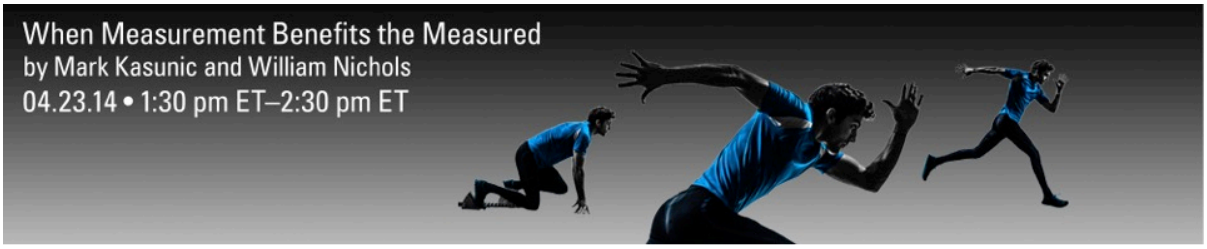
When Measurement Benefits the Measured

Table of Contents

Welcome



WELCOME

**106 Hello, and welcome to the Software Engineering Institute's Webinar Series. Our presentation today is: When Measurement Benefits the Measured.

Depending on your location, we wish you a good morning, a good afternoon or good evening.

My name is Shane McGraw, your moderator for today's presentation; and I'd like to thank you all for attending.

Mark Kasunic

Mark Kasunic
Senior Member of the Technical Staff
Software Engineering Institute

Mark Kasunic is a senior member of the technical staff at the Software Engineering Institute (SEI) at Carnegie Mellon University. He is currently a member of the Team Software Process Initiative within the Software Solutions Division. Since joining the SEI in 1994, his work has focused on transitioning performance improvement technologies into practice through applied research, course development, coaching, and training. His current research and development interests include data quality assessment and improvement, project performance measurement, and practical measurement and analysis approaches that help individuals and teams improve their technical performance. Mark has an extensive list of technical publications and conference presentations addressing software engineering and measurement. Before joining the SEI, Mark was an engineer and manager at Boeing in Seattle. He has a Masters Degree in Systems Engineering and is a senior member of IEEE. Mark is a certified TSP Mentor Coach and a certified Scrum Master.

**001 And now I'd like to introduce our two featured speakers for today.

First I'd like to introduce Mark Kasunic. And Mark is a Senior Member of the Technical Staff at the SEI, within the Team Software Process Initiative or TSP; and he is in the Software Solutions Division. And since joining the SEI in 1994 his work has focused on transitioning performance improvement technologies into practice through applied research, course development, coaching and training.

Our next speaking will be Bill Nichols. And Bill joined the SEI in 2006 as a

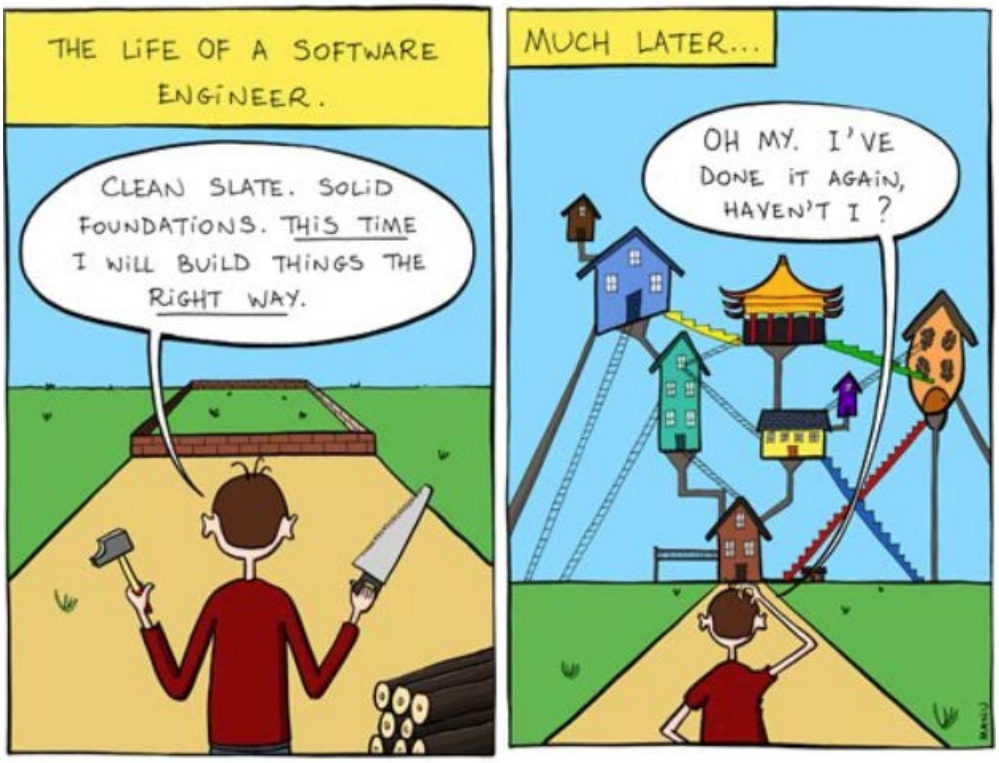
Senior Member of the Technical Staff. He is currently a personal software process instructor and team software process mentor coach; and he is also in the Software Solutions Division. And prior to joining the SEI, Bill led a software development team at Bettis Laboratory near Pittsburgh.

