibility of each of the projects-in-waiting, thereby allowing teams to prioritize the ones they want to devote time and resources to building.

Unfortunately, many teams struggle with this initial step. Some may feel that it is too early in the process to be able to develop an effective and detailed scope estimate. Others may take a misinformed (though well intentioned) shot at it by applying numbers that have no real basis in reality. Still others may disregard estimating and scoping entirely based on the rationale that they practice lean or agile development methods that, they believe, do not require estimation practices.

None of these approaches are effective. In fact, organizations that do not take the time to engage in the scoping and estimation process at the outset are more than likely doomed to not meet their development and business goals (and yes, this applies to organizations dependent on agile development, too). The process itself does not need to be extremely detailed, especially early on, but there needs to be something in place that helps both business and IT teams gain some sort of idea as to the viability of the projects that are on their drawing boards. If not, teams risk spending time and money on projects that are, at best, non-essential or, at worst, may never even get off the ground.

Indeed, scoping out projects helps teams gain a better understanding of what is feasible and what is not, given budget, resources, and even historical comparisons to the cost and work associated with past projects. The information gleaned from that initial scope should be combined with the answers to several questions -- Which projects absolutely need to be done in the short term? Which can be pushed back? What's most important to our overall business success? These answers can help project teams create accurate and worthwhile IT to-do lists.

Finally, it is important for IT teams to make sure they have the resources necessary to actually complete the projects. Sadly, that is not always the case. Individuals are often pulled in different directions due to budget and time constraints and may be unavailable to work on new projects. If that is the case, it is decision-making time yet again. Managers will need to decide if they want to bump projects until later in the planned development cycle or extend their timelines for completion. Of course, they could always decide to reallocate or add more resources, but that may require hiring new employees (which could impact corporate budgets and financials) or taking resources away from another project (which could impact the completion of that other project).

A challenging (and rewarding) process

Project portfolio management can be extraordinarily challenging. In a sense, the process is something like the business equivalent of physical building blocks. All of the pieces (business and IT) must work together to form a complete whole. Certain pieces must form the basis of the structure (estimation and prioritization), and even after it's begun, there is always going to be a constant and delicate balance between making sure the pieces work properly (resources) so that the whole project does not topple over before it is completed.

And yet, project portfolio planning can also be extremely rewarding. It can foster a closer and better working relationship between IT managers and business leaders. It can also provide them with a roadmap toward working together as a single team to create and deliver viable projects that will bring true value to their organizations.

References

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

Is Software Estimation Needed When the Cost and Schedule Are Fixed?

Keith Ciocco

In many agile environments, the budget, team size, and schedule are fixed based on an organization's predetermined targets for sprints or iterations. This leads many project managers to question if software estimation is even necessary. The problem is, without a reliable size estimate, the amount of functionality promised within the time and money constraints could be difficult to achieve, ultimately causing the product delivery to be short on features, or late and over budget.

This is where scope-level estimation tools come into play. They can help evaluate whether targets are reasonable and, even if the schedule and budget are both set in stone, they can help figure out how much work can be delivered. This type of analysis helps set customer expectations and provides data-driven leverage for negotiations. The best estimation tools leverage empirically-based models, industry analytics, and historical data. They can even be used before iteration level planning takes place, ensuring that the overall goals are reasonable before detailed plans are developed. This article will use the SLIM-Estimate® tool as an example to demonstrate the "Time Boxed" method.

Figures 1 through 3 below show an estimate generated from "Time Boxed," where the product manager was able to input a predetermined time, a productivity measure (PI), and a team size, to see how many story points could be completed within the set constraints. The analysis also includes a "sanity check" of the estimate, comparing it to an agile industry trend from the QSM® industry database and their own agile historical data.

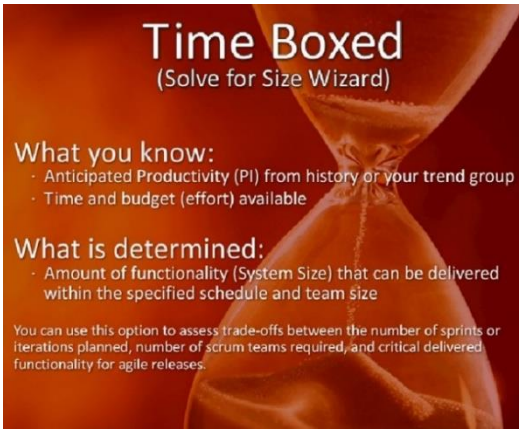


Figure 1. Time Boxed, Fixed Team "Solve for Size" wizard screen.

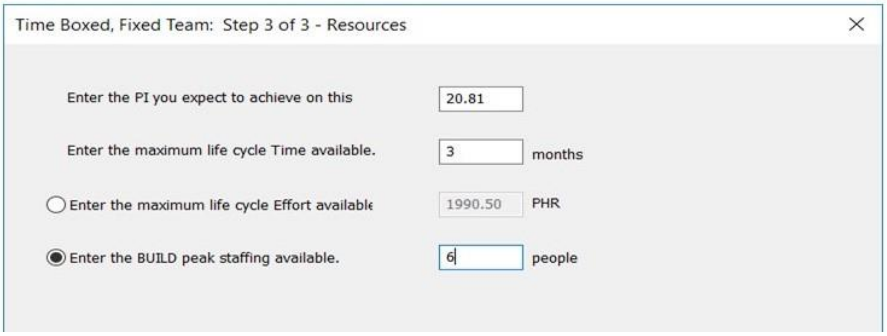


Figure 2. Time Boxed, Fixed Team data entry screen.

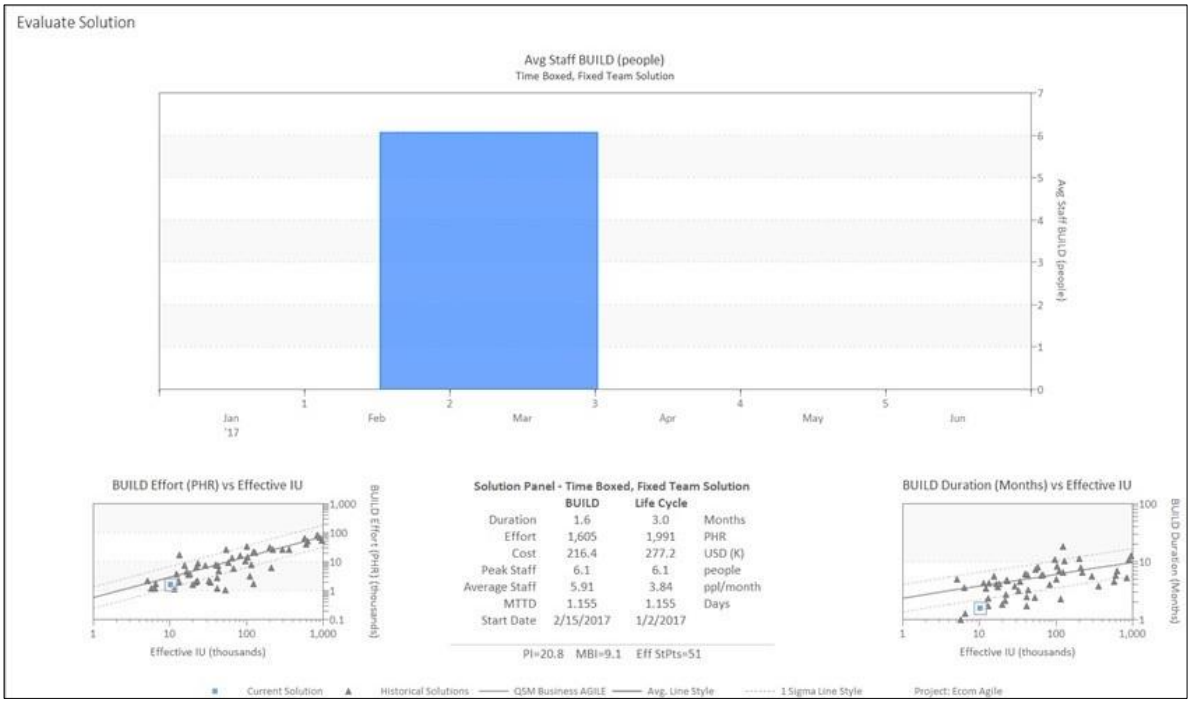


Figure 3. Time Boxes, Fixed Team evaluation of solution screen.

In addition to making sure cost and schedule targets are reasonable, an estimate can also be generated showing how reliable the system will be at delivery. The defect information at Figure 4 below was generated from the same "time boxed" method shown above.

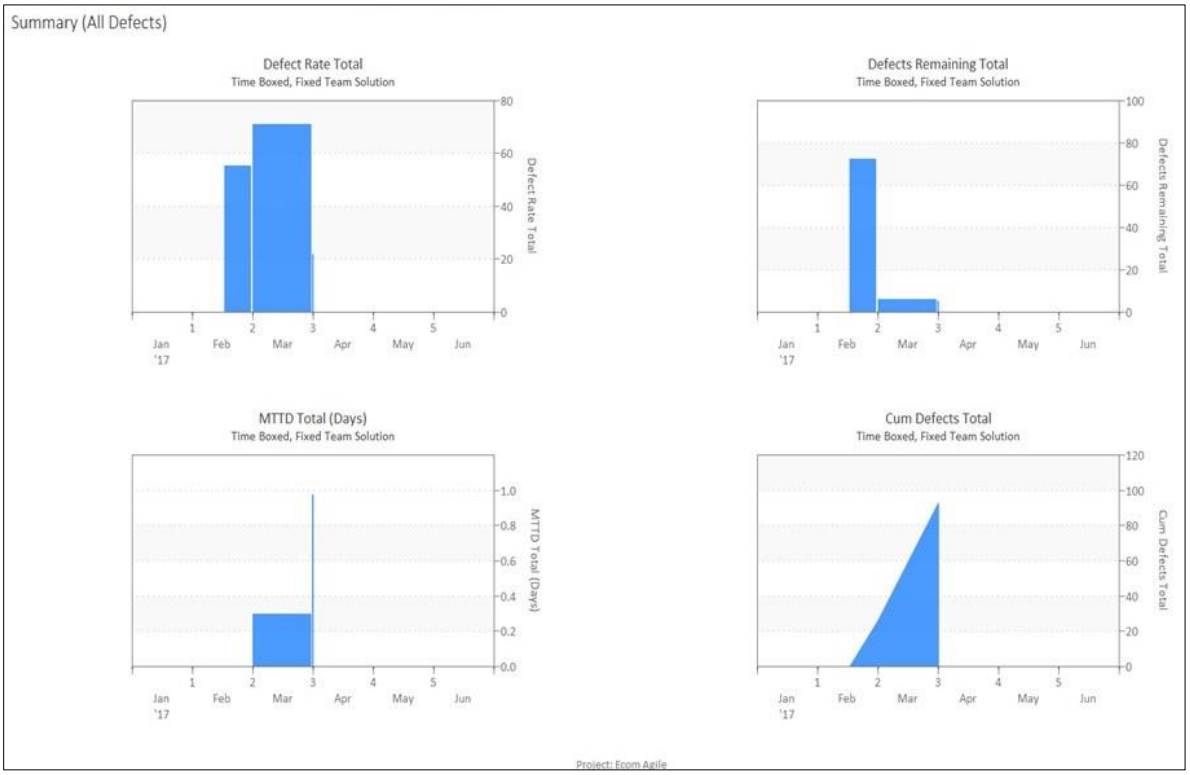


Figure 4. Defect information generated from Time Boxed solution shown at Figures 1 through 3.

Overall, leveraging this type of a data-driven approach provides the ability to see the "big picture" early in the planning process so that unrealistic schedules and budgets can be avoided. Most importantly, when those initial schedules and budgets are fixed, the amount of work can be negotiated within the set parameters with a focus on quality and the ability to avoid significant frustration later.

Role of Top-Down Estimating in SAFe

Laura Zuber

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

Introduction

The jury is still out concerning the value of project estimation for organizations employing agile methods. Opinions vary by stakeholder's position and perspective – what they need to know and when. QSM agrees that individual teams, or small organizations with just a few teams, may not realize great benefits, because capacity is fairly well known. The focus is on prioritizing the backlogs. But as organizations grow, or large enterprises adopt agile methods, the ability to scale and realize the benefits of small teams and frequent deliveries becomes quite challenging. This paper describes the benefits of using scope-based, top-down estimation with the Scaled Agile Framework (SAFe) for Lean Software and System Engineering.

Vision, Value, and Velocity

These are all terms used in the SAFe—product **vision**, business **value**, and development **velocity**. An enterprise's ability to align these key contributors to successful product development lies in the definition of these terms:

Vision: Product vision, an image, painted with words and pictures, preferably, of both the features and benefits that will inspire the teams at every level, keeping everyone working toward the same goal. The details are born in the Product Roadmap

Value: This is a measure of the benefits of achieving the vision. Consider the Tesla automobile. A couple of benefits are that it is energy efficient and ecology-friendly. The value of a Tesla as an economic measure of the benefits may be CO2 emissions in parts per million or gallons of gasoline saved.

Velocity: Simply development speed. Actually, it is not simple at all. It is more appropriately described as efficiency and what affects it, both positively and negatively.

Figure 1. Vision graphic to demonstrate scaling agile development.

