History is the Key to Estimation Success

By Kate Armel, QSM

> *There are known unknowns…*
> *But there are also unknown unknowns,*
> *The ones we don't know*
> *We don't know.*
>
> —Secretary of Defense Donald Rumsfeld, February 2002

It was late afternoon in April of 1999 when the phone in my office rang. The conversation went something like this:

"This software estimate just landed on my desk and I need to finish it by close of business today to support a fixed price bid."

"What can you tell me about this project?"

"We're rewriting an existing mainframe billing system developed in COBOL. The new system will be written in C++, so it should be much smaller than the old system."

"Great –perhaps we can use the existing system as a rough baseline. How big is it?"

"I don't have that information."

"Will this be a straight rewrite, or will you add new features?"

"Not sure – the requirements are still being fleshed out."

"What about resources? How many people do you have on hand?"

"Not sure – the team size will depend on how much work must be done... which we don't know yet."

"Can we use some completed projects to assess your development capability?"

"Sorry, we don't have any history."

"That's OK –even without detailed information on scope, resources, or productivity we should still be able to produce a rough order of magnitude estimate based on relevant industry data."

"Rough order of magnitude??? My boss will never accept that much risk on a fixed price bid! Isn't there some general rule of thumb we can apply?"

Welcome to the world of software cost estimation where the things we know – the known knowns - are often outweighed by the things we don't know. Numerous estimation methods exist. Scope is described using effort, delivered code volume, features, or function points. Expert judgment, Wideband Delphi, top down, bottom up, parametric and algorithmic models each have their determined champions. But regardless of method, all estimates are vulnerable to risk arising from uncertain inputs, requirements changes, and scope creep. Skilled estimators and better methods can reduce this risk, but they can't eliminate it. Thus, the ability to identify and account for uncertainty remains a vital component of successful risk management.

Estimation Accuracy vs. Estimation Usefulness

How accurate is the average software cost estimate? Industry statistics vary as widely as the estimates they seek to measure. One oft-cited study – the Standish Group's Chaos Report – concludes that only one third of software projects deliver the promised functionality on time and within budget[1]. A later IEEE study[2] noted several gaps in the Standish Group's criteria for estimation accuracy:

> …the [Standish] definitions don't cover all possibilities. For instance, a project that's within budget and time but that has less functionality doesn't fit any category. … The Standish Group's measures … neglect under runs for cost and time and over runs for the amount of functionality.

When studies rely on different definitions of estimation success or failure, we should expect their assessments of estimation accuracy to exhibit considerable variability. The existence of different standards raises an intriguing question: what makes an estimate "accurate"?

Most quality control measures for estimates compare estimated cost/effort, schedule, or scope to their actual (final) values. The problem with this formulation is that "accurate" estimates are an integral part of feasibility decisions made very early in the project lifecycle; long before anything but the most generic information about the system's intended features or use can be known with reasonable certainty. The technologies used to implement the requirements may be unknown and the schedule, team size, required skill mix, and project plan have yet to be determined. As design and coding progress, the list of unknowns grows shorter and decreasing uncertainty about the estimation inputs lowers the risk surrounding the estimated cost, schedule, and scope. Unfortunately, most organizations must make binding commitments before detailed and reliable information about the project is available.

