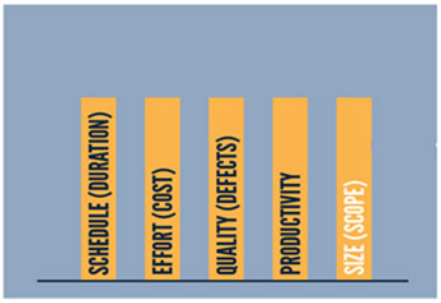
MEASURING SOFTWARE SIZE

Insights from the Past to Guide the Future



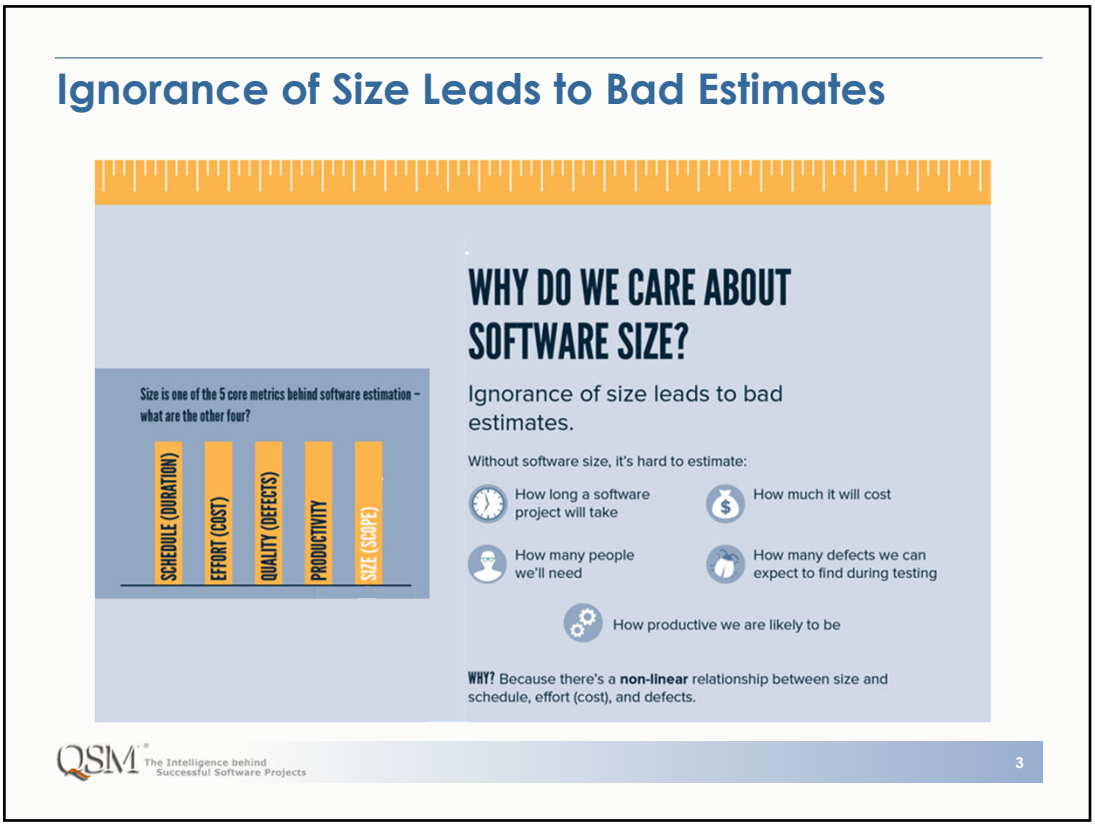
QSM The Intelligence behind Successful Software Projects

Laura Zuber,

Lead Support Representative and Instructor, CSM, SA

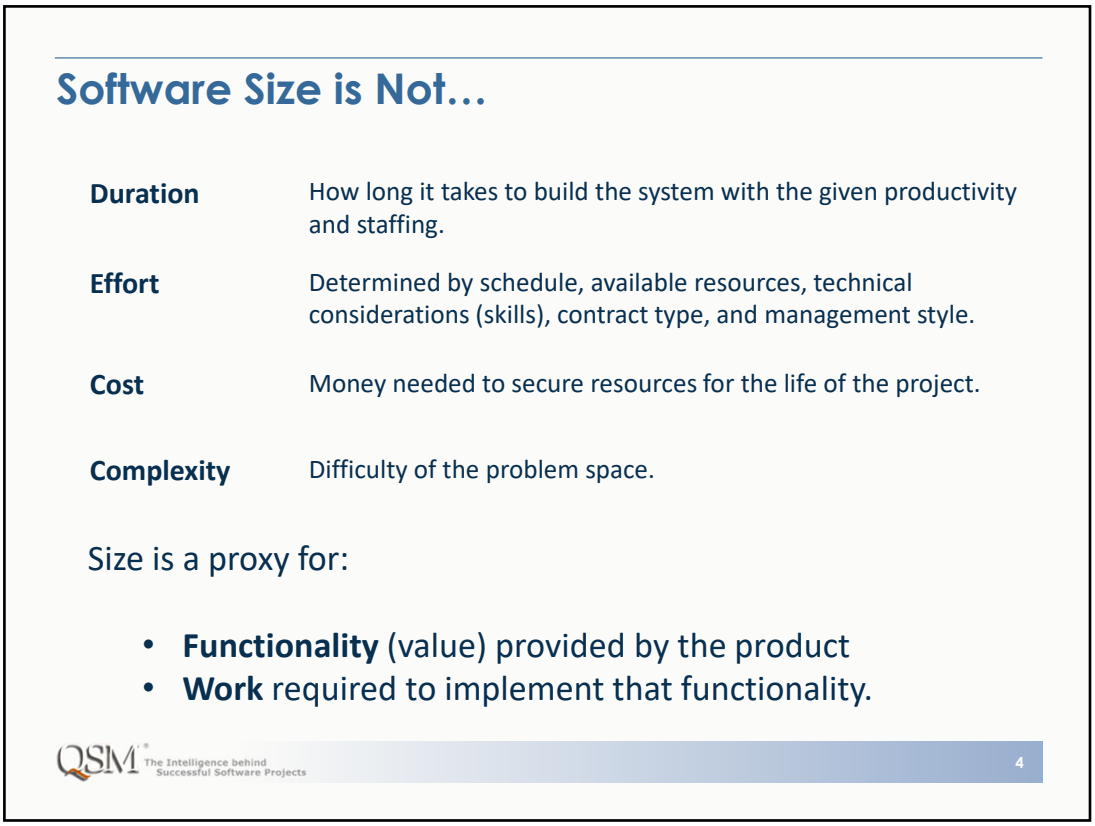
Laura Zuber has 25 years of experience in software development consulting, training, and support. She has conducted training and coaching sessions for all QSM SLIM-Suite tools and helped customers implement SLIM across a wide variety of processes and platforms. Laura has managed software development 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.