This vision graphic (Figure 1) is particularly germane to scaling agile development, because it shows the power of vision to influence – arrows pointing out show vision drives product development. At the same time, the vision is itself influenced, changed, by everything that is discovered during the development process – arrows pointing in. The product takes shape as development progresses, for the better, we hope. This is a major principle of agile.

The challenge for large organizations using SAFe is to keep these three "drivers" – vision, value, and velocity – aligned and relevant, tied together, as large epics are broken down into features and stories, smaller chunks of the vision at different levels of the framework, and the actual realized value is aggregated back up into a consumable solution.

What is SAFe?

The Scaled Agile Framework (SAFe) is a freely revealed knowledge base of integrated, proven patterns for enterprise Lean-Agile development.* It synchronizes alignment, collaboration, and delivery for large numbers of teams. SAFe embraces three processes:

- Kanban– Quality & Defects
- Lean –Fixes Scope
- Agile – Fixes Time & Resources

SAFe v4.0 addition of Value Stream from Lean. You can access detailed descriptions, videos, and abstracts at www.scaledagileframework.com. All companies have value streams.

We have included some images and descriptions from the SAFe abstracts and website to describe patterns that tie to top-down, scope-based estimation. Agile methods and practices at the Team level, the lower level on the diagram at Figure 2, are well understood. Scaling agile is quite challenging.

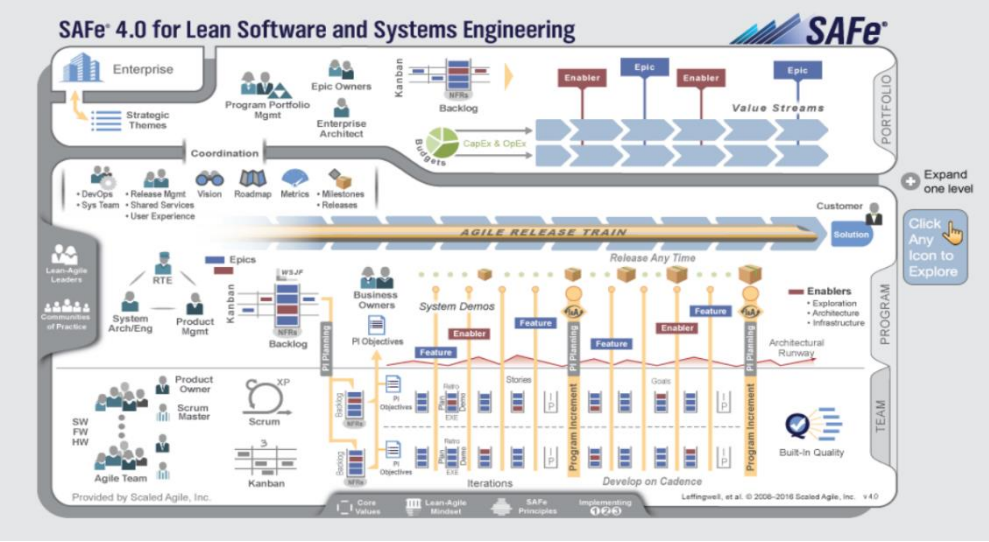


Figure 2. SAFe 4.0 for Lean Software and Systems Engineering.

SAFe Principles Supported by Top-down Estimation.

We believe that estimation supports the following key SAFe principles:

- **Size delivered functionality (Epics, Features, Stories)** - Here, size is a verb, meaning the size of delivered functionality needs to be estimated. The required time and effort are a function of the size of the product or system. Early in development, Epics are defined, and these are broken out into Features, Capabilities, and Stories – using Story Points as the size unit.
- **Take a Systems View** – A system can evolve no faster than its slowest integration point. We believe that this notion applies to not only the system itself, but that a development process system view is critical. That is exactly what top-down estimation provides: high-level projections of product delivery.
- **Base forecasts and commitments on known capabilities and resource availability** - SAFe acknowledges the need to forecast and commit to no more than what we know we are capable of (from history), and that not only do we need to anticipate and plan for the number of resources required but ensure that the teams have the right skills mix.
- **Quality matters** – You cannot scale poorly written code. Estimates must include a forecast of the product quality at every level of the framework. Being able to predict expected defects and track their removal is the true measure of "done done."
- **Working software is the primary measure of progress** - Measure the actual size delivered at the end of each sprint and Program Increment. This, plus actual velocity, is the basis of each new forecast.
- **Consider all the steps – definition, analysis, validation and delivery** – All of the work must be estimated and planned, not just development and test activities. This tells us what skills mix is needed, too, which will vary by project type. What specialists are needed, and when?
- **Roadmap forecasting requires estimating** - We want to explore the range of potential outcomes, quickly and confidently, by calculating the most likely, best, and worst-case scenarios, along with the probability of each outcome. This helps us deal with the reality that value and vision require tradeoffs. Estimate should:
    - Be fast, efficient, and as reasonably accurate as possible
    - Support "what-if" analysis of various implementation scenarios
- **Program/Portfolio Management Office (PMO) still has to do their thing** - Vision starts with an idea. The schedule and budget needed to realize the vision must be determined before we have a defined product or backlog. The results of these high-level estimates feed the strategy for how the enterprise can achieve the most important objectives.

The following sections highlight a few of these principles.

The Business Needs to Forecast

We work with many clients who have an annual project and program planning cycle. The quote here reminds us that the business has to make commitments — Budgetary commitments, Time commitments, and Staffing commitments. QSM® has long promoted the philosophy that estimates and commitments are two different things — calculate multiple **estimates**, and **commit** to the one that best achieves the project or release goals within the constraints or known risks.

> *"Some initiatives take years to develop, and some degree of commitment must be made to Customers, Suppliers, and Partners."*

SAFe enhances enterprise adaptability, providing faster response to changing market opportunities. Yet, the enterprise, its partners, and customers need to plan to get some sense of the future. The ability to perform effective, agile forecasting is a key economic strategy (Figure 3). The roadmaps capture strategic intent in the form of forecasted deliverables.

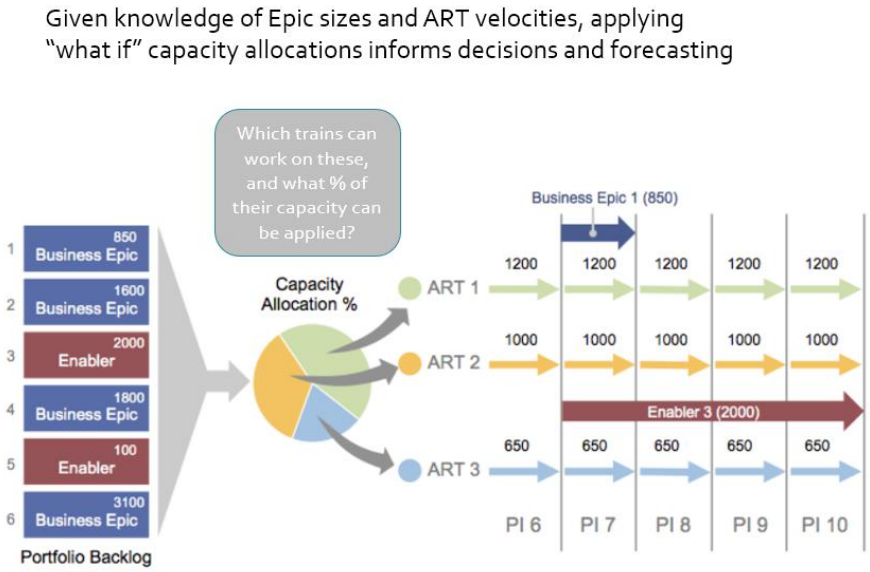


Figure 3. Agile forecasting and Agile Release Trains (ARTs).

In SAFe, the Portfolio Backlog items are allocated to Agile Release Trains (ARTs). Although a key objective is to keep well-established teams together, allowing them to accept or pull work from the Portfolio Backlog, the ability to forecast alternative strategies for each portfolio item or initiative is important. This allows the business to see the impact of each, graphically, and provides another type of Vision – not of the product, but the strategy; a visual roadmap of sorts. An example of this process is presented later in this article.

What is "Consumable Value?"

Agile development teams produce potentially releasable solutions every iteration (Figure 4). Organizations developing large system are demonstrating viable portions of the **system** every

iteration. Solution increments are demoed every Program Increment to incorporate assets across multiple ARTs. Fully functioning releases can be produced "off cadence."

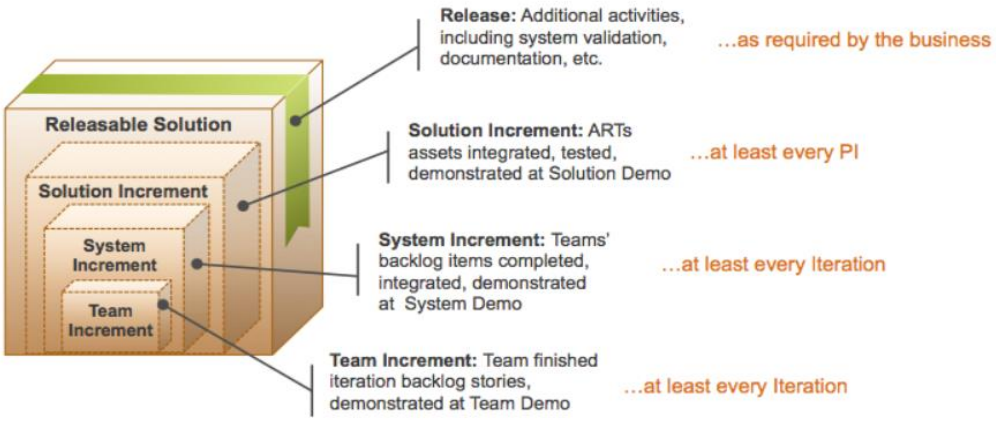


Figure 4. Building a releasable solution.

The Portfolio and Program managers are responsible for knowing when the product is ready to be consumed from the view of the customer or the market. Is there a viable product, and are the customers ready to consume the value? This is determined by the customer and the type of product, of course. Companies like Facebook think two-week increments are too slow – their value is consumable immediately! There are a lot of product, enterprise, and customer characteristics that make this so, and the business needs to know what determines the timing of consumable value for its particular enterprise.

Whether the group of features created to-date completes the vision, plus the time and effort required for the rollout, drives decisions regarding Release Planning and how each release contributes to the enterprise strategy. The ability to estimate several release scenarios, and their associated risks and tradeoffs, enables the enterprise to make the best strategic decisions.

Releasing Includes Other Activities

We need a method for estimating the time and effort required to deliver a fully functioning release. This is not complicated for small initiatives developed by one or two teams. It certainly must be accounted for at the Program and Value Stream levels. Here are just some examples of activities that may need to be considered for the release estimate:

- System validation
  - User acceptance testing
  - Final NFR testing
  - Integration testing with other systems
  - Regulatory standards and requirements
- Documentation:
  - Release communications
  - End user documentation
  - Bill of materials

- Training support personnel
- Installation/deployment instructions
- Legal, regulatory, other

Kanban System for Epics. The portfolio Kanban system (Figure 5) describes a number of stages that an epic passes through on the way to implementation (or rejection), and the collaboration that is required for each stage:

1. **Funnel** – This is the capture state, where all new "big" ideas are welcome.
2. **Review** – In this stage the preliminary estimates of opportunity, effort, and cost of delay are established. At QSM, we call this a Feasibility estimate. It can be computed very early in the idea stage to flush out unrealistic expectations, based on historical data about the industry and the organization's capabilities.
3. **Analysis** – This is where the thorough work is done to establish viability, measurable benefit, development and deployment impact, and potential availability of resources. A lightweight business case is developed. The epic is approved or rejected at the end of this state.
4. **Portfolio Backlog** – Epics that have made it through the portfolio Kanban with "go" approval wait in the Portfolio Backlog until capacity is available. What is the true capacity, and how is the availability communicated?
5. **Implementation** – When capacity becomes available, epics are transitioned to the relevant Program and Value Stream Kanbans. Teams begin implementing at Program Increment Planning boundaries.
6. **Done** – The epic is done when it has met its success criteria.

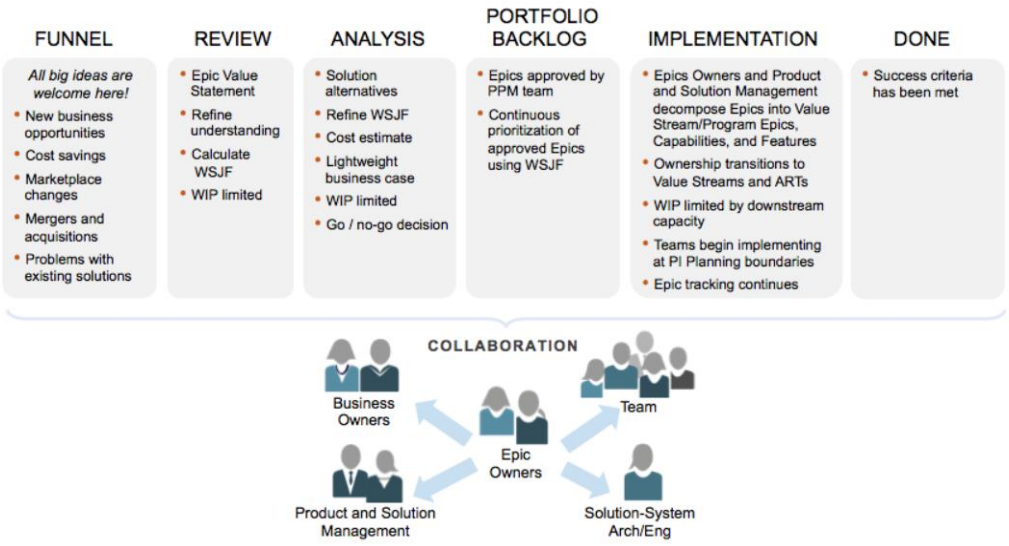


Figure 5. Portfolio Kanban system and typical collaborators.