And now I'd like to turn the presentation over to Mark Kasunic. Mark, all yours.

Mark Kasunic: Okay. Thank you very much.

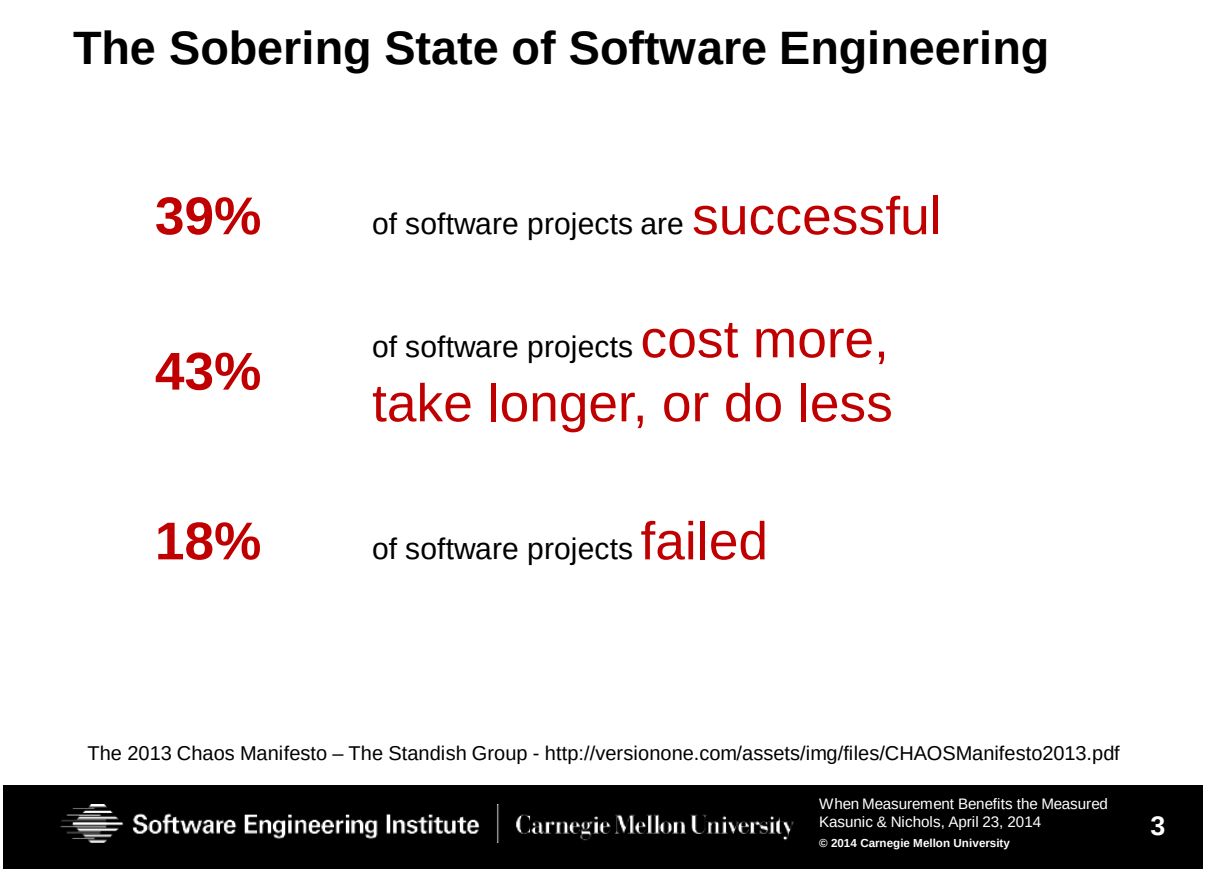
Hello. I'm Mark Kasunic. I'm glad you could be with us today.