QSM The Intelligence behind Successful Software Projects



Why do we care about software size? Because ignoring size leads to bad estimates.

We know from QSM's founder Larry Putnam, Sr.'s original work, and 40 years of data collection and analysis, that Schedule, Effort, Defects, and PI all increase with Size. And the relationship is nonlinear.



Schedule, effort, cost, and complexity are *influenced* by size. These measures don't help us understand software project dynamics in order to estimate and them well.

It is especially challenging for folks used to bottom up estimating NOT to specify effort as a measure of project size. We'll see soon that time and effort are tradeoffs, so how much effort is required is driven by the schedule (and vice versa).

We want size to measure the product, the deliverable. We want to capture what is being built, similar to sq ft of a home.

Humans Measure Everything

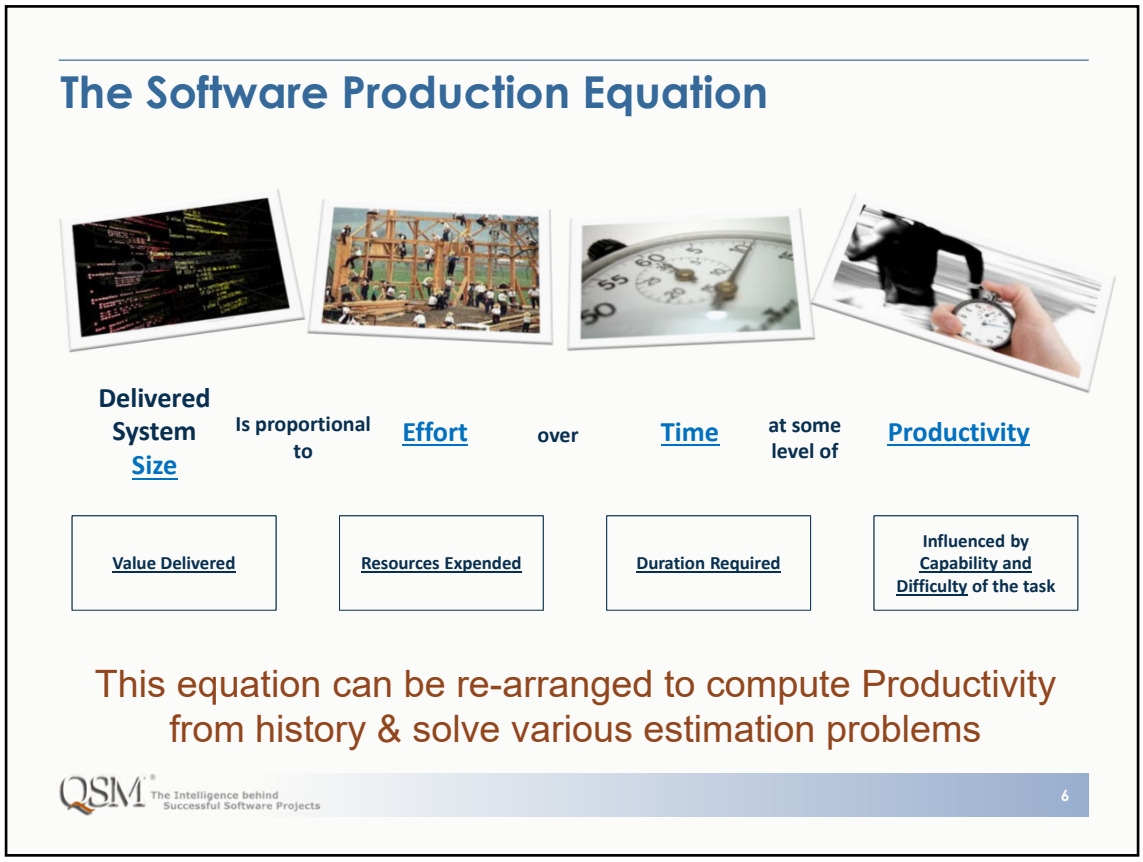
QSM The Intelligence behind Successful Software Projects

Measurement is vital to everyday life. One of the things humans measure is time. Not only do we track time, we are obsessed about it.

Other things we measure: distance, changes in the stock market, blood pressure, weight, and rhythm.

Who can tell me what the image in the lower left is measuring?

So why is measuring software size so hard?



Before we dive into the challenges of sizing software, I want to explain some basic concepts.

The software production equation is at the heart of SLIM tools and research (and Larry Putnam's original work).