Portfolio Backlog holds epics approved for implementation (Figure 6). These epics have made it through the portfolio Kanban with "go" approval. It serves as a low-cost holding pattern for upcoming implementation work. Sizing estimates are in story points.

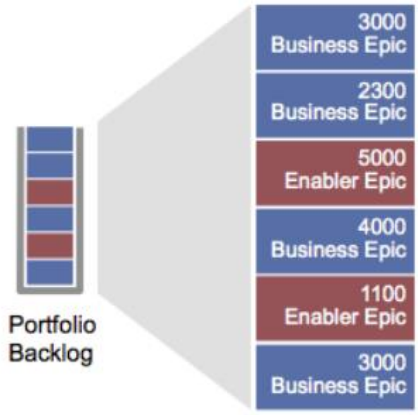


Figure 6. Portfolio backlog.

The message here is that the Program Portfolio Management team has to have timely and accurate data about ART capacity and availability to release epics into the Value Stream [and predict when the consumable value will be complete] and maintain business-as-usual support activities.

> *"Program Portfolio Management requires an understanding of the productive capacity of each ART, the velocity of each, and the availability of each for new developments and business-as-usual support activities."*

Our experience is that there is a flurry of activities with release implementation. Depending upon how well defined and visible release activities are for the enterprise, and if the enterprise is new to SAFe, epic estimates may not be accurately accounting for these activities.

The Roadmap Guides the Delivery of Features

Once a Portfolio item receives "go" approval, Value Stream and Program Management Teams need to know the current and near future capacity of their Teams, not only to anticipate Epic, Capability, and Feature allocation options, but potential viable product release dates as well. Program Increment Planning is a major SAFe construct (Figure 7). They are 8- to 12-week time blocks that facilitate team cadence and synchronization. The history of ART "velocity" and readiness to pull items from the backlog are key inputs and outputs of Program Increment estimates and forecasts.

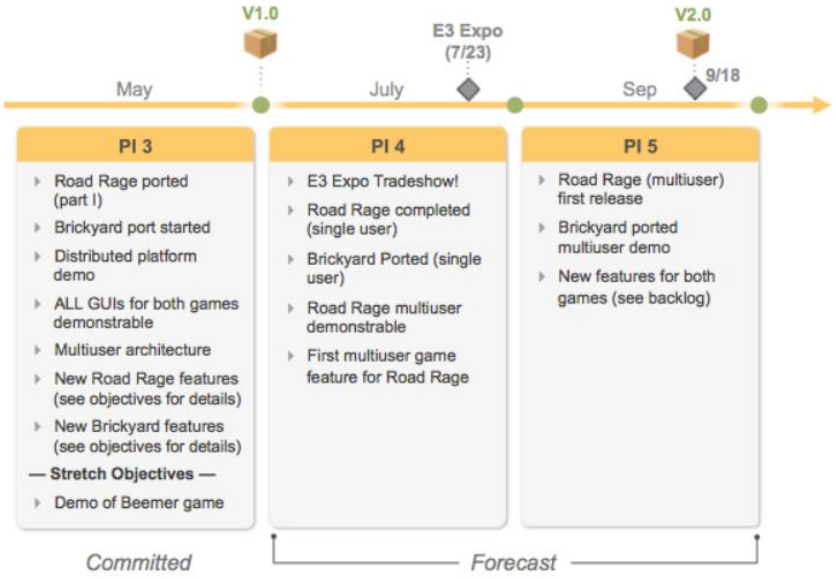


Figure 7. An example PI Roadmap for a gaming company.

Top-Down Estimate Example

We reviewed these SAFe principles to highlight several matching concepts and best practices defined in QSM's Software Lifecycle Management (SLIM®) methodology.

The best way to demonstrate this is with an example. Recall that we proposed that successful, large-scale product development requires an alignment of Vision, Value, and Velocity. We have defined two ways this can be achieved:

- **Visualize the Portfolio**. It is not enough to estimate individual Portfolio Backlog Items. We need to look at all the "big ideas," and the aggregated cost, time, and resource requirements at the Portfolio level. This is the data that supports risk assessment.
  - Perform early high-level estimates of Portfolio Backlog Items to support Kanban process
  - Assess risk areas by weighing alternative schedule and cost tradeoffs.
- **Visualize Velocity and Value Creation**. Create estimates for various combinations of ART engagement, based on known capabilities and the complexity of the interaction of people and systems.
  - Model alternative scenarios for release of epics into the Value Stream
  - Account for the reality of the rate at which work becomes available
  - Use historical productivity measures that incorporate the non-linear behavior of software development

IT Budgeting Analysis Example

QSM Co-CEO, Doug Putnam, published his work on IT budget planning (http://www.qsm.com/articles/take-risk-out-it-budgeting-linkedin). Early budget estimates, regardless of how they are derived, are typically buckets of labor hours for undifferentiated roles, with little to no tie to the schedule. We will use that as one example, using the SLIM tool suite, to demonstrate IT budgeting analysis.

QSM's SLIM tool suite allows you to visualize core portfolio planning and execution data, including cost, effort, staffing, scope or business value, and schedule. The Gantt chart below shows the proposed schedule of the portfolio items. Each bar represents an initiative that constitutes a viable, consumable release. These releases are of varying duration and have proposed start dates tied to the business strategy.

Each initiative is estimated individually and aggregated at various levels of hierarchy. Perhaps you have Program and Value Stream levels. A vision of the proposed timing of strategic initiatives often, by itself, provides great insight, and prompts discussions about Portfolio Backlog prioritization. Assigned priorities must consider not only the business value, or whatever measure your organization uses to indicate backlog item contribution to strategic goals, but also whether the Teams and Agile Release Trains have the capacity to take on the work at the proposed time.

In Doug's paper, he shows the "as is" portfolio estimate (shown below at Figure 8), or the desired outcome, to phrase it differently. The question is, can the enterprise achieve the implied targets or goals?

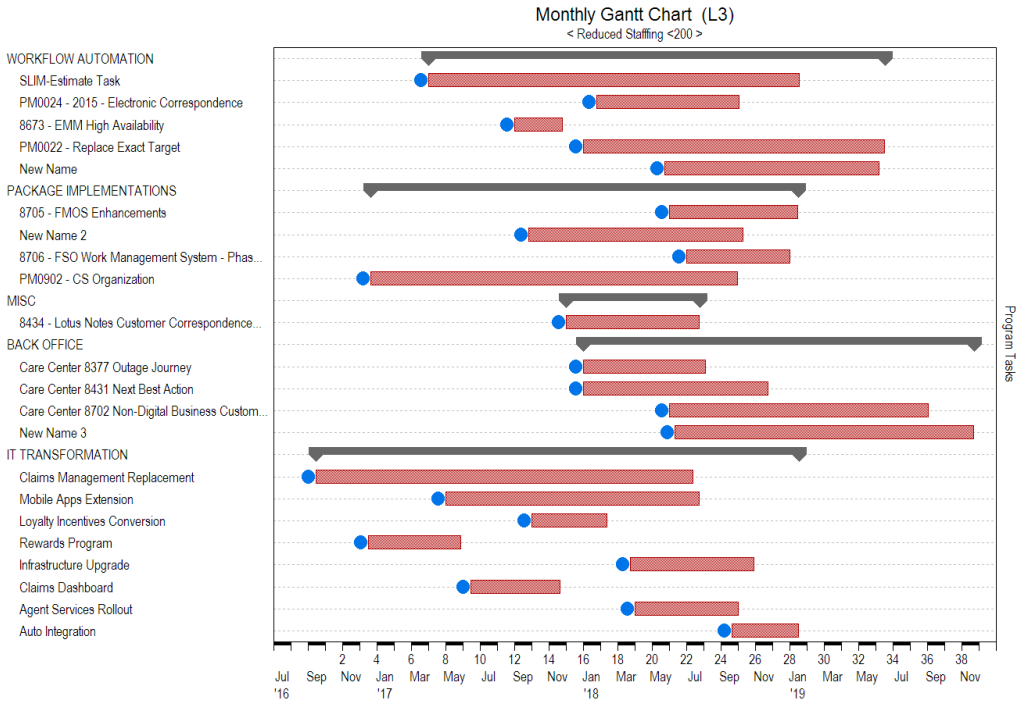


Figure 8. "As Is" portfolio example.

Visualize Resource Demand. System development is labor-intensive. Staffing needs determine cost and budget requirements. So, we need a way to:

- Assess high-level staffing needs
- Adjust Epic prioritization given budget and resource goals or limitations

The Monthly Avg Staff chart shown below at Figure 9 shows the staffing needs of the portfolio, by initiative, by month. Notice that this plan covers over three years. The big initiatives span 18 to 24 months and require multiple teams. There are almost always constraints affecting the entire portfolio. In this case, the organization requested that the portfolio not exceed a peak spending rate of $2.2 million per month and that the staffing capacity not exceed 225 people. The "as submitted" plan exceeded the organization's current and near future capacity. The goal to adjust the schedule to stay below 200 FTEs was easily achieved in SLIM by changing start dates, adjusting scope, and verifying the reasonableness of proposed productivity assumptions.

We have several customers whose strategic initiatives are constrained by staff; not due to the budget, but because of the limited availability of people with the special skills required to develop the product or service. Being able to visualize resource demand lends a different, often extremely important perspective to Portfolio Backlog Item prioritization.

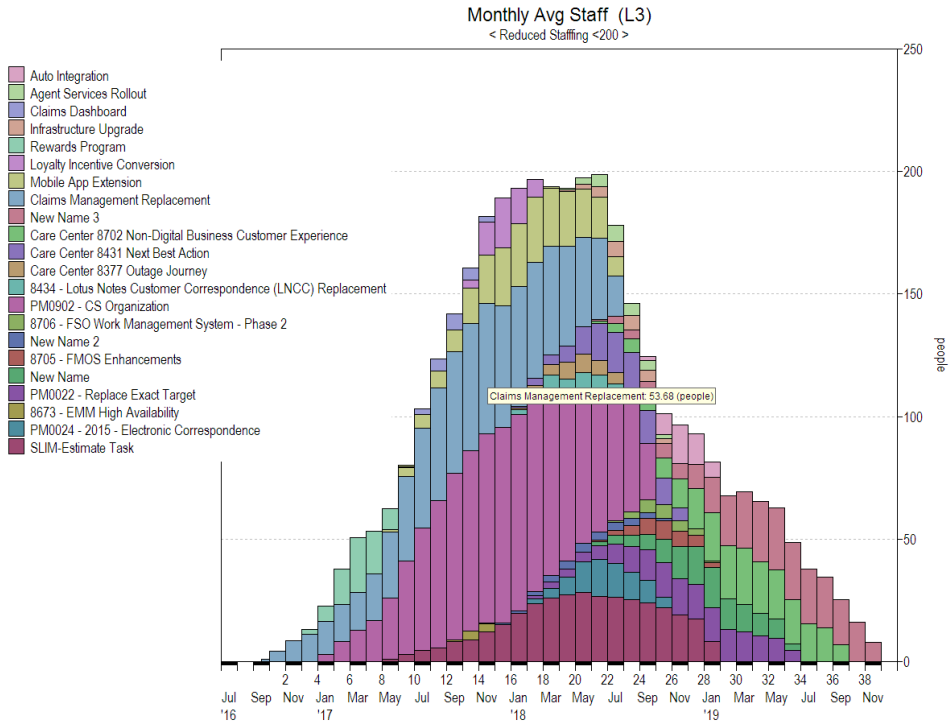


Figure 9. Monthly average staff resource chart.

Forecast Points. Estimates are usually required early in the lifecycle, when we know the least about the big idea. The Portfolio's IT budget plan consists of Feasibility estimates that have been adjusted to meet the organization's critical constraints. As Portfolio Backlog Items make it through

the Kanban process and receive approval, more information, particularly about scope and size, is available, enabling managers to explore ART allocation strategies. This is the Detailed estimate.

- Estimates are needed…
  - Feasibility Estimate – little information; high degree of uncertainty
  - Detailed Estimate – review & analysis are complete; PBI

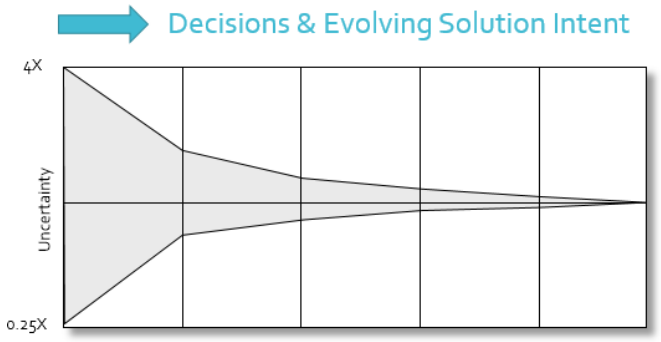


Figure 10. Cone of Uncertainty.