Building Things the Right Way



**002 To get started, what I'd like to do is to throw out a question to all those software engineers out there. What I'd like you to consider is does this cartoon, this situation that you see in the graphic, does this cartoon resonate with any of you out there? That is, can you relate to this type of situation? Maybe just a little bit. Yes/no?

Well let's look at some data.



**003 This is from a 2012 CHAOS Report on the State of Software Engineering.

I realize this looks kind of bad. But what I can say is there have been improvements- there have been actual improvements in these statistics over the last 10 years.

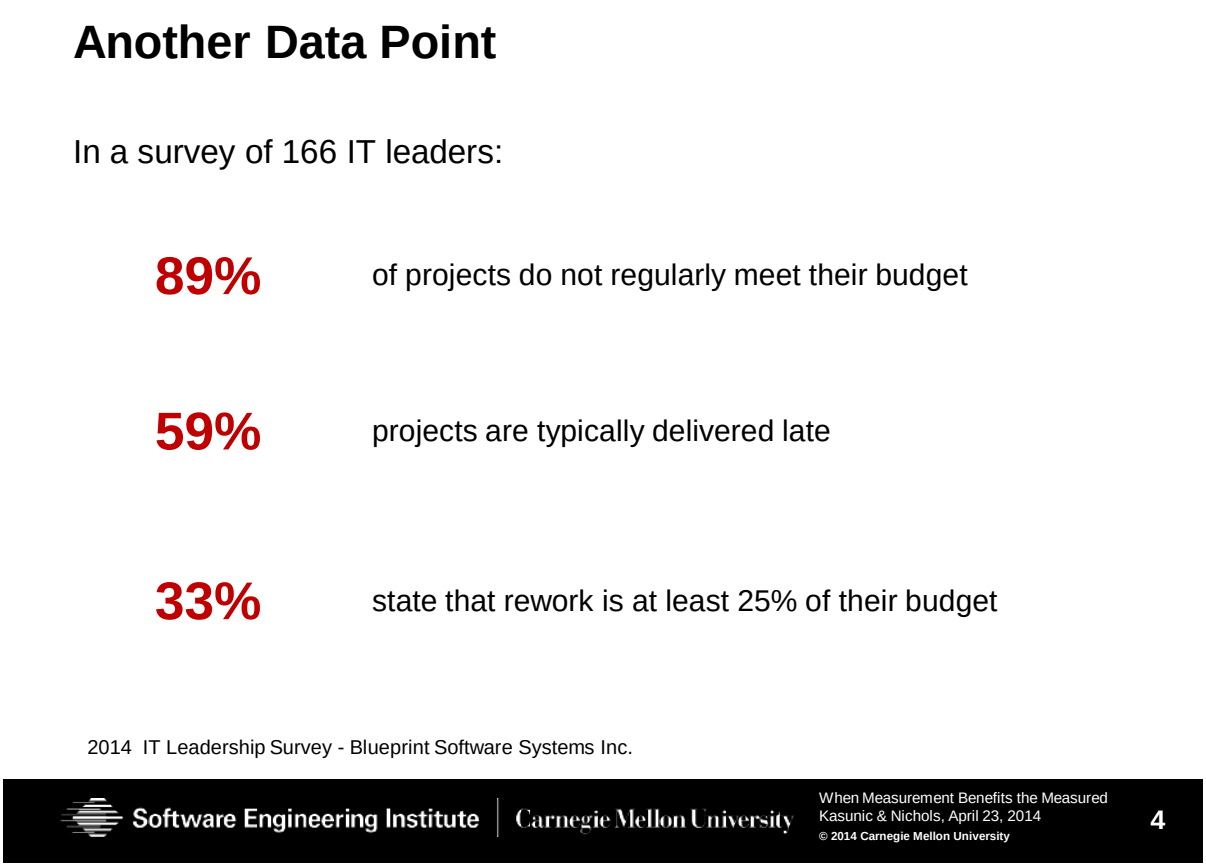
But still, only 39 percent of projects are completed on time and on budget.