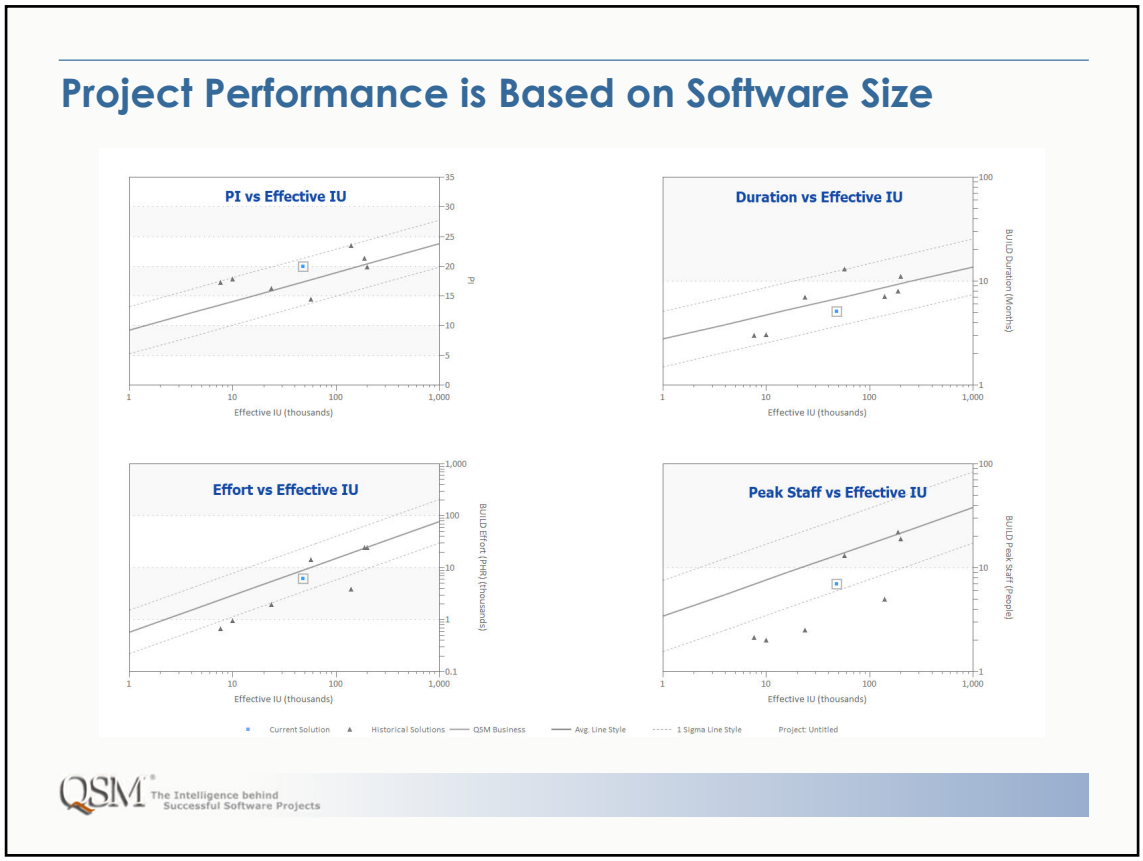
For straightforward estimates, inputs are Size and Productivity.

Outputs are Time and Effort

Equation can be re-arranged to solve for unknowns.

Time and Effort trade off against each other. A project with given size and productivity can be developed in less time with more effort, or more time with less effort – driven by business decisions.

This time/effort tradeoff can be measured and modeled empirically. It is non-linear (and thus, hard for humans to predict).



This view shows 4 core metrics plotted against software size – size is the horizontal axis. Except for PI, SLIM's Productivity Index, the axes scales are exponential, so the straight trend lines are actually curved lines on a linear scale. Small changes in size have a dramatic effect.

We should be motivated to measure size. We know that the core metrics (time, effort, defects, productivity) all increase with product size, and can be predicted from size using trend lines.

It is intuitive that this is so, but many organizations are not collecting and analyzing their own data.

Agile teams are measuring velocity, usually in story points/sprint and/or tasks/sprint. I have not encountered many customers measuring at a higher organizational level or longer release time frames. They want to estimate at the higher levels, but they're not measuring completed project. Yet this is the data we need to estimate releases, value delivered, strategic business initiative, or to respond to RFPs – the business still wants predictability.

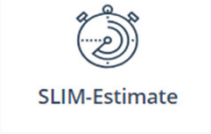
A recent buzz word/phrase is Value Stream Management – have to measure value = size can be one of those measures.

Past, Present, and Future

- Size Completed Project

- Size In-Flight Product Production

- Size Estimates



QSM The Intelligence behind Successful Software Projects

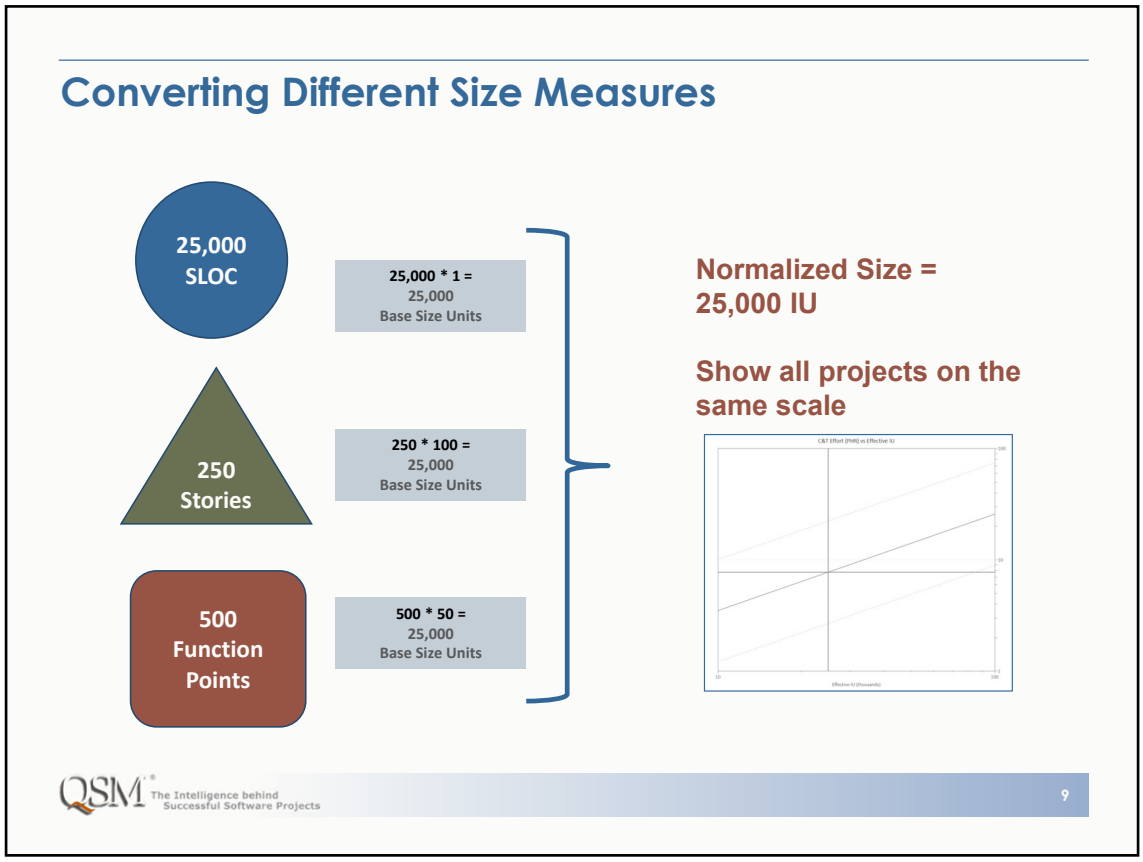
I titled this presentation Insights From the Past to Guide the Future, because sound software sizing methods start with completed projects.