In the general agile community, the famed Core of Uncertainty is used to refer to scope management (Figure 10 above). Agile's focus on frequent, consistent feedback to stakeholders reduces risk that the product fails to meet the desired needs or expectations. Sound estimation processes also embrace the Cone of Uncertainty, because uncertainty cannot be eliminated – estimates are uncertain by definition. The good news is that we can measure the uncertainty of scope, effort, schedule and quality, and deal with it. Specifying uncertainty ranges for estimate inputs produces ranges of potential outcomes for the outputs.

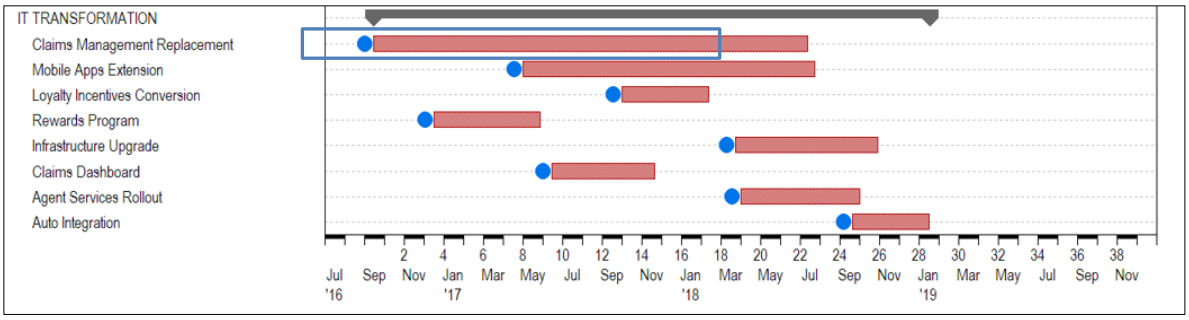


Figure 11. IT Transformation Value Stream – Claims Management Replacement initiative project estimated using SLIM®'s agile template.

Feasibility Estimate – Consumable Release

The Claims Management Replacement initiative was estimated as a Consumable Release, using SLIM's agile template (Figure 11 above). Two major groups of activities or phases are included: Story Writing, and Build and Test. My colleague, Dr. Andy Berner, presented a two-part webinar in 2015, entitled "Agile Estimation: Beyond the Myths." I will refer you to Andy's presentation, accessible on the QSM website at http://www.qsm.com/webinar/agile-estimation-beyond-myths-part-1 for details about how agile development is modeled in SLIM. A couple of key points he makes are:

- Phases define groups of activities, not when they occur. The high degree of overlap of these phases, shown on this staffing chart, models the work done by cross-functional teams during each sprint to deliver working code.
- "Potentially shippable is a description of quality, not value.
- There is a group of functionality that makes up a **Minimum Viable Release** that users can take and use.

This epic was sized at 2000 Story Points. An "out of the box," trend-based solution was computed (see Figure 12), because an estimate of size is the only input. It uses either industry or custom trends of completed projects, taking the average duration and effort for projects of this type and size. The resulting estimate reflects the resources required to develop a typical project of this type and of this size. This release is estimated to take 12.4 months, close to 55,000 phr, and cost more than $7 million.

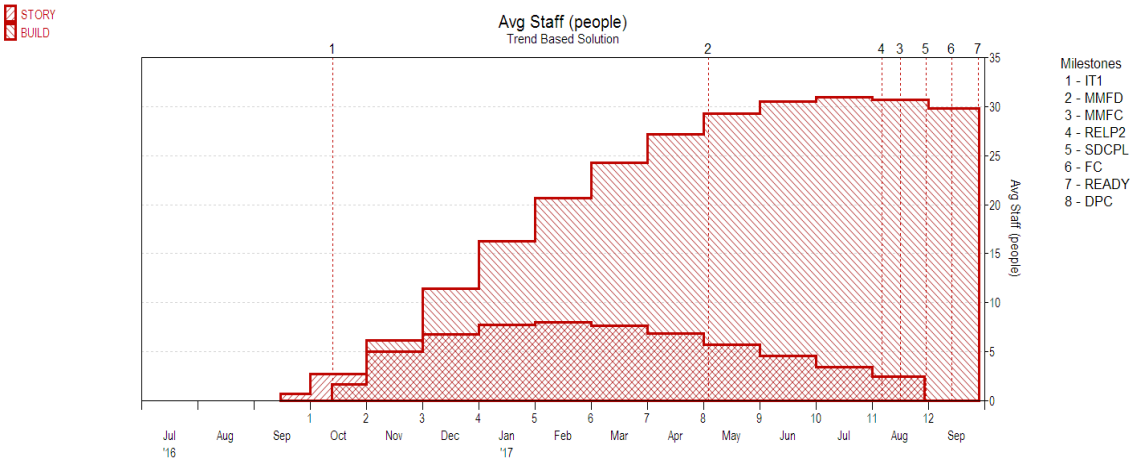


Figure 12. Trend-based solution for the Claims Management Replacement initiative.

SLIM Production Equation

To help you understand the basis of the Feasibility and Detailed estimates, it is important to share some key concepts and relationships. Both the Feasibility and Detailed estimates are top-down, scoped based estimates. The SLIM Software Production Equation (Figure 13) was developed by Larry Putnam, Sr., over 40 years ago. Its foundation is the Rayleigh statistical curve that models how staff is applied to iterative development projects and models the non-linear time and effort trade-off.

At its most simplistic form, we see that Size, a proxy for business value, is delivered by applying some amount of Effort over some period of Time, at some level of Productivity or Efficiency. This equation can be rearranged to use what we know to solve for what we don't know.

SLIM's Productivity Index (PI) is not simply units produced per time. It is a combination of:

- Capability: How effective is the development process?
- Difficulty: How challenging is the task?

Figure 13. SLIM® production equation.

Productivity is affected by numerous factors, so PI encapsulates all environment factors such as tools and methods, processes, people, and development methodologies. SLIM calculates Productivity from completed project metrics:

$$\textbf{Productivity (PI)} = \textbf{Size} / (\textbf{Effort}^{1/3} * \textbf{Time}^{4/3})$$

Detailed Estimate – Program Increment

The Feasibility estimate is critical for early strategic planning to set reasonable expectations and, for many organizations, provide the basis of proposal bids and negotiation. When it comes close to the time to release the Epic to the Value Stream, we need to anticipate the breakout of scope into Features and Capabilities to support Program Increment Planning, along with alternative scenarios regarding how these might be allocated to the ARTs or Teams.

The staffing profile for the Detailed estimate (Figure 14) looks different. It is for one Team, for one Program Increment.

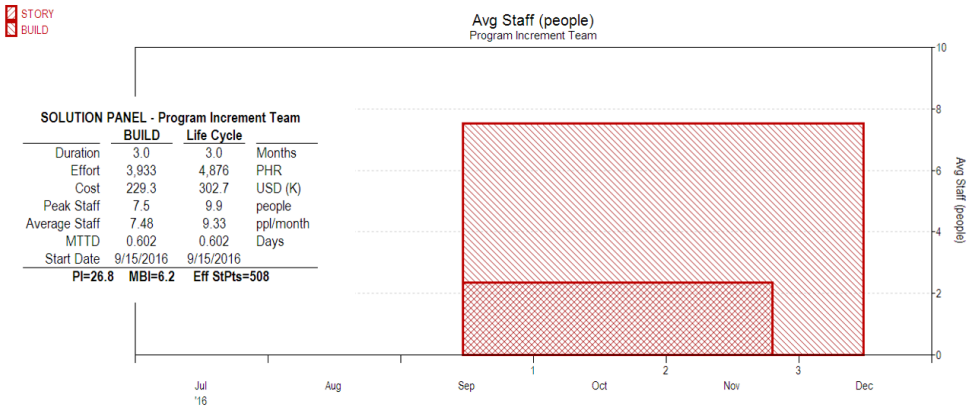


Figure 14. Staffing profile for the Detailed estimate for Claims Management Replacement.

At this detailed level, the forecast is Time Boxed: Duration is fixed at 12 weeks (or 3.0 months) and the Team is fixed at 10 people. Productivity Index is taken from history, so the number of Story Points this team is capable of producing is what is being calculated, i.e., the forecasted capacity.

Visualize Velocity and Value Creation

This Gantt chart (Figure 15) shows a conceptual plan for how development of the Claims Management Replacement release might be achieved. There are a number of potential options or scenarios for how Product Backlog Items may be assigned to Teams. This is just one example scenario.

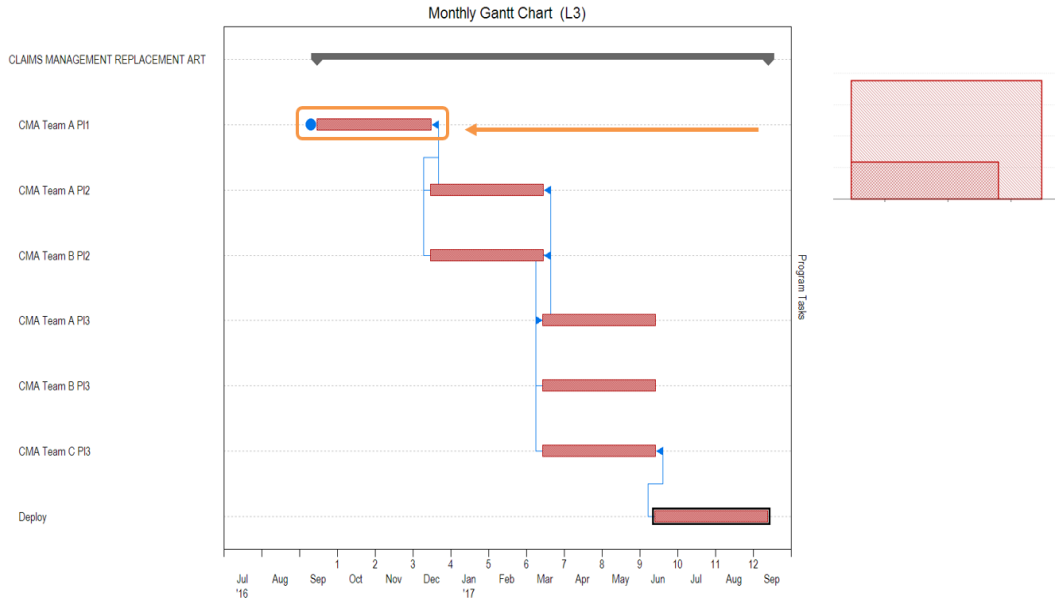


Figure 15. Gantt chart showing conceptual plan for development of Claims Management Replacement release.

Each bar is an estimate of the work to be done by one Team during a particular Program Increment. The Epic must be broken down into smaller Features and Capabilities and prioritized, so the detailed estimates for each Program Increment include both Story Writing and Build and Test. Backlog items will be released to three Teams, starting with Team A for the first Program Increment, then adding one more team each subsequent Program Increment. We had a customer using scaled agile who took a very similar approach to their large delivery. Notice the dependencies between Program Increments. In this example they are strictly based on schedule, that is, the Program Increment cadence, but SLIM's capability to establish dependencies between tasks also allows for modeling product development dependencies among Features and Enablers. Because the estimate supports release planning, a Deployment task has been included to account for system-level release management and DevOps activities we reviewed earlier. This solution accounts for other initiative features and stories for which this ART will be responsible. The power is in the ability to quickly and easily assess the relative risk and strategical merits of numerous alternatives.

A report view of the Gantt chart (Figure 16) shows the start and end dates, elapsed time, effort, and cost for work to be done for this Epic – traceability that continues to support a vision of value creation. The Feasibility estimate was for 11.5 months and $2.6 million dollars. The SAFe Implementation Detailed Estimate requires 12 months, yet it saves at least $0.5 million dollars.

Gantt Report (L3)
Release

Task	Task Description	Start Date	End Date	Elapsed Months	PHR	Cost (USD 1000)
CLAIMS MANAGEMENT REPLACEM...	Summary Task	9/15/2016	9/12/2017	11.93	34,252	2,101
CMA Team A PI1	SLIM-Estimate (CMA ART Team Est TB....	9/15/2016	12/15/2016	3.02	4,876	303
CMA Team A PI2	SLIM-Estimate (CMA ART Team Est T...	12/15/2016	3/14/2017	3.00	4,808	299
CMA Team B PI2	SLIM-Estimate (CMA ART Team Est T...	12/15/2016	3/14/2017	3.00	4,808	299
CMA Team A PI3	SLIM-Estimate (CMA ART Team Est T...	3/14/2017	6/13/2017	3.01	4,842	301
CMA Team B PI3	SLIM-Estimate (CMA ART Team Est T...	3/14/2017	6/13/2017	3.01	4,842	301
CMA Team C PI3	SLIM-Estimate (CMA ART Team Est T...	3/14/2017	6/13/2017	3.01	4,842	301
Deploy	SLIM-Estimate (CMA ART Team Est T...	6/13/2017	9/12/2017	3.00	5,233	298
RELEASE	Release	9/15/2016	9/12/2017	11.93	34,252	2,101

Figure 16. Report view of the Gantt chart for Claims Management Replacement release.

Regardless of whether funding is actually disseminated according to SAFe's recommendation that funding be allocated to the Value Stream or Program, not to projects, it must be estimated and budgeted. Now, with a detailed forecast, we also have a vision of when Teams will be engaged, helping plan for resource demand. Remember that this Epic is only one in the Value Stream.