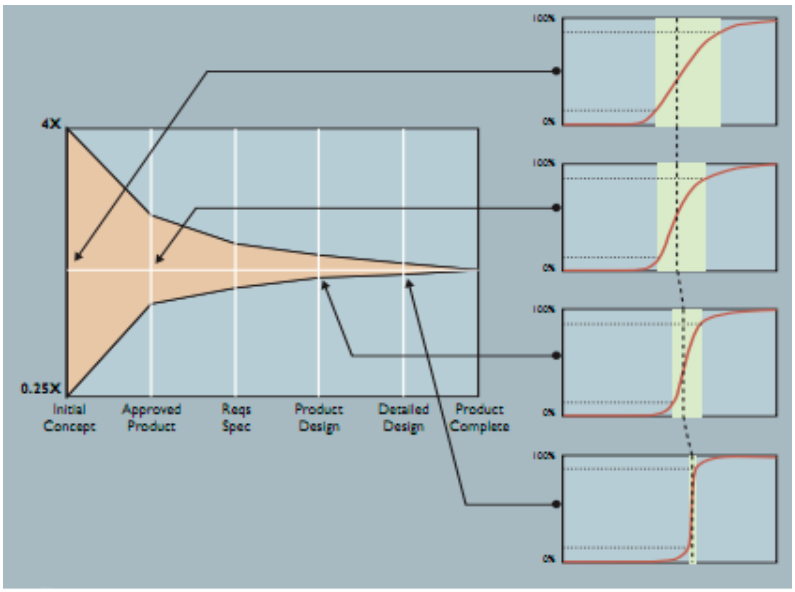
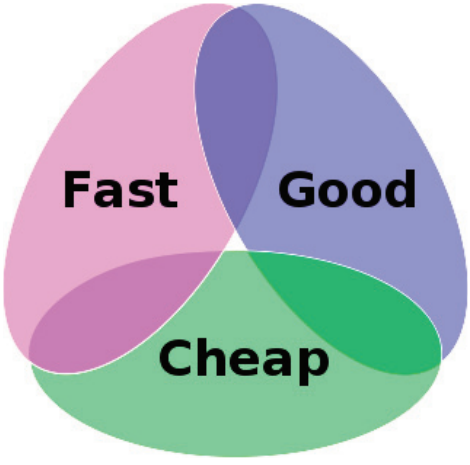


Figure 3. Commitments made early in the project lifecycle must account for greater uncertainty surrounding estimation inputs.

Given the degree of uncertainty surrounding early estimates – and the correspondingly broad range of possible time/effort/scope combinations - estimation *accuracy* may be less important than estimation *usefulness.* In an article for the ACM, Philip Armour explores the difference between these two concepts[3]:

> The commitment is the point along the estimate probability distribution curve where we promise the customer and assign resources. This is what we need to hit, at least most of the time. It is not a technical estimation activity at all but is a risk/return based business activity. It is founded on the information obtained from the estimate, but is not the estimate. Using Figure 3 as an example, if we needed an accurate commitment in the earliest (Initial Concept) phase based on how the diagram shows the project actually worked out, we would have had to commit at around a 75% probability. From the figure, committing to the "expected" result at Initial Concept would have led to a significant overrun beyond that commitment, and the project would have "failed." We can consider the 50% (expected) result to represent the cost of the project and the 25% increment to the higher commitment level to represent the cost of the risk of the project.



Measures of estimation accuracy that treat an estimate as "wrong" or a project as "failed" whenever the final scope, schedule, or cost differ from their estimated values penalize estimators for something outside their control: the uncertainty that comes from incomplete information. We should measure deviations between estimated and actual project outcomes because this information helps us quantify estimation uncertainty and account for it in future estimates. But if measurement becomes a stick used to punish estimators, they will have little incentive to collect and use metrics to improve future estimates.

Understanding and Assessing Tradeoffs

An old project management maxim succinctly summarizes the choices facing software development organizations: *"You can have it fast, cheap, or good. Pick two."* Given that estimates (and therefore, commitments) are made early in the project lifecycle when uncertainty is high and the range of possible solutions is still wide, how do we select plans with a high probability of success? A thorough understanding of management tradeoffs can help. The idea behind the infamous Project Management Triangle is simple but powerful: the tradeoffs between software schedule, effort or cost, and quality are both real and unforgiving. Thanks to the work of pioneers like Fred

Brooks, most software professionals now accept the existence and validity of these tradeoffs but as Brooks himself once ruefully observed, quoting famous maxims is no substitute for managing by them.