Measuring size is critical to all stages of the life cycle and serves slightly different purposes at various points in time.

Examples (reverse order):

- Estimation – use to validate schedule, cost, resource plans (6 mos may be entirely reasonable for a 20K system but clearly not for a 150K system)
- Tracking – use to track actual code production against plan, get early warning of scope creep/size underestimation, assess impact of changes to plan
- Closeout – use to calculate actual PI, benchmark defects found, calculate gearing factors to support future estimates

The past is the place to start, because working through the data gathering process forces you to define the measures that work for you and establish relationships between different size units and between size and other core metrics.



One more key concept – we need to be able to work with multiple size measures/units.

SLIM tools use the notion of a "Base size unit" (the smallest identifiable unit of software work, roughly equivalent to writing a line of code) as a "least common denominator" that allows us to meaningfully compare size estimates and actuals measured in different size units. We call this an Implementation Unit (IU)

We can express software size as a SLOC, Story, or Function Point count, but without some way to scale a IU to a story or Function Point, our size measurements cannot be displayed on the same charts or compared to one another.

A non-software example would be how useful it is to know how many calories there are in piece of pie vs. a piece of fruit. To make decisions, we need a way to normalize size measurements to a common scale of reference. We call this a Gearing Factor – Here, we convert SLOC, Stories, FP to IU.

Software Sizing Challenges

1. Unit of Measure
   - Requirements, Stories, Features, RICEFW, ……
2. Gearing Factors
   - IU/Story, IU/Function Point, ……
3. Complexity/Size Rating
   - Simple, Average, Complex, ……

RICE Object	Complexity Definition
Reports	**Low:-** Less than five standard application tables. As many as one external file. Straightforward data retrieval. Logic: Basic, single-level report. Little aggregation or sorting. No use of external subroutines. One version suits all requirements.
	Medium: Between five and eight standard application tables. As many as three external files. Some cross-checking. Logic: Multiple-level drill down capability. Moderate calculation, sorting. Some customization (i.e. company-wide). Field translations required.
	High: Nine to ten standard application tables. Three to four external files. Data from multiple functional areas. Logic: Use of sub-screens, pop-ups, etc. Significant authorization checking. Complicated data retrieval. Some customization (i.e. plant-wide). Field translations required.

QSM The Intelligence behind Successful Software Projects

There are 3 min challenges organizations face, either just starting out or as new technologies are introduced.

Unit of Measure

- Determined by
  - Product: SLOC, RICEF, Modules, Objects, ETLs,
  - Process: Epics, Story Points, Use Cases, Requirements
  - People: FP, all*
  - Tools: Requests, Features, Tasks
  - Where you are in the life cycle

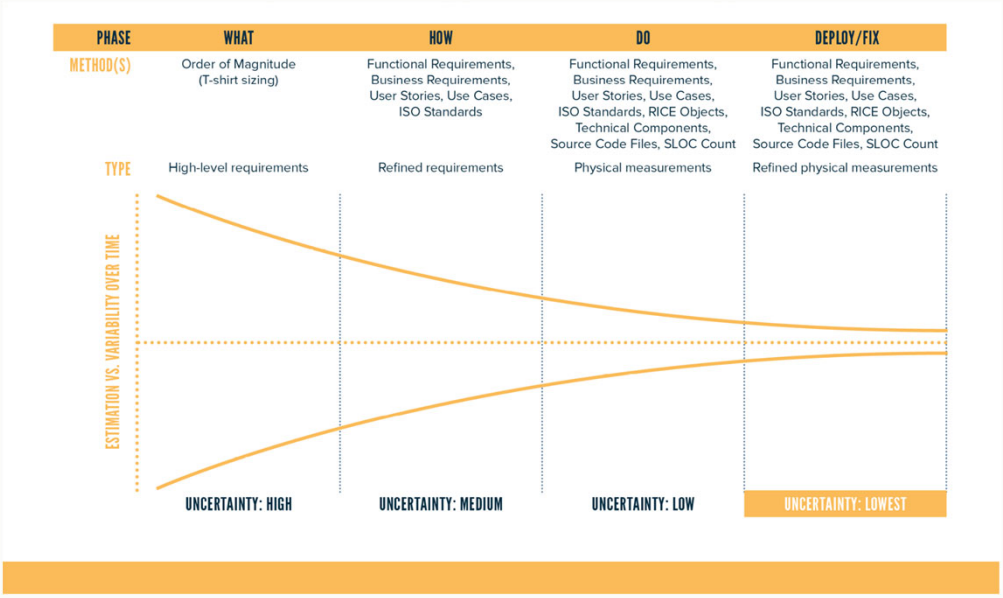
"Story Points have nothing to do with Agile."
Steve McConnell

QSM The Intelligence behind Successful Software Projects

Challenge #1 – Selecting an appropriate unit of measure.

These are some of the factors that will drive your choices:

What Kind of "Ruler" Can You Use?



QSM The Intelligence behind Successful Software Projects

Because some size measures are more detailed than others, more units of measure become available as the project progresses through the life cycle.

If you have not considered using multiple units, I would encourage you to do so, because you not only increase your understanding of what's happening by looking at it from different perspectives, but it provides you with more options for estimating software size across a variety of project types.

Early in the life cycle we have to deal with higher degrees of abstraction - T-shirt sizes and requirement counts. As requirements and design activities are completed, smaller chunks of functionality are documented as user stories, features, or tangible components such as reports or screens.

Of course, when the release is complete we should have the ability to get a code count or file count, or some representation of what has been constructed.

Whatever units you are using for a particular release, you want to capture actuals when development is complete.

Start With History

- Why does QSM have some **gearing factors** to help you get started?

QSM Database

Consulting & Support

QSM The Intelligence behind Successful Software Projects

Challenge #2 –Determining Gearing Factors

When I teach sizing in the SLIM-Estimate class, I tell learners not to fear – QSM has gearing factors for a large number of size units to help them get started.

SLIM has different set of sizing units and associated gearing built in.