SLIM's program management tool's "What-If" capability (Figure 17) lets you calculate different forecasts and explore a range of potential outcomes by altering the major inputs of the Production Equation:

- Size, shown here in Story Points
- Productivity Index
- Effort, shown here as Peak Staff

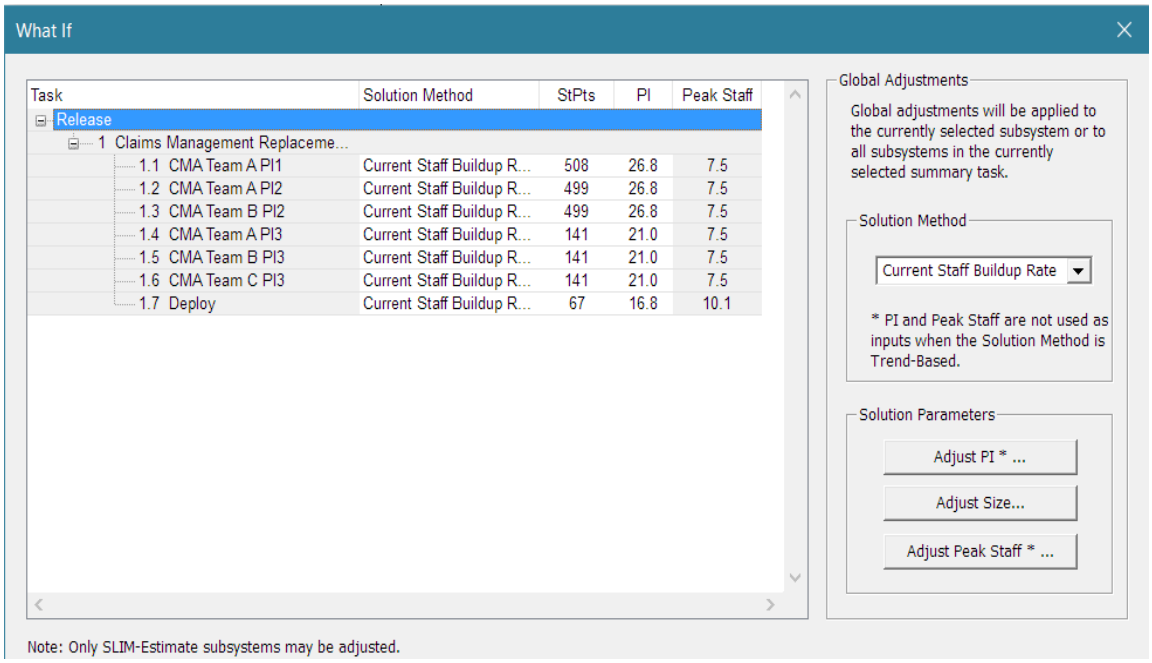


Figure 17. "What-If" view showing alternate possibilities and potential outcomes for Claims Management Replacement release.

To preserve the nonlinear development of Value between the Feasibility and Detailed estimates, we adjust for the reality that velocity is not constant across all program increments in an ART. What does that mean? As we build up an increasing amount of functionality, there is additional overhead we incur with backward integration not only with Program Increments in this cycle, but software that was delivered in previous Program Increments. If we keep the time box and staffing fixed, it can only mean that the amount of software that can be newly developed and integrated must decrease. It also accounts for emergent design and refactoring. This causes our internal Productivity Index to go down for subsequent Program Increments.

Microsoft published an article in the Microsoft IT Showcase, entitled, "Meeting the Challenges of Agile Development at Enterprise Scale." They state that agile worked well for small teams, but when they scaled to large, enterprise-sized projects, they ran into bottlenecks that slowed the team down and reduced quality. As teams grew, so did project complexity, and their productivity suffered. No one knew how much work was required for big projects.

Updating the Forecast

A major benefit of creating the Program Increment Release Plan is the ability to track actual progress and forecast expected release dates based on actual performance (Figure 18). The green bar in the Gantt chart indicates an in-progress release. Counts of Story Points completed each sprint, a standard practice in agile methods, allows SLIM to calculate the implied actual Productivity Index. Running a forecast of Story Points that can be completed in the current Program Increment based on actual performance will dynamically update the Release Plan and allows more What-if analysis to explore outcomes based on alternative backlog prioritization and team utilization as development progresses.

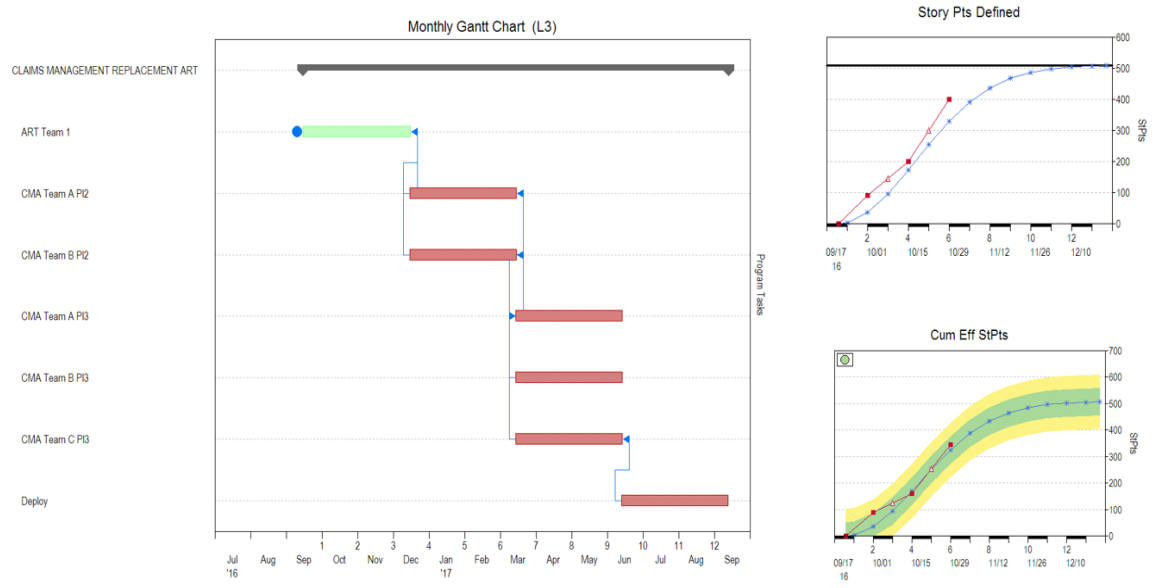


Figure 18. Tracking actual performance and forecast against planned performance.

Summary

Top-down, scope-based estimating supports many principles of the SAFe. Product and Release Management Teams at Portfolio, Value Stream, and Program levels are equipped to visualize potential release plans for Consumable Value, to ensure the timing, budget, and resource requirements are aligned with the enterprise's strategic business objectives. A top-down approach allows you to build a defensible estimate based on known capabilities at the development environment level, allowing quick and easy vision of alternative scenarios and their relative risks.

---

4. BEST PRACTICES

"People think that computer science is the art of geniuses, but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones."

– Donald Knuth,
American computer scientist, mathematician, professor emeritus at Stanford University, and author of the multi-volume work The Art of Computer Programming

"People in any organization are always attracted to the obsolete – the things that should have worked but did not, the things that once were productive and no longer are."

– Peter Drucker,
Austrian-born American management consultant, educator, and author, whose writings contributed to the philosophical and practical foundations of the modern business corporation

"It is not the strongest species that survive, nor the most intelligent, but the most responsive to change."

– Charles Darwin,
English naturalist, geologist and biologist, best known for his contributions to the science of evolution

"Intelligence is the ability to adapt to change."

– Professor Stephen Hawking,
English theoretical physicist, cosmologist, and author who was director of research at the Centre for Theoretical Cosmology at the University of Cambridge

Re-Discovering the Rosetta Stone: A Strategic Method to Software Sizing

Victoriano Fuster III and Taylor Putnam-Majarian

*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.*

Estimating IT budgets has never been an easy task. Companies struggle with this across the globe, and the U.S. government, including the Department of Defense (DoD), is no exception. In support of our clients, QSM® is often asked to evaluate estimates put forth by other analysts and program offices. One of the most significant inputs in top-down parametric estimation methods is the scope, or size of the program. Consequently, in our role as estimators, the size of the program is one of the first metrics we aim to better understand. To facilitate that process, we'll ask metaphoric questions such as "I*s the system you are estimating more like a Little League Baseball field or a Major League Baseball stadium?*"

Figure 1. If you were building a baseball facility, would the scope of the program be most like the Little League Baseball field or the Major League Baseball stadium?

Depending on who we are speaking with, we often get very different answers to that question. Since many must size programs using analogous methods, in much the same way one chooses a T-shirt to wear (e.g., S, M, L, XL, etc.), it should come as no surprise that estimating a system size at the beginning of the software development lifecycle can be a challenge.

What Is "Size"?

Before getting into too much detail, it is important to first understand what is meant by the term "size." For the purpose of this article, size is a proxy for the value and knowledge content of the delivered system, i.e., what the system is worth. The size of a system can be quantified in a variety of ways. A list of these types of sizing units and categorizations can be found below in Figure 2.

Front end: Unit of Need	Back end: Unit of Work
Based on characteristics of the statement of needs	*Based on the characteristics of the system when built*
• Requirements	• Lines of Code
• Function Points/Object Points	• Statements
• IO Counts	• Actions
• States/Events/Actions	• Modules
• Use Cases	• Subsystems
• Stories/Story Points/Epics	• GUI Components
• Objects/Classes	• Logic Components
• Components	• Logic Gates
• Design Pages	• Tables
• Web Pages	

Figure 2. Units of need versus units of work.

Generally, the front-end methods (*units of need*) tend to be less precisely defined, while predicting the count of back end methods (*units of work*) tends to be less precise. So, when is the appropriate time to use each method? As shown in Figure 3[1], as the project progresses through the software development lifecycle, different sizing methods may become available as more information is known.

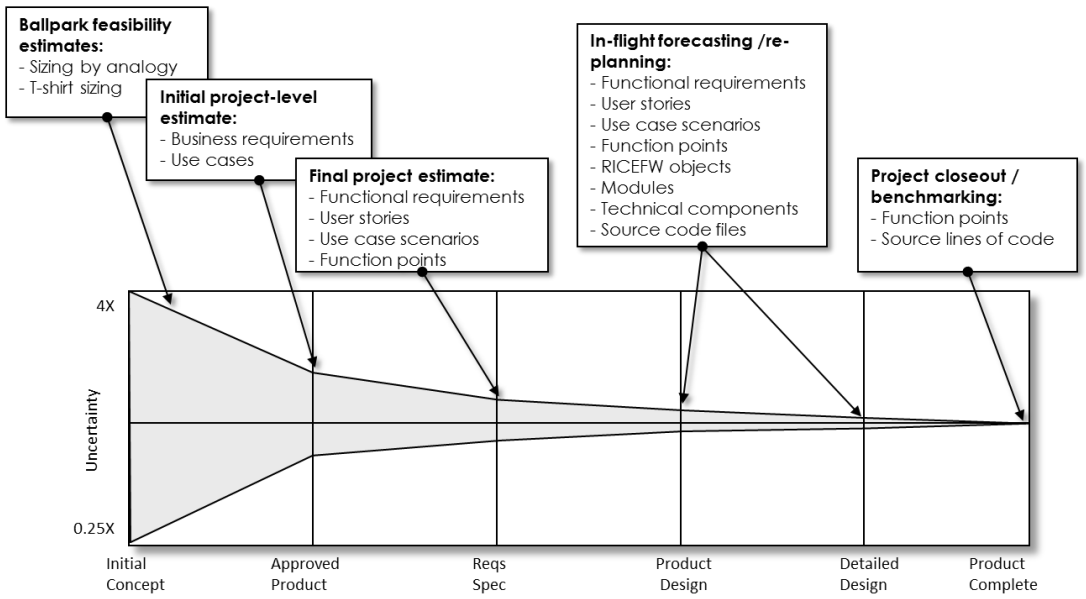


Figure 3. The Cone of Uncertainty.

The center line in this diagram represents the actual size of the program. In the beginning when very little information is known, analogous sizing methods will be the most useful but will also carry the most uncertainty. The estimate could be ¼ of the actual size, or four times as large. However, as the program progresses through the software development lifecycle and more information becomes available, the uncertainty around the estimate decreases. For example, once the requirements have been written, the size estimate will be more precise than the analogous methods used previously. Requirements can be broken down into smaller, more specific sizing methods, such as RICEFW (Reports, Interface, Conversion, Enhancements, Forms, and Workflow) Objects or Stories. These units can be subdivided further until they are in their simplest form, a *basic unit of work,* which often equals a Source Line of Code (SLOC) or Line of Code equivalent. We have seen all of these popular methods used in attempt to size various programs. While some organizations have experienced greater success using one or several of these methods, as opposed to another, they are all valid methods for quantifying the scope of a system.

Sizing Challenges

Theoretically then, the challenge would be to figure out how to effectively size the system using any one of these methods. However, there are actually a number of sizing-related challenges faced before even getting to that point. In general, sizing can be an emotional topic for scope-based estimation methods because the outcome depends so much on the size estimate being accurate. One point of contention is that people speak vastly different *languages* when it comes to software product sizing. Often, individuals, such as those deciding the tasks that the system will perform, will be familiar with one method of sizing (i.e., requirements), whereas those who are building the system may be more familiar with another (i.e., user stories or SLOC). These individuals may have difficulty communicating with one another because they view sizing from different perspectives (see Figure 4).