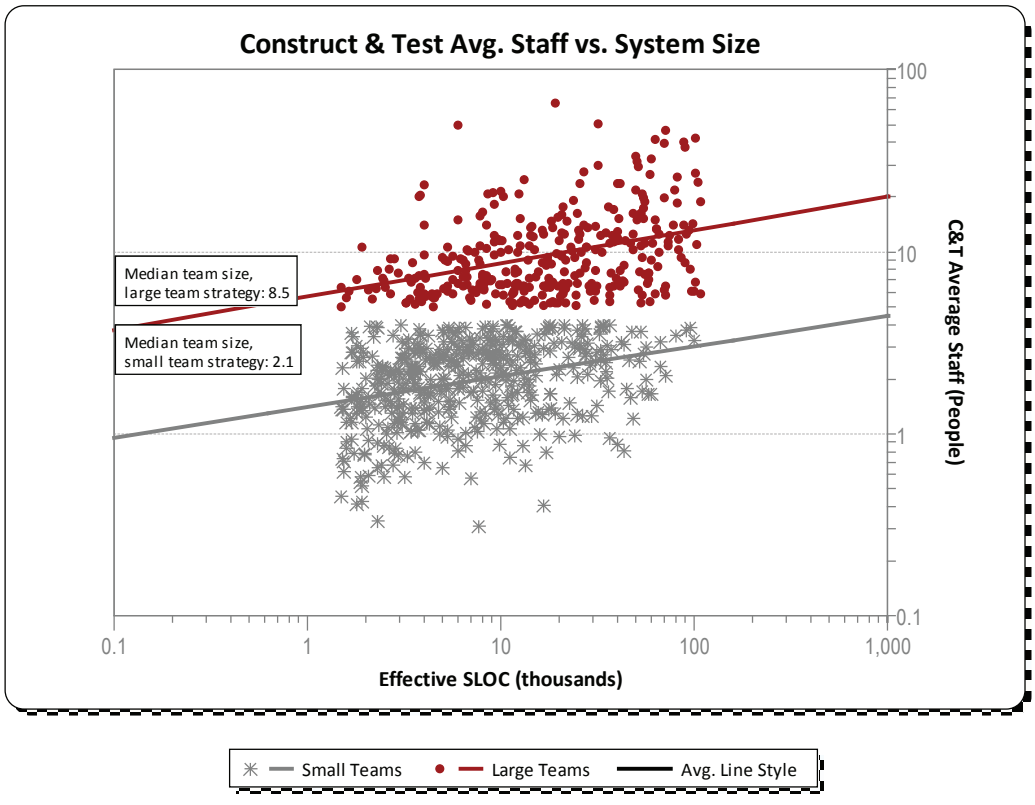
With so many unknowns out there, why don't we make better use of what we do know? Most software "failures" are attributable to the human penchant for unfounded optimism. Under pressure to win business, organizations blithely set aside carefully constructed estimates and ignore sober risk assessments in favor of plans that just happen to match what the company needs to bid to secure new business. Lured by the siren song of the latest tools and methods, it becomes all too easy to elevate future hopes over past experience. This behavior is hardly unique to software development. Recently two economists (Carmen Reinhart and Kenneth Rogoff) cited this tendency to unfounded optimism as one of the primary causes of the 2008 global financial crisis. Their exhaustive study of events leading up to the crash provides powerful evidence that optimism caused both banks and regulators to dismiss centuries-old banking practices. They dubbed this phenomenon the "This Time Is Different" mentality[4]. Citing an extensive database of information gleaned from eight centuries of sovereign financial crises, bank panics, and government defaults, Reinhart and Rogoff illustrate a pattern that should be depressingly familiar to software professionals: without constant reminders of past experiences, our natural optimism bias makes us prone to underestimate risk and overestimate the likelihood of positive outcomes.

The best counter to unfounded optimism is the sobering voice of history, preferably supported by ample empirical evidence. This is where a large historical database can provide valuable perspective on current events. Software development is full of complex, nonlinear tradeoffs between time, effort, and quality. Because these relationships are nonlinear, a 20% reduction in schedule or effort can have vastly different effects at different points along the size spectrum. We know this, but the human mind is poorly equipped to account for non-intuitive exponential relationships on the fly.