We have these gearing factors because:

- Clients provided data on multiple sizing units for projects submitted to the QSM database
- We calculated them by working with clients in consulting and support

Example: ETLs for data warehousing

You can use these GFs as-is but the best practice is to compute your own gearing factors from your completed projects. I am hoping that many of you have suggestions or tips to share. I have a couple of examples to present to describe the basic steps so you can be thinking how to apply this approach on your projects.

Identify Tangible Items – Code & Artifacts

- Can you get a SLOC count?
  - What's included
  - What's not included
- What tracking systems do you use?
  - VersionOne
  - Jira
  - Other
- What supporting process documents to you have?
  - Requirements
  - User Stories

Where is all of this located and who has access to it – who do you need to work with?

QSM The Intelligence behind Successful Software Projects

The first step is to identify tangible items you can count. This is where actual data from completed projects or in-progress projects comes in.

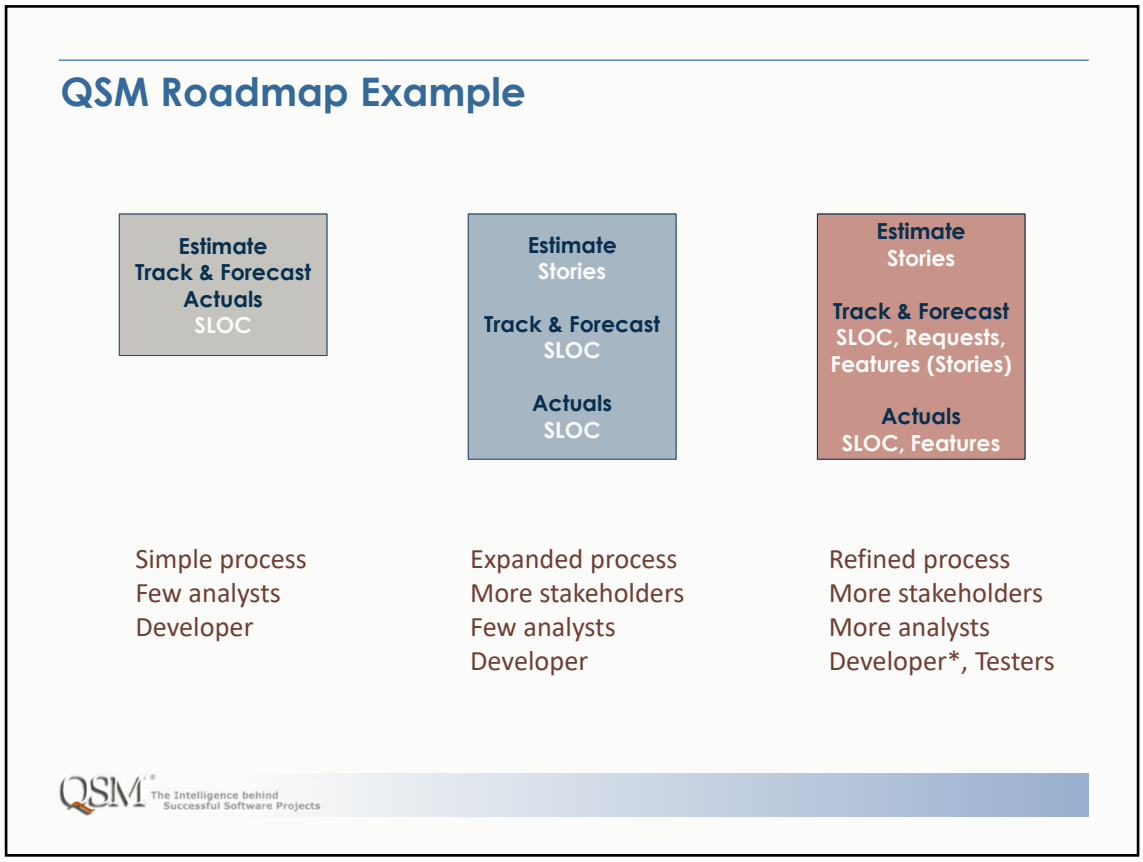
We are talking about completed projects, so there should be something you can identify.

Bullet 1 – Yes, a SLOC count is still the best, along with other function units. If you can get a code count you need to know what is in it, how it was created. I'll show you an example in a moment. But this is good because it can be automated.

Bullet 2 – Take a look at the tracking tools your organization uses and find out what data you can glean from them. The data available is dependent upon how the tool was configured. Most likely you can make changes to set up new metrics to derive more useful information. I will show an example of this also.

Bullet 3 – A third place to look is at your development process artifacts outside of the tracking tools – documents, state of work.

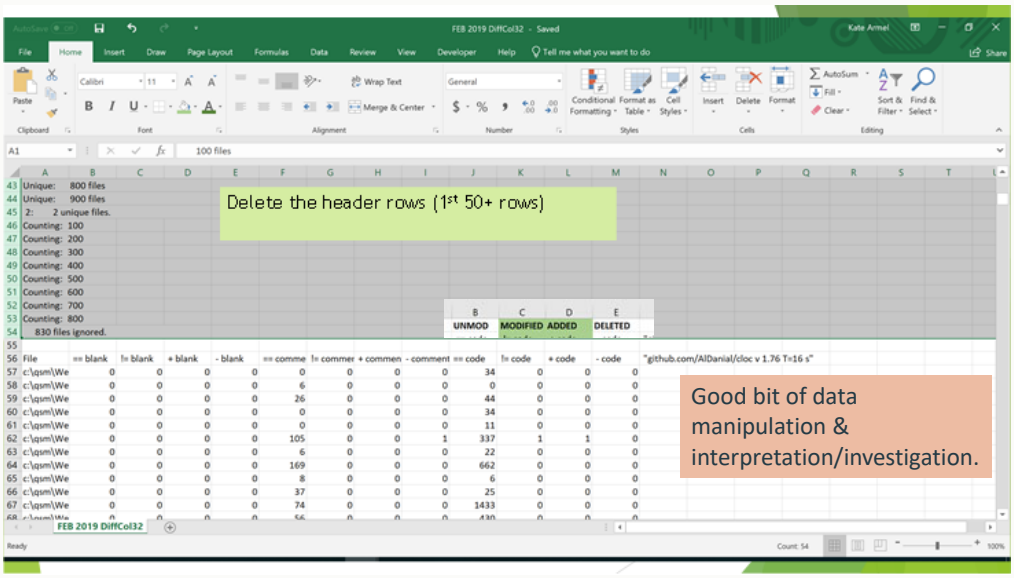
Lastly, I know that a challenge many of you face is that you don't have access to the data – it's there but no one is sharing. This is a business process issue that management needs to address. If management or customers want better estimates they need to support the collection and analysis of completed projects. There is, of course, a diplomatic way to go about this. There has to be consensus regarding what is beneficial or required and make it happen.