Forty-three of the projects are considered challenged; that is, the projects were completed but they were over-budget or late, and they

offered fewer features than they originally specified.

Eighteen percent of the projects were cancelled at some point during the development cycle. That's not so good.

Another Data Point



**004 Okay this is a different survey, another data point.

The first two figures are pretty much similar to the CHAOS Report findings that we just looked at. But take a look at the last figure here, because it is directly related to quality performance.

Thirty-three of- thirty-three percent of IT leaders state that at least 25 percent of their budget is spent addressing rework; that's at least 25 percent. Of course this is a tremendous resource strain of an organization's ability to deliver functionality and business value.

Problem?

**005 So do we have a problem here? Well when you see numbers like this, it can't help but to make you feel that as a community we need to do better.

Can Measurement Help?

> CEO's have a lower opinion of software groups than of other technical groups due to consistently optimistic estimates, schedule delays, cost overruns, poor quality when delivered, and outright failures. Software is much worse in all of these.
>
> Better measures of projects … will improve the professional status of the software community and perhaps lead to CEO's having more respect for software groups than they have today.

Capers Jones
InfoQ Interview
March 30, 2014
http://www.infoq.com/articles/Jones-measuring-agile-adoption

**006 This is from a recent InfoQ interview with Capers Jones-- who I'm sure many of you have heard of; he's a well-known researcher and author-- from onsite interviews with hundreds of software organizations.

He draws the conclusions that you see on this slide; namely that when you- when compared to other technical teams within their organization, software teams are not getting much respect from their CEOs. And this is due mainly to a failure associated with the lack of engineering and product development discipline, especially in the area of measurement; which is the topic of our webinar today.

The World Without Measurement

Science?

Engineering?

Medicine?

Software Engineering Institute | Carnegie Mellon University

When Measurement Benefits the Measured
Kasunic & Nichols, April 23, 2014

7

**007 When you think about it, can you imagine what the performance results of other technical areas would be if measurement wasn't considered an integral part of their domain? Of course measurement is considered an integral part of any other technical area. It's just taken for granted.

We All Measure…

We All Measure ….

Driving?

Clothing Size?

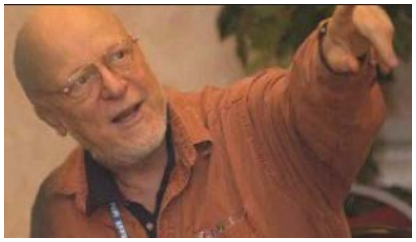
Cooking?

Getting to Work on Time?

**008 And when you think about it, measurement is actually an integral part of all of our lives. We're measuring all the time, whether we realize it or not.

For example, think of yourself driving down the road, looking outside your front windshield. When you see the brake lights go on for the car up ahead of you, when you see that happen you quickly-- unconsciously you measure the distance to that car visually. Why? So that you know whether you need to hit your brakes or not.

How We Use Measurement



Gerald (Jerry) Weinberg

First-Order Measurement

What *seems* to be happening*?*

Tends to be qualitative and fast.

Second-Order Measurement

What's *really* happening? And how is it changing?

It needs to be quantitative; subject to more refined models.

Third-Order Measurement

What *happens* in a more general and universal sense?

Needs to be precise with checks for validity; statistical variation must be characterized and interpreted appropriately.

**009 That type of subjective measurement is what Jerry Weinberg referred to as First-Order Measurement. It's qualitative and it's fast. It answers the question: What seems to be happening? We use this type of measurement all the time.

Now if you are driving down the road and you notice a car with a red light on the top of it, a police car, you probably- probably would shift to Second-Order Measurement. You would actually look down at your speedometer on your dashboard to see how fast you're really going; and then make whatever adjustments you need to avoid getting a speeding ticket.

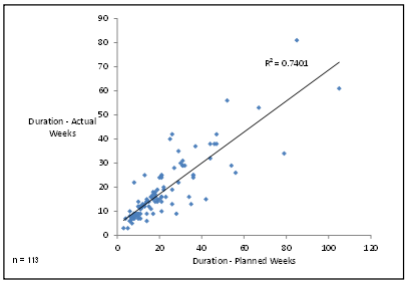
Now Third-Order Measurement addresses measurement trends and patterns.