Without historical data, estimators must rely on experience or expert judgment when assessing the potential effects of small changes to effort, schedule, or scope on an estimate. They can *guess* what effect such changes might have, but they cannot empirically prove that a change of the same magnitude may be beneficial in one case but disastrous in another. The presence of an empirical baseline removes much of the uncertainty and subjectivity from the evaluation of management metrics, allowing the estimator to leverage tradeoffs and negotiate more achievable (hence, less risky) project outcomes. One of the most powerful of these project levers is staffing. A recent study of projects from the QSM database[5] used 1060 IT projects completed between 2005 and 2011 to show that small changes to a project's team size or schedule dramatically affect the final cost and quality. To demonstrate the power of the time/effort tradeoff, projects were divided into two "staffing bins":

- Projects that used **small teams of 4 or fewer FTE staff**
- Projects that used **large teams of 5 or more FTE staff**.

The size bins span the median team size of 4.6, producing roughly equal samples covering the same size range with no overlap in team size. Median team size was **8.5 for the large**



team projects and **2.1 for the small team projects**, making the ratio of large median to small median staff approximately 4 to 1. The wide range of staffing strategies for projects of the same size is a vivid reminder that team size is highly variable, even for projects of the same size. It stands to reason that managers who add or remove staff from a project need to understand the implications of such decisions.

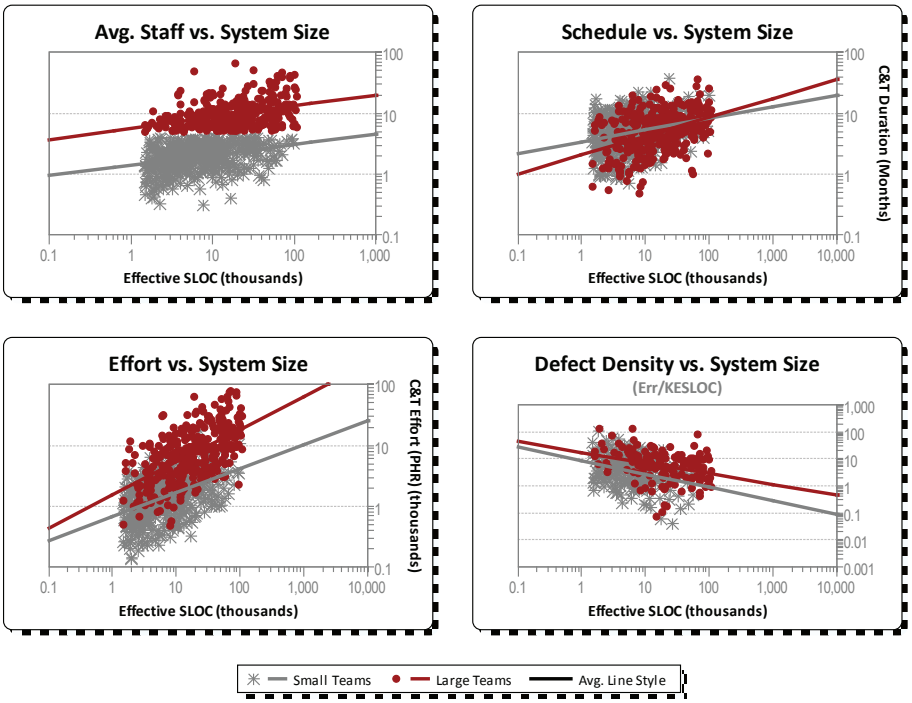
Regression trends were run through each sample to determine the average Construct & Test effort, schedule, and quality at various points along the size axis. For very small projects (defined as 5000 new and modified source lines of code), using large teams was somewhat effective in reducing schedule. The **average reduction was 24%** (slightly over a month), but this improved schedule performance carried a hefty price tag: **project effort/cost tripled** and **defect density more than doubled**.

For larger projects (defined as 50,000 new and modified source lines of code), the **large team strategy shaved only 6% (about 12 days) off the schedule** but **effort/cost quadrupled** and **defect density tripled**.

At 5K ESLOC	Schedule (Months)	Effort (Person Hours)	Defect Density (Defects per K ESLOC)
Small teams	4.6	1260	3.7
Large teams	3.5	4210	9.2
Avg. Difference (Large team strategy)	**-24%**	**334%**	**249%**
At 50K ESLOC			
Small teams	7	3130	1.2
Large teams	6.6	13810	3.9
Avg. Difference (Large team strategy)	**-6%**	**441%**	**325%**

Quantitative Software Management, Inc.