The QSM Roadmap is a report of our development projects at various stages of the lifecycle, updated monthly. The estimation and tracking process, how it has changed over time and the size units it uses is a good example – simple, and something I have just recently learned more about.

Early on and for years, we only used SLOC for our size units, but refining the process and involving more stakeholders (nondevelopers), we included other units – taken from the tracking tool and customized.

What's In The Code Count?



QSM The Intelligence behind Successful Software Projects

QSM uses actual code counts. You will see an alternative approach in a few slides.

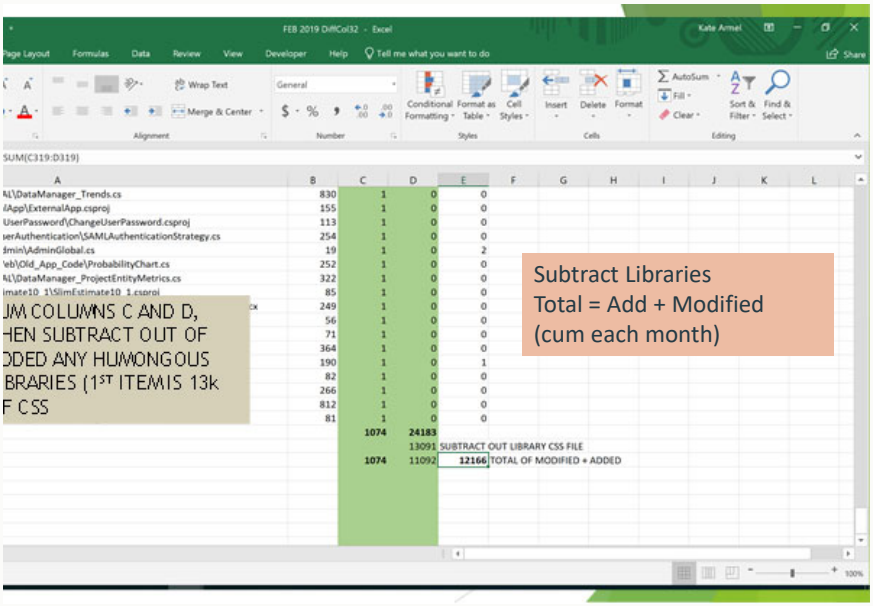
Script is run against the configuration management system to get cumulative code on a monthly basis

Raw data is to- much and not really in terminology that makes sense to non-developers like me, so we do a little data manipulation.

Rows del

Cols del and renamed – this has to be done every time, but its not a big deal (could improve)

What's In The Code Count?



QSM The Intelligence behind Successful Software Projects

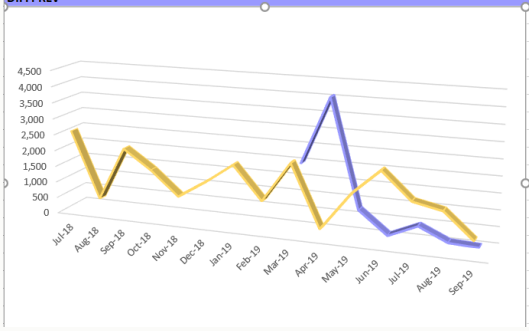
Interpretation – What's included?

Collaborate uses some libraries for web page style sheets, fonts, graphics, etc., so these need to be removed.

How do you know what to expect, what is reasonable? I will show you one way to identify anomalies on the next slide, but you have a conversation with the development team!

Recognize Patterns – Does it make sense?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
3.2 CODE COUNTS																
		Jul-18	Aug-18	Sep-18	Oct-18	Nov-18	Dec-18	Jan-19	Feb-19	Mar-19	Apr-19	May-19	Jun-19	Jul-19	Aug-19	Sep-19
SC ENTRY		2,600	3,126	5,311	6,910	7,768	9,164	11,171	12166	14396	14731	16193	18448	19895	21158	21644
DIFFPREV		**2,600**	**526**	**2,185**	**1,599**	**858**	**1,396**	**2,007**	**995**	**2,230**	**335**	**1,462**	**2,255**	**1,447**	**1,263**	**486**
10.2 CODE COUNTS																
SC ENTRY										1,995	6,048	6,791	6,844	7,290	7,334	7,336
DIFFPREV										**1,995**	**4,053**	**743**	**53**	**446**	**44**	**2**



QSM The Intelligence behind Successful Software Projects

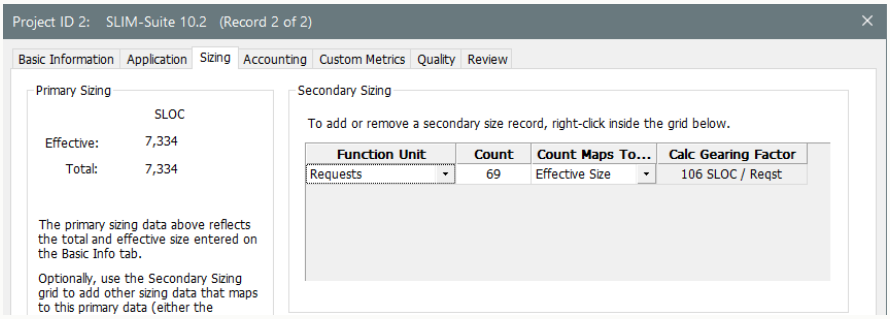
The code count we get is cumulative because that's the easiest. This step can be used with any size unit.

We started computing the monthly production rate by calculating the difference in cumulative values. We can identify values that seem out of whack with the normal production rate for each project. That's a signal to find out if we made a data manipulation error, the script includes something new, or what the cause may be. Data validation checks are necessary.