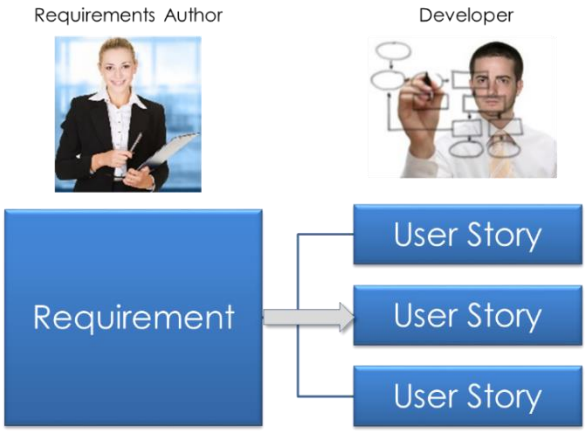


Figure 4. Requirements authors and development teams may have difficulty communicating the same amount of work because they approach sizing from different perspectives.

To mitigate this challenge, we will try to translate between these two different environments, but often they are at odds. The requirements author may view an individual task or "shall statement" as the size. At the beginning when the only size known is quantified using requirements, it can be difficult to interpret how many user stories or SLOC the final system will produce because requirements are much more abstract. On the other hand, a technically-minded development team may know how long and how much effort it will take them to complete a certain number of user stories, but they may not be able to initially read a requirement and determine how many stories it

will produce. Since these two people view size from different perspectives, it can often be challenging for them to communicate effectively with one another.

Another issue is that defense systems tend to take many years to build and maintain. It is not uncommon to see new technology and development methods, and their subsequent sizing units, emerge over the course of the development lifecycle. For example, we have observed ERP systems (typically sized in RICEFW Objects) that begin their development using more traditional methods. However, as contractors have begun to adopt a more agile approach to software development, the sizing method used have shifted to Stories. Some of the programs we have observed now have a mix of several different sizing methods in use, making it difficult to track the program's progress. If there is no meaningful way to translate from RICEFW Objects to Stories, it can be incredibly difficult to quantify how much work has actually been completed and what is still left to do.

If analysts are not careful, this type of situation has the potential for grave consequences, especially if there is no way to adequately track and audit the work completed. Together all of these challenges bring to light the need to further examine the relationships between the sizing methods. In our research, we have sought to combat some of these issues by examining the relationship between some popular sizing methods. We hoped to be able to create a translator, so to speak, that would allow us to be able to convey the same size in a variety of different sizing methods, and we looked to the Rosetta Stone for inspiration.

The "Rosetta Stone" Method of Translation

Discovered in 1799 near the town of Rashid, the Rosetta Stone translated the same text into three distinct languages: Demotic script, which was the common language in Egypt at the time; Ancient Greek, the language used by ancient Egyptian rulers; and Ancient Egyptian hieroglyphics, which previously could not be understood. That is, until a gentleman by the name of Jean-François Champollion cracked the code. Champollion already had a working understanding of Ancient Greek and Coptic, which he used to interpret the Demotic text. From there, he used all three of those languages to decipher the hieroglyphic symbols. Ultimately, Champollion's work became essential in understanding Ancient Egyptian literature and civilization.

The reason for sharing that brief history lesson is because the methods utilized by Champollion were quite similar to the methods we used in our sizing research. Much like how the Rosetta Stone translated the same meaning into three different languages, our hope was to convey the same functionality in different popular sizing methods. Each sizing unit has a "relative weight" which measures the unit's "complexity" and is called a *Gearing Factor*. Gearing Factors are common denominators, calculated by sizing the same program at various points throughout the software development lifecycle using different sizing methods. By definition, the relative weight of a Line of Code (LOC) is 1. At QSM, we use a neutral term, *Implementation Units (IU),* to stand for a line of code equivalent.

Table 1 shows an example gearing factor table, demonstrating a notional relationship between Implementation Units and an imaginary sizing unit called *Purple Dragons*. A Purple Dragon could be a Requirement, a User Story, a RICEFW Object, or any other sizing technique used by an organization. For the purposes of this article, a purple dragon (PD) is used to explain the concept of how to calculate and use gearing factors.

Table 1. Example gearing factor table.

Component	Number of Components	Gearing Factor	Total Implementation Units (IUs)
Simple Purple Dragons	**25**	**600 IU/PD**	**15,000 IU**
Average Purple Dragons	**35**	**1200 IU/ PD**	**42,000 IU**
Complex Purple Dragons	**10**	**5500 IU/PD**	**55,000 IU**
Dragons	**70**		**112,000 IU**

Suppose that a sample project has a total size of 70 Purple Dragons. Upon closer examination, the 70 total purple dragons are comprised of 25 simple, 35 average, and 10 complex purple dragons. After examining the relationship between IU and purple dragons with several historical projects, it was found that these different complexity levels have an associated average IU equivalent (*simple = 600 IU, average = 1200 IU, and complex = 5500 IU respectively*). In order to determine the total size of the program in Implementation Units, one must simply multiply the number of purple dragons by the respective gearing factor to obtain the size. In this case, the 70 purple dragons equal about 112,000 IU. Once a set of gearing factors have been established, it is now possible to translate between the different sizing techniques used. A visual of this concept is shown at Figure 5.

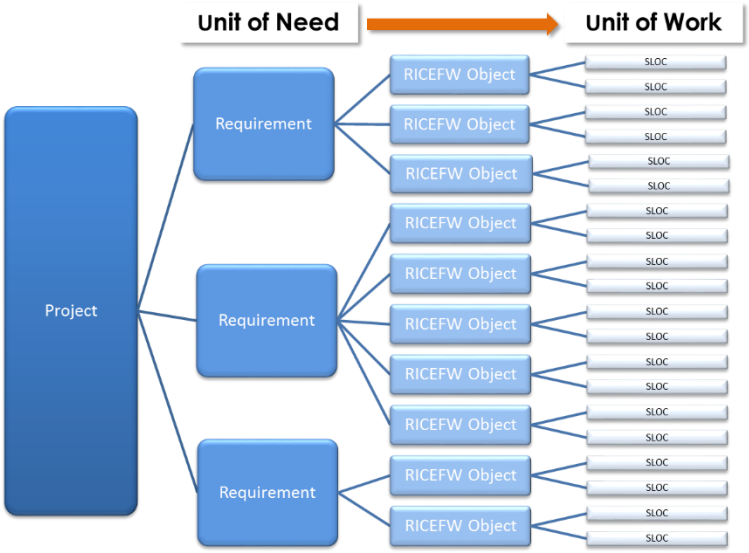


Figure 5. Sizing a project using multiple techniques.

A project is typically made up of several different requirements, of varying complexity. Each of those requirements might be comprised of several smaller units, such as RICEFW Objects or Stories. While there may be an average number of RICEFW Objects per requirement, some larger, more complex requirements may require more RICEFW Objects, while smaller, simpler requirements may need fewer. The RICEFW Objects, can be further subdivided into even smaller sizing units, and so on, until the program can be sized using a basic unit of work. In the example above, the project has been sized using three different techniques throughout its development lifecycle. By examining the relationships between these sizing methods, and after repeating this process with several other programs, one can create a basic rule of thumb for sizing.

That is exactly what we did. QSM examined a sample of over 150 completed and validated projects that reported sizing metrics in two or more methods. Sizing methods included: SLOC (*n = 150*), Requirements (*n = 65*), Function Points (*n = 100*), Use Cases (*n = 10*), User Stories (*n = 10*), and RICEFW Objects (*n = 85*). The sample was comprised of projects from over 100 organizations and Business, Engineer, and Real Time domains were represented. First, we examined the relationship between a sizing method and its base size unit (SLOC/ IU) to calculate a set of gearing factors (see Figure 6). Then we examined the relationship between the same sizing method and any "container" sizing methods (e.g., functional requirements and business requirements). Finally, a Certified Function Point Specialist conducted an independent function point count on a representative sample of the programs to give us a third set of gearing factors. Since Function Point counting can be a more objective sizing technique, due to its ISO standards, this sizing method has also been implemented as a "sanity check" against other sizing techniques. Once complete, we were able to triangulate the size using multiple sizing techniques to produce a range of gearing factors for each sizing method.

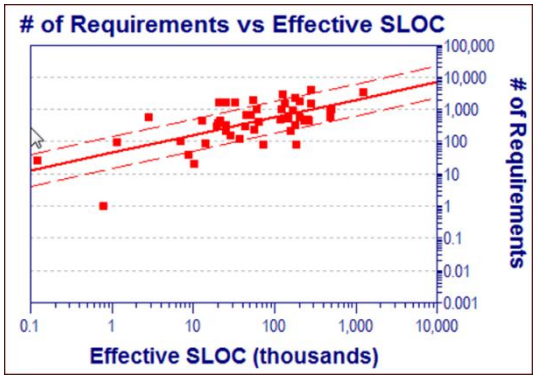


Figure 6. Scatterplot showing the relationship between Effective SLOC and requirements.

Similar to translating language via the Rosetta Stone, if the size is known in one unit, all the other popular sizing methods studied could have calculated equivalent sizes (see Figure 7). For instance, if the number of functional requirements was counted, that amount of functionality could be "translated" into Function Points through use of a gearing factor, and then translated once again into all of the other methods mentioned.

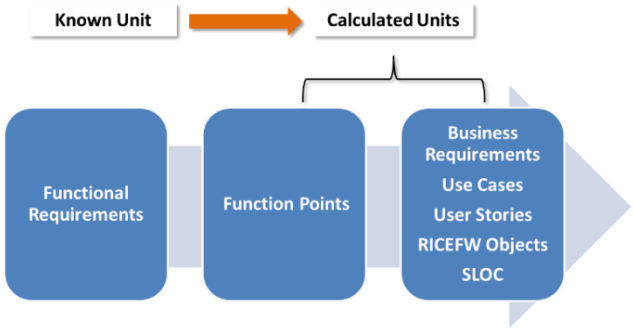


Figure 7. Sizing translation method.

Putting Theory into Practice

One of the reasons we conduct this sizing research is so that we have a methodology in place to support our clients. Since beginning this initiative, we have had the opportunity to use our sizing translator to support our work with actual programs in a variety of ways. The objective is to leverage the understanding of these popular sizing methods to improve estimates at each stage of the software development lifecycle.

Probably the most obvious application of the sizing translator is for crosschecking estimates. *Is the size estimate reasonable compared with similar completed programs? Does the estimated size encompass all the desired functionality?* These are the types of questions to facilitate those initial discussions, as well as after the program's kickoff. For projects that are already underway, the sizing translator can be very useful in progress tracking, especially if a project is using more than one sizing technique. If a program that originally was sized exclusively using RICEFW objects begins to also include agile stories, it is now possible to determine the total size in IUs for both RICEFW objects and stories, and then convert the total size into one common unit for comparison purposes. This enables quantifying the size developed, so that projects do not fall behind, or inadvertently grow without approval.

With these tools in place, conversations between requirements authors and development teams can happen much more easily, and all team members can be on the same page. Development teams will be able to convert the requirements into a more familiar unit of work, which will allow them to develop more accurate estimates of the time, effort, and staff needed to complete the work, when bidding on the work. Meanwhile, program offices can conduct audits of vendor bids, by modeling a proposed bid, to determine whether it is consistent with typical trends in the industry. If the bid differs greatly from historical performance, it might be an indicator that the estimate's assumptions could be off and that it is time to ask some additional questions.

To conclude, the Rosetta Stone method of translating between various sizing units was intended to facilitate sizing-focused discussions. Since software estimation and development is really a "team sport," it is critical that all the "players" are able to communicate effectively in order to adequately contribute. Using sizing translator methods such as these can "level the playing field," so to speak, such that everyone is on the same page when it comes to sizing and, ultimately, make better economic decisions.

References

Adapted from: Boehm, *Software Engineering Economics (1981) &* McConnell, *Software Estimation (2006)*

How to Avoid the Three Top IT Project Risks

Douglas T. Putnam

This article originally appeared in the May 18, 2017 online edition of GCN and is reprinted here with permission.

For a number of years, the federal government has been on a mission to reduce waste and enhance efficiencies across departments, including IT. A recent example of this is President Trump's innovation team, which is powered by some of the tech industry's biggest names and echoes a similar program that President Barack Obama initiated in 2008. The goal for both programs: streamline government, raise accountability, and improve performance — without sacrificing innovation.

Unfortunately, these efforts have fallen short of the mark, at least in terms of IT. In fact, according to the CIO Council's 2017 State of Federal Information Technology report, 43 percent of the federal government's $80 billion in IT projects cataloged in September 2016 were listed as over budget or behind schedule.

Inadequate project scoping and estimation are key factors behind this alarmingly high percentage. As outlined in the Government Accountability Office's (GAO's) Cost Estimating and Assessment Guide, "Software is a key component in almost all major systems the federal government acquires. Estimating software development, however, can be difficult and complex. Most often, creating an estimate based on an unachievable schedule causes software cost estimates to be far off target."

The GAO guide further highlights how bad estimates stem from "a lack of understanding of how staffing, schedule, software complexity, and technology all interrelate."