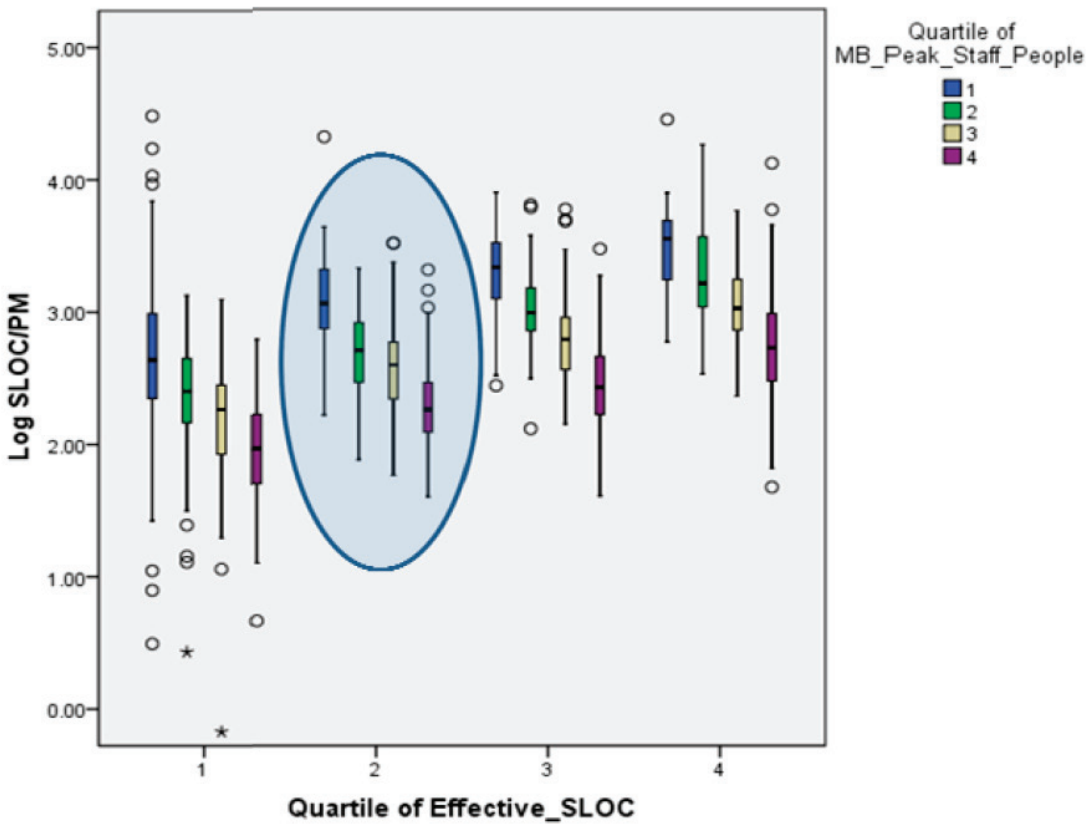
The relative magnitude of tradeoffs between team size and schedule, effort, and quality is easily visible: large teams achieve only modest schedule compression while causing dramatic increases in effort and defect density.



What else can the data tell us about the relationship between team size and other software management metrics? A 2010 study by QSM consultant and metrics analyst Paul Below found an interesting relationship between team size and conventional productivity (defined as effective SLOC per unit of construct and test effort). 6 To make this relationship easier to visualize, Paul stratified a large sample of recently completed IT projects into 4 size quartiles or bins, then broke each size bin into sub-quartiles based on team size. The resulting observations held true across the entire size spectrum:

- In general, productivity **increased with project size**
- *With any given size bin* **productivity decreased as team size went up.**

To see the relationship between average productivity and project size, compare any four staffing quartiles of the same color in the graph below from left to right as size (bottom or horizontal axis) increases:



As the quartiles increase in size (bottom axis), average productivity (expressed as SLOC per Person Month of effort on the left-hand axis) rises. The slope is reversed for projects of the same size (i.e., within a given size quartile). To see this, compare the four differently colored box plots in the second size quartile highlighted in blue. The size and staffing vs. productivity relationships hold true regardless of which Productivity measure is used: SLOC per Person Month, Function Points per Person Month, and QSM's PI (or Productivity Index) all increase as project size goes up but decrease as team size relative to project size increases. The implication that the optimal team size is not independent of project scope should not surprise anyone who has ever worked on a project that was over or under staffed but the ability to demonstrate these intuitively sensible relationships between scope and team size with real data is a valuable negotiation tool.

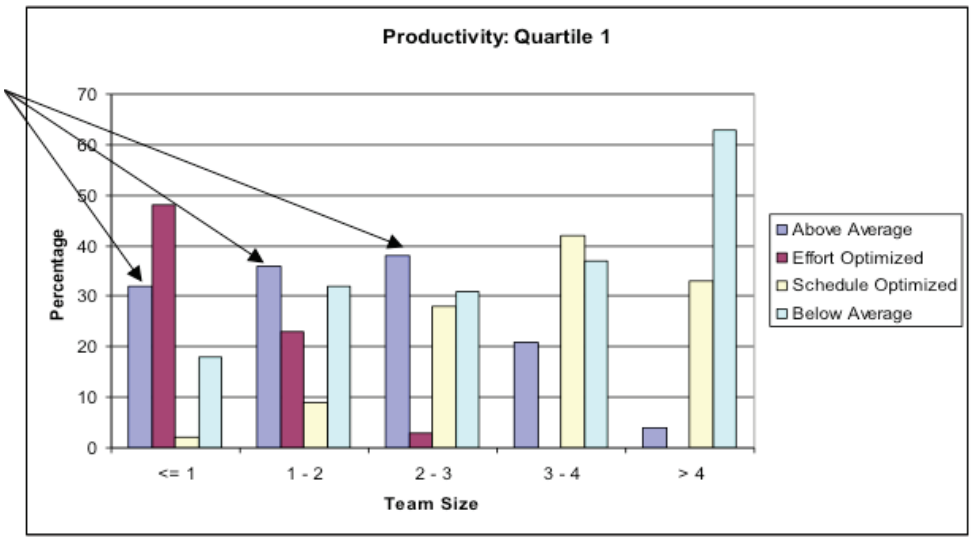
Determining the Optimal Team Size for your Project

If the data suggest that optimal team size is related to project scope, it should be able to help us find the right staffing strategy for projects of various sizes. In a study conducted in the spring of 2011, QSM Consultant Don Beckett decided to explore

the best team size for different project sizes and management goals. He divided 1920 IT projects completed since 2000 from the QSM database into four size bins: less than 4000, 4001 – 9400, 9401-25000, and over 25000 SLOC. For each of these size bins, he determined median effort (SLOC/PM) and median schedule (SLOC/Month) productivity values. Based on the results, he assigned projects to one of four categories:

Better than average for effort & schedule	Worse than average for effort & schedule
Better for effort/worse for schedule	Worse for effort/better for schedule

As the chart below shows, projects in the smallest size quartile (under 4000 SLOC) using teams of **3 or fewer people** (blue bars) were the most likely to achieve balanced schedule and cost/effort performance. Teams of **2 or fewer** (purple) achieved the best cost/effort performance and teams of **2-4** (yellow) delivered the best schedule performance. Teams that used **more than 4 people** achieved dramatically worse cost/effort and schedule performance (green bar). This process was repeated for projects in the next 3 size quartiles and the results were entered into a team size matrix:



Size Bin	Schedule Optimized	Cost/Effort Optimized	Balanced Performance
1 - 4000 ESLOC	2 - 4	2 or fewer	3 or fewer
4000 - 9400 ESLOC	2 - 6	3 or fewer	3 or fewer
9401 - 25000 ESLOC	2 - 4	4 or fewer	2 - 4
Over 25000 ESLOC	4 - 6	5 or fewer	2 - 6
Large Projects > 70000 ESLOC	10 - 20	10 - 20	10 - 20

Quantitative Software Management, Inc.