NOTE THE WIDE VARIATION IN MONTHLY CODING RATES! CONTEXT IS IMPORTANT. VARIATION REFLECTS:

- STAFFING
- MULTI-TASKING (TIME SLICING BETWEEN PROJECTS)
- FEATURE COMPLEXITY
- PARTIALLY FINISHED WORK (MAY NOT CHECK IN UNTIL "DONE-DONE")
- WHERE YOU ARE IN LIFECYCLE (TAIL OFF AT END)
- SCOPE CREEP (LAST MINUTE ADDITIONS)

DataManager Calculates Gearing Factors

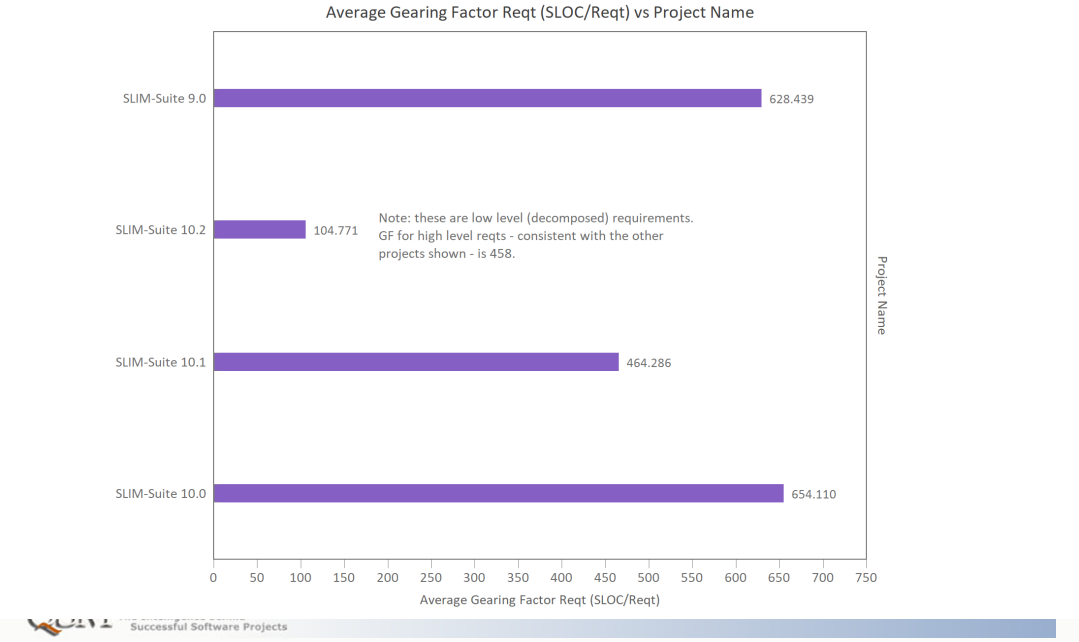


QSM The Intelligence behind Successful Software Projects

Once the project is over, your final code count (you may have to determine/define what is "done done.") you enter that as your Primary Size Unit in DataManager, and for every other size unit you can count, DataManager will compute and store the gearing factor.

These Function Units must measure the entire system, as does the Primary.

Effective Size & Gearing Factors by Project



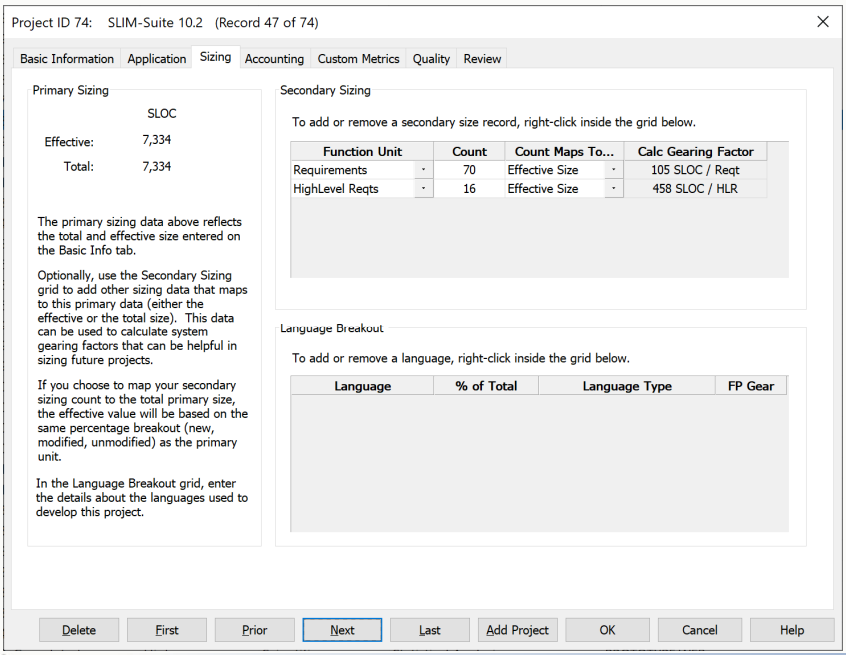
Different projects may have different gearing factors for the same software unit, depending upon the rigor, formality, and consistency of your processes.

This chart shows recent SLIM-Suite releases.

- 9.0 & 10.0 are very consistent – on average, each requirement resulted a big change, more work.
- The smaller releases were close to each other.

It doesn't take long for you to recognize patterns for your work.

Effective Size & Gearing Factors by Project



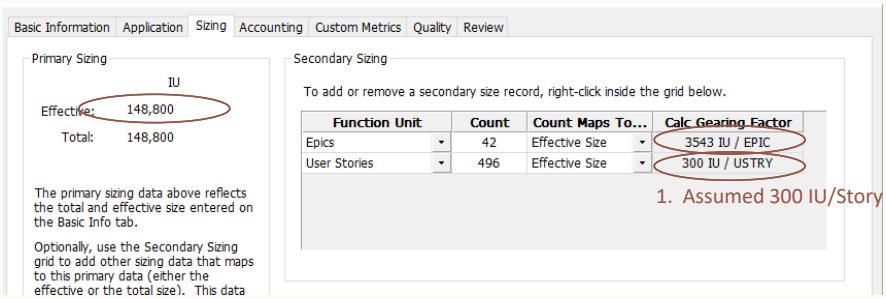
This example supports the concept presented on the previous slide. We had counts for High Level Requirements and Requirements. This allows the calculation of two GFs that can be used based on the information available at different estimation points or for project tracking.

What If You Can't Get Code Counts?