If I'm- let's say I'm going to buy a new car. I wouldn't really want to rely on First-Order Measurement; you know, the opinion of my neighbor about his car. I would probably go beyond that. It's a big purchase. I'd go beyond First-Order or Second-Order Measurements and I'd probably use Third-Order Measurement. I'd want to see statistical data on performance of past models of that type of car I'm considering because it's- like I say, it's an important decision.

Second-Order Measurement

First-Order Measurement

Second-Order Measurement



Third-Order Measurement

Software Engineering Institute | Carnegie Mellon University
When Measurement Benefits the Measured
Kasunic & Nichols, April 23, 2014

10

**010 So the type of measurement that we use should be a function of the type of decision making that we need to make.

And if you're wondering what those guys off in the slide are doing with their First-Order Measurement, they're just licking their index finger and holding it up to check the direction of the wind. A good example of First-Order Measurement; very subjective and qualitative.

The State of the Practice

An Issue

The results of applying many software development methods are unpredictable.

Decision making about method selection is based on suppositions, opinions, and fads.

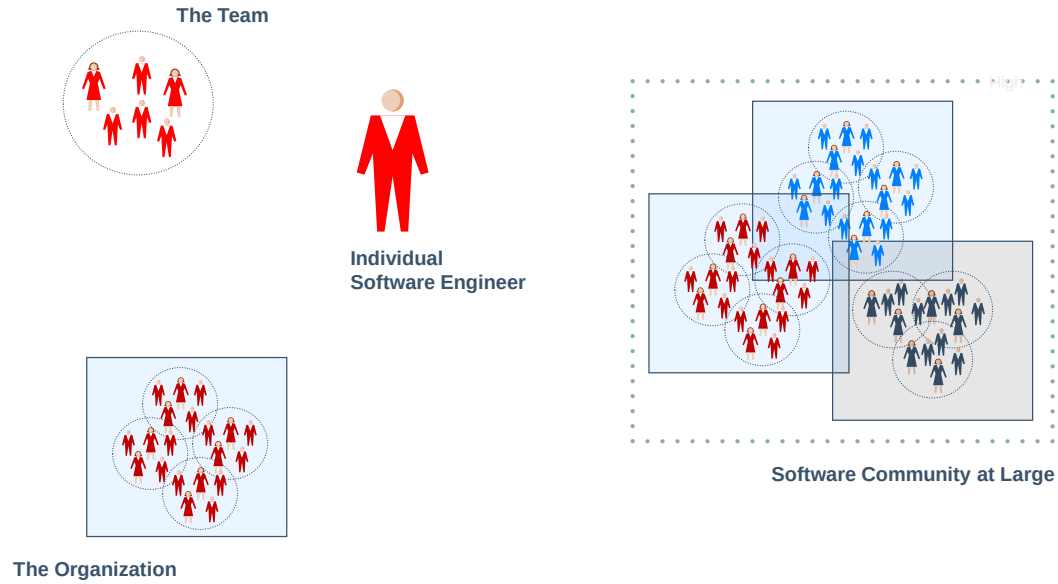
What We Need

We need to set aside perceptions and market-speak … and transform software engineering into an engineering discipline.

Decisions should be based on fair and unbiased analysis of information.

**011 Now at issue-- with software development here's what we're seeing. Decision making is largely based on First-Order Measurement, at best. Quite often claims made about the effectiveness of development approaches are based on perceptions of whatever the new fad happens to be.

Now if we go on to mature software engineering into a real engineering discipline, then we need to elevate our game in the measurement arena and become more like other engineering disciplines, other technical areas, where the use of Second and Third-Order Measurement is taken for granted.



**012 So that's what we're going to be addressing in today's webinar: Elevating our game with measurement. And we'll do that from several perspectives including at the individual level, the team level, and then briefly beyond to the organization and community level.

Measurement & the Individual Software Engineer

You

**013 So let's talk about- first let's talk about you, the software engineer, and how you can use measurement in a way that benefits you personally.

Necessary Feedback

Stellar athletes understand that they must set specific goals to reach their potential.

Measurement provides the necessary feedback that drives athletes to achieve world-class excellence.

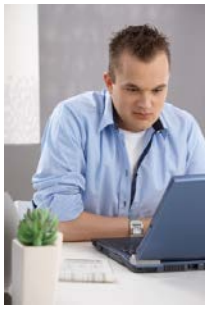
**014 Now I'll bet that some of you out there, you play sports; you play golf, compete in races, go bowling, lift weights. If you do, then you probably keep track of your performance. Perhaps it's as simple as I knocked off three strokes from my golf game today. Right? I clocked in on my two-minute mile 16 seconds faster than yesterday.