GAO advocates for a scope-based parametric estimation approach, which has proved to be much more accurate and effective in supporting software project planning compared to role-based and task-based estimation processes. Role-based approaches rely on the knowledge of subject matter experts, but that knowledge can vary wildly in terms of its accuracy. Task-based approaches require an enormous amount of detail that's often very difficult to incorporate into the critical early planning stages of a software project and are not conducive to change.

Conversely, the scope-based parametric approach provides stakeholders with a more accurate and flexible scoping and estimating solution that is grounded in historical data. Estimators can create a number of "what if" scenarios early in the project's life cycle so they can better plan for the inevitable challenges and roadblocks that will occur during development. Knowing these ahead of time allows them to more accurately scope costs, timeframes and staffing required to finish the project on time and within budget, thereby avoiding some n mistakes that lead to overruns. Let's take a look at some of these pitfalls and how parametric estimation can help government CIOs and their teams avoid these traps.

Inaccurate Estimates

One of the biggest mistakes CIOs make in IT portfolio planning is not using a robust, scientific method to estimate software project duration and effort demand at the outset of a project. Too many rely on informal role-based or tasked-based estimation approaches to answer key questions: What is a truly realistic timeframe for completion of this project? How many people will we really need to make this work? How much will it actually cost?

The danger with these approaches is that they are often overly optimistic and fail to consider important factors that will impact the project's completion, such as complexity, team efficiency, or requirements growth. The result could be a HealthCare.gov or Defense Integrated Military Human Resources System-level debacle, a project marked by poor quality and cybersecurity deficiencies or, worse, the need to completely scuttle a project.

Parametric estimation involves a top-down process that allows agencies to leverage historical information and forecasting models to more accurately assess projects from the beginning. It provides a flexible framework that can be re-forecasted as requirements change. This helps CIOs address potential risks before the start of a project and mitigate those risks throughout development.

Overstaffing of Projects

In 1975, software engineer and author Fred Brooks wrote that "adding manpower to a late software project makes it later." Yet in 2017, the practice of throwing more bodies at a problem still persists. In fact, research shows that overstaffed projects result in more defects, increased costs and the need for rework. They also often take longer to complete.

The reason is simple math. Fewer people means a more streamlined and efficient process. Adding more people to a project increases the necessary communication paths and adds to the complexity of the process, leading to miscommunication, software defects, and increased costs.

CIOs should model appropriate staffing levels in relation to their IT portfolios during the planning stages of every project. They must analyze and assess strategies to ensure that the staff they are committing to a project accurately matches their demand versus capacity planning models. While some teams may need to be larger than others, smaller teams will be more efficient, nimble, and productive.

The "smaller team size is better" rule should apply even with projects that may be running behind schedule. Adding more people to a late running project will only compound the problem and slow things down even more. Aside from adding to the cost of the project and introducing new lines of communication, developers already working on the project will need to take time to train the new people -- slowing things down even more.

Denial of a Project's True Status

No one wants to admit when a project is in trouble, but a system that allows for quantitatively measuring progress enables a CIO's team of project managers to promptly take corrective action to keep projects on track. Conversely, waiting until the end of a project can have an adverse impact on any follow-on activities. Look no further than the problems plaguing the F-35 Joint Strike Fighter program to see how waiting too long to course correct can cause major issues.

If there is even a hint that things are falling behind or going wrong, managers should stop, re-estimate, and take an honest look at what it will take to correct any issues. If what they discover is unacceptable, they may need to reduce their scope, examine different options, and come up with a more realistic plan.

Fortunately, agencies that have implemented agile development methodologies are already ahead of the game in their ability to leverage parametric estimation and tracking. Agile embraces a scope-centric approach to estimation and planning that recognizes that the longer the duration of a project, the more likely it is to be impacted by changes. Agile projects are built in short iterations, which means that quantitative data on product construction is readily available on a routine basis for tracking and forecasting in a parametric tool.

Parametric estimation provides agencies with a solid foundation for project success that doesn't tie them into a rigid framework. This combination of flexibility, reliable historical data and proven mathematical algorithms is the key to helping agencies gain efficiency, stay on track and still deliver innovative software programs.

Data-Driven Approach for Defense Acquisition Accountability

Victoriano Fuster III

As the Pentagon continues to reorganize how they oversee and implement Defense IT Acquisition and demand a higher level of accountability from program managers, many acquisition professionals feel mixed emotions. On one hand, those who have been calling for reform for years may be reinvigorated by such progressive intentions. However, there are also struggling Acquisition program managers and support contractors` likely feeling they have a more focused target on their back. New and reorganized Defense Acquisition offices surely be eager to show their value to the Pentagon in their responsibilities to improve the DoD Acquisition process. Particularly, as the DoD continues a focus on the DoD IT Business transformation priorities and ensuring that they are acquiring effective Defense Business Systems with capabilities to support those priorities, an observation may be that they lack consistency in how they manage that information systems acquisition process.

Accountability Requires Consistency

The word "accountability" often comes up in the discussions for reforming the way the Pentagon provides oversight to the DoD Acquisition programs and the vendors who support them. Many pundits say the DoD must increase accountability to reduce the risk associated with project success. Although the sentiment and importance of that view should not be debated, it begs the question of how they cannot seem to achieve accountability despite all the current milestone reviews, assessments, and reporting already mandated by acquisition rules. One major factor to be considered is *consistency*. There appears to be a lack of consistency in the application, approach, and results of traditional processes that exist to reduce risk and increase accountability in program acquisition. Program scenarios should be better scrutinized before being given waivers or treated

differently for convenience across the present landscape. Nor should decision-makers accept as all "green" those program status quad charts that are not based on actual program performance data. The DoD requires consistent, repeatable processes for acquisition programs if they are to measure and benchmark progress across the DoD; doing so enables them to legitimately focus on accountability. Only when measures of success are consistent and horizontal across the acquisition portfolio can the DoD realistically hold program managers accountable.

Data-Driven Approach to Acquisition Program Management

The utilization of quantitative management methods are not new. They have, however, certainly seen a resurgence in their use and appreciation in recent years. The progressive use of business analytics applied to diverse industrial sectors, including DoD programs, has ushered in an era of expectation for data-driven insights that some are apparently still struggling to realize. The DoD position as innovator of many technologies over the last few decades should enable its strategic position for this wave. But, as various DoD Acquisition programs are struggling to meet cost and schedule milestones, it appears an obvious opportunity to explore new or updated program management approaches. The DoD initiatives to reach out to Silicon Valley for software/IT best practices are one of those attempts to positively affect the Pentagon's acquisition process, as it is admitting it is time to look to others for support. Undoubtedly, the DoD does currently use data-driven, quantitative methods in many areas, as both government and contractor teams deliver these capabilities daily. However, consistent application across programs is lacking and the current environment provides an opportunity to achieve the horizontal data benchmarking capability that is critical for enabling accountability.

There are many benefits to using a proven, data-driven approach including:

- ✓ *Information-centric approaches* to program management that aligns with acquisition reform objectives.
- ✓ *Consistent, credible and defensible information* throughout the acquisition life cycle to enable more confident decision-making.
- ✓ *Risk reduction* through more accurate estimates and assessment of software project cost, size, effort and schedule during initial planning, re-planning activities, and ongoing monitoring of actual-to-planned project status.
- ✓ Programs with *quantitative insight needed to confidently make programmatic and contractual decisions* earlier, weighing impacts of proposed requirements changes, and identifying early requirements risk.

The DoD can realize these benefits by applying consistent, data-driven solutions throughout the life cycle such as:

- *Implement appropriate data collection* foundational and recurring activities utilizing approaches like Goal-Question-Metric (GQM) to ensure we are focused on relevant data and not drills to collect data just for the sake of saying we are doing it.
- *Create benchmarking products* that compare performance with DoD and commercial systems.
- *Use statistical control practices and tools* to continuously re-estimate and re-forecast cost, schedule, and performance expectations.

- *Evaluate other estimates* and proposals from vendors and critical stakeholders throughout the life cycle, leveraging the insight gained from our own data activities.

Keep Fighting the Fight

Communication is likely one of the main problems that challenges the use of these methods that some may still commonly see as only another "black box" activity. Whether it be ignorance or negative experience with uses of data analysis for program management, some just do not accept the value of having a quantitative insight to program health and progress, or merely as even just other indicator to complement current *status quo* management methods. There are certainly pockets of steadfast advocates using these data-driven methods, especially for the Defense Business Systems environment, which are often data rich. Still, it seems that for every organization effectively using the practice, there exist many more skeptics and the cycle continues. When DoD Acquisition culture progresses to not only horizontally appreciate, but necessitate, a place for data-driven acquisition management methods across all programs, then they can start to get serious about a consistent approach to accountability.

Five Software Laws for Smooth Product Development

Lawrence Putnam, Jr., and Donald Beckett

This article originally appeared in the October 2, 2017 online edition of Software Executive Magazine, and is reprinted here with permission.

How to ensure your project will stay on schedule and within budget.

Business leaders want to be at the top of their game when it comes to software development. After all, making the wrong moves during the development process can cost time and money – and result in bug-ridden, defect-plagued applications. This is not something that corporate executives typically worry about, even though it can directly impact their companies' bottom lines and reputations. Executives are often removed from the daily ins-and-outs of software development and execution by necessity. They envision a company's whole trajectory, including long-term projects and goals that will lead to profitability. They simply do not have time to be a part of daily development meetings.

Even so, executives should take steps to ensure that they are firmly in the loop with software development projects, especially in the critical planning phases prior to project kickoff. One way to do so is to ensure that software development teams are mindful of five core software development laws. Following these laws makes it more likely that high-quality products will be built on schedule and within budget.

Law 1: Every software project has a minimum development time.

Unrealistic schedules cause projects to fail. Fred Brooks first identified this law back in 1975 in his book, *The Mythical Man-Month*. Though software has changed drastically, the law remains apt. Developers who set schedules that are too aggressive are doomed to come up short. No amount of wishing, bargaining, or rationalizing will change the fact that deliverables will be late if adequate time is not built into the schedule.

Fortunately, organizations can use metrics derived from past projects to forecast how long a new software build is likely to take. This requires some foresight as businesses need to measure, collect, and analyze certain build data to take full advantage of the power of metrics during forecasting. It's never too early to start collecting data on a project, and the more data executives and team leaders have, the easier it will be to create an accurate schedule for the next software development project or cycle.

Law 2: Schedule and cost do not trade off evenly.

While unrealistic schedules cause projects to fail, there is no single schedule that will guarantee project success. Projects can be executed under a range of schedules and budgets. Generally, executives can expect that the more aggressive the schedule the higher the cost and, ironically, the poorer the quality of the product.

Initially, such a generalization would not seem to make sense. But consider that even as project managers seek to tighten schedules, they often add people to a project. In an ideal world, the additional staff would come on board already up to speed and ready to work. Unfortunately, that is rarely the case when bringing new staff onto a project in progress. Existing staff have to take time away from their tasks to help train new staff.

Adding staff can negatively affect quality as well. As staff are added to a project, more communication paths are created which creates miscommunication. Communication disconnects become a challenge during the QA phase of a project when more people are testing products parts, fixing them, and then retesting. If a project already has an unrealistic deadline and staff are added (at increased cost) to get to the finish line, QA is one of the last parts of development and might be "trimmed" from the budget. That may bring a project closer to cost but actually can end up costing more over time because of the need to fix unforeseen errors later on.

Law 3: Projects grow.

Despite the best efforts of development teams, projects inevitably grow. According to QSM®'s own research, on average, scope increases 15 percent; schedules, eight percent; and cost/effort, 16 percent. Executives should keep these percentages in mind when building initial project schedules and budgets.

Everyone has the best intentions at the outset of a project. Teams are optimistic and budgets and schedules reflect that optimism. When project parameters are set, most executives are looking — as they should — at the high-level, abstract view. This view will be more accurate when informed with data from past performances, but it is, at best, an estimate.

Once actual work begins, a project's scope increases. Estimates become real numbers as work is actually performed. Unforeseen but necessary parts of a build are added but they are not accounted for in the schedule or budget, leading to problems later on. The solution is simple: Try to contain scope creep but realize that despite everyone's best intentions, it is inevitable. Build extra time and cost into the project from the outset.

Law 4: Small teams are better.

When faced with a large software development project, do not put more people on it and believe that doing so will bring faster results (see Law 2). Small teams are often more capable of delivering better-quality products in the same amount of time as large teams, and small teams cost less. Small teams also communicate more efficiently. They take less time to come to resolution when faced with

decisions. In agile workflows, people are used to working in small teams. Members often know each other's roles and can move fluidly through an entire project's development, further reducing cost.