- Use the size unit data you have
- Find a component that is similar for which gearing factors are available
- Use relative sizes of other components/units you're using

2. Convert # USTRY to IU

3. Enter # Epics to compute GF IU/EPIC



1. Assumed 300 IU/Story

QSM The Intelligence behind Successful Software Projects

You say, "Laura, that's really awesome, but there's no way I can get a code count!"

Okay, here's another approach that works.

Accounting for Complexity

Simple Average Complex

QSM The Intelligence behind Successful Software Projects

We know that not all requirements, features, user stories … are created equal. Separate from the level of decomposition, size is used to account for complexity – using different GFs.

Assigning components to complexity bins decreases the uncertainty of our total size estimate; more confidence we have capture the big chunks.

Selecting a Complexity/Size Bin

It's not Magic! There is simple process

QSM The Intelligence behind Successful Software Projects

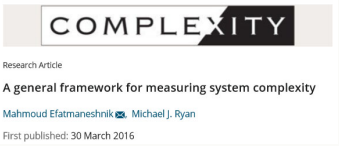
Since we don't have a Sorting Hat to select the right complexity bin for us, we need to define some guidelines.

This is what many organizations as for. There are no guidelines that apply to everyone because, other than FPs, software size units don't have standards.

Size – A Measure of Complexity

"…complexity of any particular matter or thing has a significant subjective component in which the degree of complexity depends on available frames of reference. Any attempt to remove subjectivity from a suitable measure therefore fails to address a very significant aspect of complexity. Conversely, there has been justifiable apprehension toward purely subjective complexity measures, simply because they are not verifiable if the frame of reference being applied is in itself both complex and subjective. We address this issue by introducing the concept of subjective simplicity – although a justifiable and verifiable value of subjective complexity may be difficult to assign directly, it is possible to identify in a given context what is "simple" and, from that reference, determine subjective complexity as a distance from simple."

Reference Rock



QSM The Intelligence behind Successful Software Projects

Every organization is slightly different and developing diverse sets of applications on different platforms.

I wanted to see what I could find to guide, you I did a Google search and stumbled upon this article from Complexity scientific publication – who knew there was such a thing!

This gist is that we acknowledge there is an element of subjectivity we must deal with, however (read last green highlighted text).

Sizing Grounding (Epics)

- Don't assign points until after T-shirt sizing is finished
- Smallest T-shirt size > largest Story
- Select size everyone agrees is Medium; something well understood
- Create brackets for XS and XL; the rest of the sizes can be defined as you start estimating

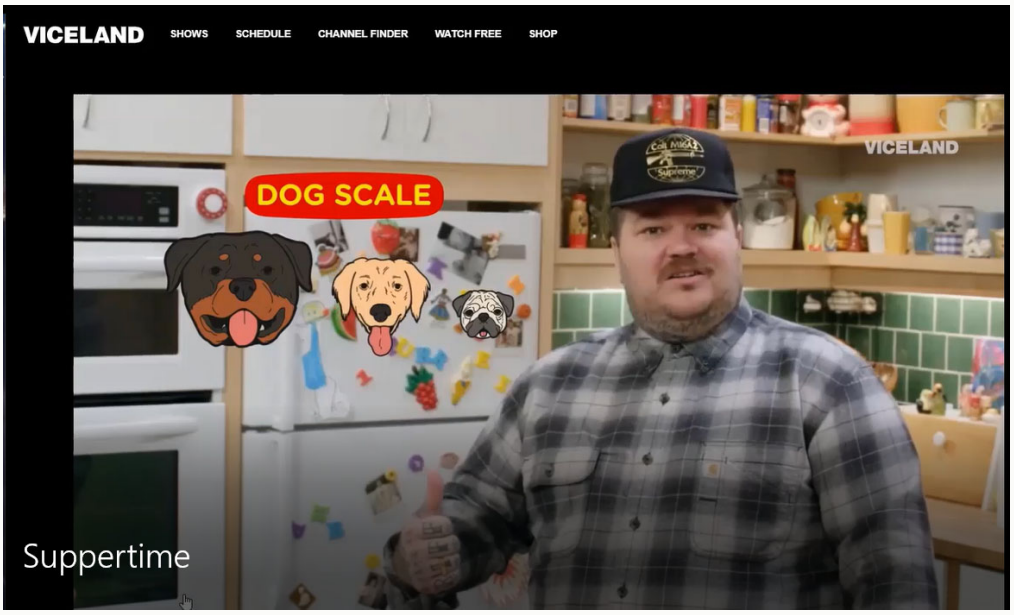
QSM Rule-of-Thumb:
- Simple is .50 x Average
- Complex is 2 to 3 x Average

Article: Big Rock Estimation by Aaron Jeutter on QSM website

QSM The Intelligence behind Successful Software Projects

The reference is presentation given at a QSM Agile Round Table discussion (see QSM website).

Sizing Grounding

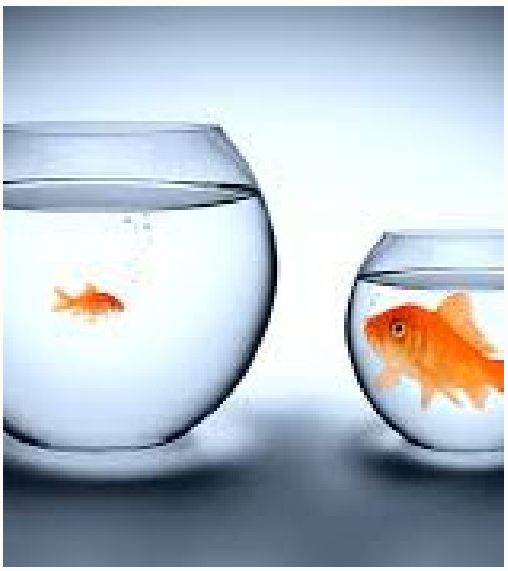


QSM The Intelligence behind Successful Software Projects

Like Aaron, you can use something like Matty's dog scale, which he uses to rank the complexity of food recipe! Have fun.

One Size Does Not Fit All

- What data is currently available?
- What can be gleaned from tools & methods?
- What measures are available at different points in the life cycle?



QSM The Intelligence behind Successful Software Projects

Questions?

www.qsm.com
info@qsm.com

800-424-6755

PMI PDU claim code:
3670Z0P591

QSM The Intelligence behind Successful Software Projects