Don's results confirm the findings from our previous two studies: the maximum optimal team size for cost/effort performance increases steadily with project size. The relationship between schedule performance and team size is less clear, with the optimal team size for balanced schedule and performance falling somewhere in the middle.

Expert Judgement vs. Empiricism

Regardless of which estimation methods are used in your organization, uncertainty and risk cannot be eliminated and should never be ignored. Recognizing and explicitly accounting for the uncertainties inherent in early software estimates is critical to ensure sound commitments and achievable project plans.

Measures of estimation accuracy that penalize estimators for being "wrong" when dealing with uncertain inputs cloud this fundamental truth and create powerful disincentives to honest measurement. Recording the difference between planned and actual outcomes is better suited to quantifying estimation *uncertainty* and feeding that information back into future estimates than it is to measuring estimation accuracy.

So how can development organizations counter optimism bias and deliver estimates that are consistent with their proven ability to deliver software? Collecting and analyzing completed project data is one way to demonstrate both an organization's present capability and the complex relationships between various management metrics. Access to historical data provides empirical support for expert judgments and allows managers to leverage tradeoffs between staffing and cost, quality, schedule and productivity instead of being sandbagged by them.

The ideal historical database will contain your own projects, collected using your organization's data definitions, standards, and methods but if you haven't started collecting your own data, industry data offers another way to leverage the experiences of other software professionals. Industry databases typically exhibit more variability than projects collected within a single organization with uniform standards and data definitions, but QSM's three-plus decades of collecting and analyzing software project metrics have shown that the fundamental relationships between software schedule, effort, size, productivity and reliability unite projects developed and measured over an astonishingly diverse set of methodologies, programming languages, complexity domains and industries.

Software estimators will always have uncertainty to contend with, but having solid data at your fingertips can help you challenge unrealistic expectations, negotiate more effectively, and avoid costly surprises. Effective measurement puts managers in the drivers' seat. It provides the information they need to negotiate achievable schedules based on their proven ability to deliver software, find the optimal team size for new projects, plan for requirements growth, track progress, and make timely mid-course corrections. The best way to avoid a repeat of history is to harness it.

About the Author

Kate Armel is the Director of Research and Technical Support at Quantitative Software Management, Inc. She has 12 years of experience in technical writing and metrics research and analysis and provides technical and consultative support for Fortune 1000 firms in the areas of software estimation, tracking, and benchmarking. Ms. Armel was the chief editor and a researcher and co-author of the QSM Software Almanac. She also manages the QSM database of over 10,000 completed software projects.

Endnotes

i The Standish Group, New Standish Group report shows more project failing and less successful projects. http://www1.standishgroup.com/newsroom/chaos_2009.php, (April 23, 2009).

ii J. Laurenz Eveleens and Chris Verhoef, The Rise and Fall of the Chaos Report Figures, http://www.cs.vu.nl/~x/chaos/chaos.pdf, (January/February 2010).

iii Philip G. Armour, The Inaccurate Conception, http://dl.acm.org/citation.cfm?id=1325558&bnc=1, (March 2008).

iv Carmen Reinhart and Kenneth S. Rogoff, *This Time Is Different: Eight Centuries of Financial Folly*, (New Jersey: Princeton University Press, 2009).

v Kate Armel, An In-Depth Look at the QSM Database, http://www.qsm.com/blog/2011/depth-look-qsm-database, (September, 2011).

vi Paul Below, Part II: Team Size and Productivity, http://www.qsm.com/blog/2010/part-ii-team-size-and-productivity (April, 2010).

Quantitative Software Management, Inc. | 2000 Corporate Ridge | Suite 700 | McLean, VA 22102

we like your feedback

At the DACS we are always pleased to hear from our journal readers. We are very interested in your suggestions, compliments, complaints, or questions. Please visit our website ***http://journal.thedacs.com****, and fill out the survey form. If you provide us with your contact information, we will be able to reach you to answer any questions.*