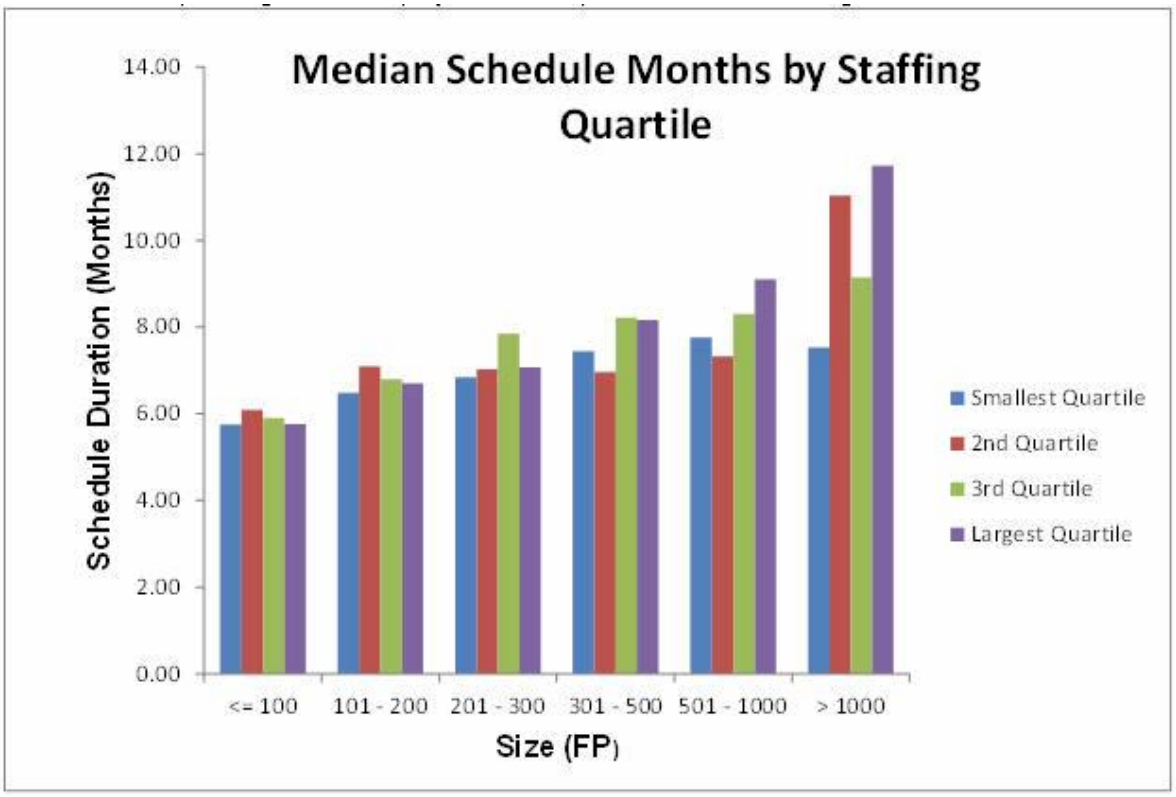


Figure 1. Study of software project sizes and duration against staffing.

The above graphic (Figure 1) is based on a study of over 2,200 completed software projects. In it, the vertical bars represent the median schedule for the smallest to largest staffing quartiles for various sized projects. Note that larger teams do not succeed in reducing schedule even though they normally represent an effort to accomplish exactly that.

Law 5: Allow sufficient time and effort for analysis and design.

At the beginning of any new project, the books and schedule are clean, and the staff is fresh. Everyone is ready to get started on what is sure to be the most efficient, best-quality piece of software the company has ever delivered. The inclination, especially when faced with an aggressive production schedule, is to dive right in and start coding. That is a mistake.

In fact, on average, executives and teams who spend more time and effort on analysis and design end up spending less overall. They also complete projects sooner, have fewer product defects, and are more productive as teams. The old adage "Measure twice, cut once" proves true. Take sufficient time before coding begins to design and analyze the products being built. How much time is sufficient depends on the team and the project. Consider spending roughly 20 percent of any given project in analysis and design for maximum payoff.

This fifth law — allow sufficient time and effort for analysis and design — will help lead to software development success, but executives cannot rely on it, or any one of the laws, alone. Having

all the time in the world for analysis will not help a project if that project is subjected to an aggressive schedule with no time built in for growth.

Executives who are mindful of these five laws will have a much easier time with product builds, and those who neglect these laws could face a bumpy road toward project completion. With appropriately sized teams, schedules built with inevitable growth in mind, and strategies to better manage scope creep, teams that heed the laws will be prepared when high-level plans from analysis and design change as they are executed. The highest-functioning teams will keep data at every step in the development process so as to better inform their estimates for the next software build. At this point the five laws will come into play again, creating a virtuous — not vicious — cycle of software development.

Table 1. Comparison study of projects allocating more than 20 percent effort to analysis and design and those that did not.

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%

The above table (Table 1) summarizes the results from a recent study that compares projects that allocated more than 20 percent of their effort to analysis and design to those that did not.

Top-Down Estimation Can Drive Efficient and Boundaryless Software Development

Lawrence Putnam, Jr.

In 1990, former General Electric CEO Jack Welch wrote a prophetic passage in the company's annual report. "Our dream for the 1990s is a boundaryless company…where we knock down the walls that separate us from each other on the inside."

Ever since, large enterprises have attempted to live by Welch's dream, yet the reality of the "boundaryless" organization remains tantalizingly out of reach for software development companies. They remain hampered by set hierarchies: development teams and product owners exist on one level, business management and system engineers on another, while enterprise architects and portfolio managers reside atop the organizational food chain.

These hierarchies impede efforts to achieve the three Vs of corporate success – vision, value, and velocity. *Vision* defines organizational goals, and development teams must deliver *value* that aligns with that vision. To do so, they must be able to accurately plan and estimate *velocity* – the amount of work completed during a development iteration – and gauge factors that could impact the speed of that work.

Employing a top-down estimation approach to project management can help organizations overcome boundaries and satisfy the three Vs. Let's take a closer look at how this approach can work for software companies, particularly larger organizations, to help them improve project management, team collaboration, and development practices.

Top-down estimation is a preferable alternative to traditional bottom-up estimation processes. Bottom-up estimation calls for every team member to estimate how many hours it will take to

complete a project. The project manager then takes all of this information and compiles it to get a sense of overall total hours for project completion. Unfortunately, much of this is guesswork and opinion and can lead to unrealistic estimations.

A top-down approach can be much more effective and accurate. Rather than relying on the input of individuals, top-down estimation takes a look at the scope of the entire project, overall efficiency of the organization, and complexity of the proposed work – before any actual work is done. It provides valuable data that companies can feed into any large-scale agile framework they may be using, including SAFe, DAD, or other methodologies.

Top-down estimation is grounded in planning and forecasting the time and effort it will take to complete epics (large bodies of work) from the outset. Through this process, managers can break epics up into stories – smaller chunks of work – product features, capabilities, and other more manageable, bite-sized pieces. Then, they can estimate the appropriate number of resources and skillsets required to successfully complete the entire project.

Top-down estimation also helps teams leap across boundaries by providing everyone with a holistic view of work being done across the organization. Teams are provided with comprehensive insight into all of the connections and interactions involved in producing consumable products. Thus, top-down estimation supports the theory that the value of a system is only as good as its various connection points.

Those connections are comprised of staff working at different hierarchical levels, and top-down estimation can keep those people on course and in alignment with organizational requirements. For instance, product managers may have their own ideas regarding staff and spending rates, but they may not be commensurate with those of the company as a whole. Estimation allows those people to see if their plans exceed their organization's current and near future capacity. If they do, teams can make adjustments to bring them more in-line with reality by changing start dates, tweaking their scope, and other simple practices.

By providing a holistic view of software projects and the teams working on them, top-down estimation helps organizations build better systems and software and achieve the three Vs. More than that, it can help organizations achieve Welch's long-sought goal of fewer boundaries and better production.

AI and Automation Make Software Reliability More Important than Ever

Lawrence Putnam, Jr.

If you were thinking about purchasing a driverless car and the salesperson told you that there is a "slight" chance that the car will fail during transit, would you still feel comfortable laying down your money? Or, if you faced an emergency, would you trust an automated robot to perform open-heart surgery, rather than the hands of a skilled physician?

While these questions might seem like the stuff of a science fiction novel, they're quickly becoming a part of our normal, everyday world. We're hearing a great deal about artificial intelligence and how it is replacing tasks that were once done by humans. AI is powered by software, and that software is becoming increasingly vital to our lives. This makes ensuring its reliability more important than ever.

But here's a sobering thought: right now, IT operations teams are building software that is, on average, 95% reliable out the door. That is right; today, a 5% unreliability gap is considered "good enough."

That may be true when you are dealing with traditional web and mobile applications, or some other piece of technology that doesn't necessarily have a significant impact on a person's life or business. But when you scale that software up – for example, to SpaceX testing a rocket, NASA doing a space shot, or even something more Earthbound, like a driverless car – and take out the human factor, that software must be incredibly reliable and leave very little room for error.

It is not just lives and businesses that are at stake, either; our country's security is also dependent on software reliability. Most federal agencies have begun to deploy software automation solutions that relieve IT managers of much of the burden involved in day-to-day network management and security. But what if that software is only 95% reliable? What if a savvy hacker finds his or her way into that 5% window?

If we cannot close that window completely, we need to at least get the software that is rolling out the door to 99% reliability – minimum.

Cutting Corners Is Not an Option

I acknowledge that is not an easy proposition, given certain constraints. For one, organizations in all industries are under enormous pressure to introduce solutions in extraordinarily tight timeframes. Many are also under incredibly tight budgets, compelling them to aim for that 95% "good enough" threshold and not necessarily any higher.

But software reliability is not something that organizations can afford to cut corners on. They need to find ways to work within those constraints while still focusing on creating rock-solid products.

Throwing tons of people at the problem is not the solution. In fact, increasing headcount unnecessarily can be detrimental to the project and create more problems than it solves. We have all heard the term "too many cooks in the kitchen spoil the broth," and that's true – too many people trying to fix a software problem can actually introduce new vulnerabilities and actually end up making the software less reliable. Adding to the team may also require pulling people off of another project, thereby potentially compromising the reliability of other solutions that are in the pipeline, creating a vicious circle.

Mandated deadlines can also hamper attempts to deliver an exceptional finished product. Yes, deadlines are important – but I think we have all experienced the pain of deadlines that are unnecessarily unrealistic, forcing us to deliver software that has not been fully tested. Projects that are dictated by these types of situations actually run the risk of falling well below 95% reliability.

Estimation = Reliability

This is why the practice of software estimation has become even more important today. Software estimation helps ensure that projects remain on track, on budget, and, above all, completed to satisfaction. By "satisfaction," I am not saying "mostly done right." I'm talking about projects without major defects or vulnerabilities – projects that meet the minimum required reliability.

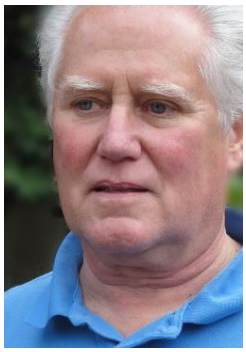
Estimating and planning at the outset of any software development project, even those that employ agile methodologies, can help developers conceptualize the types of resources they will need to successfully build a reliable product. They will be able to gauge the tools and number of people necessary and develop accurate timelines for completion of the project. They will also identify cost drivers (such as team meetings and bug fixes) that will inevitably impact the development process.

All of this helps increase efficiency and build a better – and more reliable – product. That may take a little more time in both development and testing, but it will ultimately save organizations a lot of money and hassle. Most importantly, the final result will be a high quality, dependable product.

INDEX

CONTRIBUTING AUTHORS

Kate Armel, Director of Testing, Training, & Technical Support at QSM, has 19 years of experience providing technical and consultative support for software estimation, project tracking and forecasting, and industry benchmarking. She oversees data collection, validation, and data analysis for the QSM database; development of over 900 industry regression trends; client, internal, and technical support services; software testing and quality assurance; training, documentation and online help for SLIM-Suite and SLIM-Collaborate applications, APIs, and utilities; and technical writing, research, and analysis to support product development, research, and consulting services. Ms. Armel was the Chief Editor and analyst/co-author of the 2006 QSM IT Software Almanac and has authored several published articles.



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 30 years of experience working in customer service, with 21 of those years spent at QSM. As Vice President, his primary responsibilities include supporting the company client base with its software measurement goals, managing business development, and managing existing client relations. He has developed and directed the implementation of the sales and customer retention process 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.

Victoriano Fuster III is a Director at QSM with more than 19 years of technical project and program management experience across government and commercial sectors, including ten years of specialized software/IT estimation experience using the SLIM tool suite. He leads large and small engagements supporting diverse organizations, including the Office of the Secretary of Defense, U.S. Army, U.S. Air Force, and other defense and commercial clients. He has served as a technical expert and curriculum developer in the areas of software/IT development and sustainment costing, multi-disciplinary security analysis, and critical technology protection, and is also Course Director for QSM's government software cost estimation courses. He holds a master's degree, is a PMP-certified project manager, and is a U.S. Marine Corps veteran.

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 in both the government and commercial sectors, Taylor has presented research at national conferences and has authored numerous publications in agile development, ERP development, software estimation, and process improvement.

John A. Staiger, Jr., worked with SLIM tools since 1985, with extensive experience in data analysis and parametric methods. A retired QSM principal consultant to both government and commercial clients, he was also 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, and a master's degree in project management from The George Washington University. He is also a Distinguished Graduate of the Navy War College and retired Navy aviator. He now enjoys traveling around the world with his lovely wife.

Laura Zuber has 25 years of experience in software development consulting, training, and support. She has conducted training and coaching sessions for all QSM SLIM suite of tools and has helped customers implement SLIM across a wide variety of processes and platforms. She has managed software development projects, served as a senior software process improvement specialist, performed process assessments, designed and implemented best practices, and authored numerous training programs. She is a Certified Scrum Master and SAFe Agilist.