Athletes measure like this because they want to know whether their performance is improving or declining; and how their performance compares with their own personal best or with the performance of others.

This is the type of measurement, feedback measurement, that is valued; because it drives action. It helps you achieve your goals. If a measurement doesn't drive action, then what value does it have?

Measurement and Software Engineers

Can software engineers leverage goal-setting and measurement in the same way?



**015 And so the question is: Can software engineers use measurement in this way?

Do you?

Do you?

**016 More specifically: Do you use measurement in this way?

Measures Poll

What type of measures do you *typically* use to estimate the duration of your work for a schedule?

- ☐ First order: Qualitative - based on what I think I've done before.
- ☐ Second order: Quantitative - based on quantitative data from previous project(s).
- ☐ Third order: Statistical - based on statistical patterns of data from my previous projects.
- ☐ None of the above.

**017 So let's conduct a little poll. I would like you to answer the following question.

The scenario is that you have been asked to estimate how long you believe a new task is going to take, at your work. Select the response that you believe is the typical approach that you would use to help you provide the estimate.

Shane McGraw: Okay we got 50 percent as the Second order; First order at 36 percent; Third order 14 percent; and None of the above at one percent.

Mark Kasunic: Okay.

Shane McGraw: So back to you.

Mark Kasunic: Interesting. I wish you were here with me so that we could have a little discussion about that.

Should You Be Using Measurement?

Should You Be Using Measurement?

Yes. And it needs to go beyond first order measurement.

Measurement is needed to manage your work.

**018 But we'll do the best we can.

Well should you be using measurement? I'm glad to hear that a lot of you are, according to your responses.

You know, yes we should be using measurement; and it should go beyond first order measurement.

And why do we need measurement? Well the reason you need to use measurement is because you need it to make good decisions about how you manage your work. That's basically why you need measurement.

Managing the Work

Managing the Work

**019 Now I can hear a lot of you out there-- I can imagine what you're thinking. Isn't it the manager's job to manage my work? I'm an engineer. I do engineering work; the manager manages the work.

The Evolution of the Management Approach

Body Management

People as oxen.

Frederick Taylor

Task Management

People as machines.

Peter Drucker

Knowledge Management

People as individuals.

**020 Well not exactly. And I'd like to talk about that a little bit.

The approach to management has actually evolved over centuries to this day. As you see on the side: What has driven this evolution?

Evolution of the Worker

**021 Management approaches have evolved because there's been an evolution of the worker, way back from the hunter-gatherer to the technology professional of today--that would be you.

A major turning point in this evolution was the advent of the Industrial Revolution. You see, prior to the Industrial Revolution work was performed by craftsmen who had learned their jobs through lengthy apprenticeships.

They made their own decisions about how to do their job. They didn't have any bosses. There was no

standardization. But now they were in the factories. At the same time you had many unskilled and uneducated workers flocking from the fields to the factories.

As you can imagine, operations were rather inefficient during this transition.

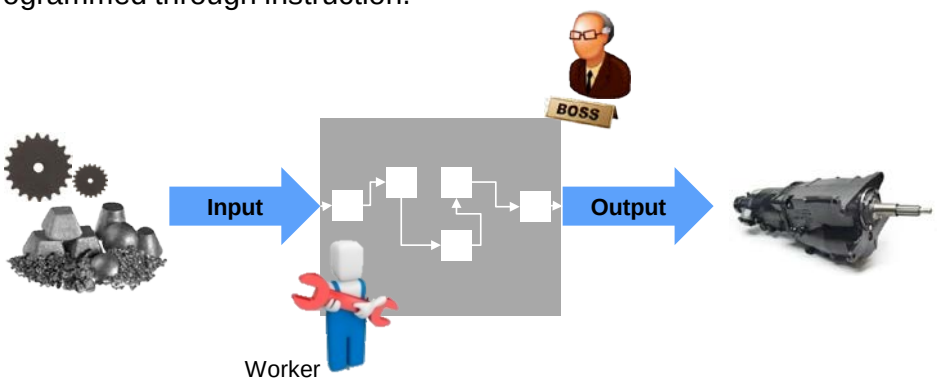
Taylorism – Scientific Management

Taylorism – Scientific Management

For years, the basic power equation in organizations was simple and effective:

Knowledge held by a few (the managers), plus iron discipline over the many (the workers).

The worker was viewed as an instrument, a bundle of muscles programmed through instruction.



**022 It was in this type of environment that inspired Frederick Taylor to write his book titled: The Principles of Scientific Management; also called Taylorism now.

Taylorism looked away- took away the autonomy of the craftsman worker and converted labor into a

series of simplified steps- simplified jobs that could be performed by unskilled workers.

Taylor's main premise was that the knowledge about how the work should be done existed only among management; that is, managers think, workers do. That was the dichotomy that Taylorism brought to the workplace.

Well that worked okay for awhile.

The Birth of the Knowledge Worker

The Birth of the Knowledge Worker

The Technology Professional

- New data processing age was born during 2nd half of 20th century
- Work became asynchronous and non-linear
- Nature of knowledge work demanded significant control by the worker (instead of the manager)

**023 But the world was rapidly changing and so was the nature of the worker. After World War 2 more than 50 percent of veterans used the GI Bill to obtain a college degree.

By the late 20th century there were major changes: social, economic, political and technical. There was a significant shift in how work was being performed due to the technical innovations that were coming about.

What Differentiates Knowledge Work?

What Differentiates Knowledge Work?

Manual work

Consists of converting materials form one form to another.

The work output is tangible.

Knowledge work

The work is done in the head.

The work can't be seen.

**024 The advent of this Knowledge Age really brought a real shift in terms of how work is viewed and managed.

With manual work you can actually observe the product being constructed. However knowledge work is much different; the work is done in the head and so you can't really see it being performed. It's intangible.

Management can't see the work performed, like they could when they were managing manual work. So in a real sense they lost their ability to manage this type of work, this type of knowledge work.

A Shift In the Locus of Control

A Shift In the Locus of Control

This new breed of worker has a new job: converting knowledge into actions that convert information from one form to another.

Because the behaviors of a knowledge worker are primarily private, supervisors cannot supervise.

Due do the nature of knowledge work, it is the worker that has almost total authority in matching methods to the varying job tasks and situations that they encounter.

However, with this reality, there is also a shift in responsibility …

Software Engineering Institute | Carnegie Mellon University
When Measurement Benefits the Measured
Kasunic & Nichols, April 23, 2014

25

**025 So the basic tenets of Taylorism, well they have become untenable in the Knowledge Age. It's now really up to the knowledge worker to manage the work; because the work is in his or her head.

However with this new reality, there needs to be an attitude shift on the part of the knowledge worker. Knowledge workers-- that would be

you, the software engineers-- must take responsibility for managing your work.

Managing the Work

**026 So that was kind of a long-winded response to the question posed by the guy in the blue shirt there; that in summary no, it's not the manager's job to manage how you work. Management provides goals; managing the work is really your responsibility.

Controlling Your Own Destiny

To control the way they work, software engineers must plan their projects.

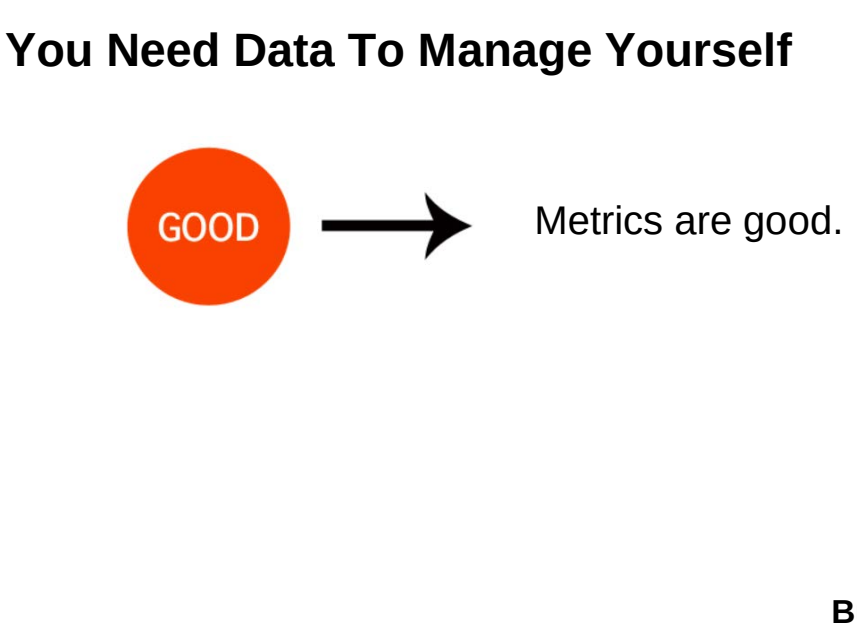
For management to trust these plans, the engineers must make accurate plans.

To make accurate plans, they must have data.

To have data, they must measure their performance.

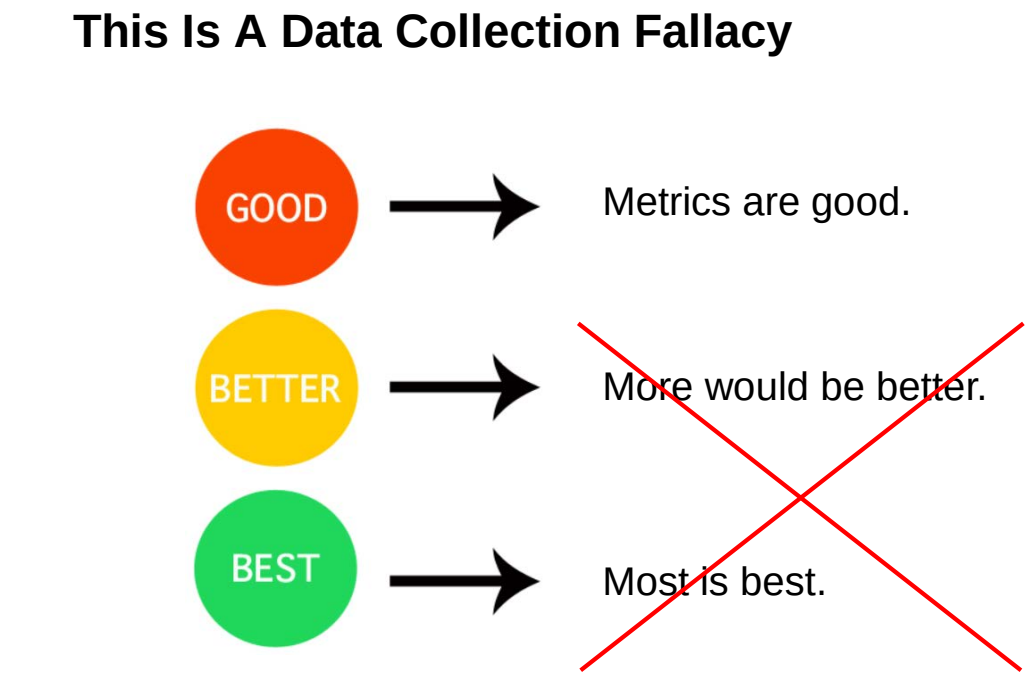
**027 You're the one that needs to have control over the way you do your work; and to do that you need to plan your work. If you want management to trust your plan, you need to make an accurate plan. To make an accurate plan, you need to have data; otherwise you're just guessing. To have data, you have to measure your performance.

You Need Data To Manage Yourself



**028 So you see, measurement is a good thing because it can help you to better understand and manage your work.

This Is A Data Collection Fallacy



**029 However-- this is not true:
It's a data collection fallacy. Let's not take this too far. Metrics are good.

Only Four Basic Measures Needed

Software engineers only need to collect four basic measures to mange their schedule performance and the quality of their work.

Time on Task

Size

Defects

Schedule

**030 But the fact is for a software engineer to be able to manage the work there are only four measures that you need to collect: the size of the product; the actual time you spend on a task in your plan-- we call that task time; the defects, you need to track the defects to manage quality; and schedule, that's the date that you complete a task. Four basic measures, that's it.

Measures Are Estimated and Then Tracked

At the beginning of an effort, the work is planned and divided into a *set* of tasks or activities called *phases.* The basic measures are *estimated.*

- product size
- time-in-phase
- defects injected into a phase
- defects found in a phase
- task completion dates

During the project, these measures are collected in real time

- time-in-phase
- defect type injected in phase
- find/fix time for each defect
- task completion dates

When a product has been completed

- product size is measured

**031 You first estimate these measures during planning; and then you collect actual measurements as you work. Because you're collecting these measures in real-time, they're very accurate. By collecting these measures you can answer all sorts of important questions about the status of your schedule and the quality of your work.

Measured by Others

No one wants to be measured by others.

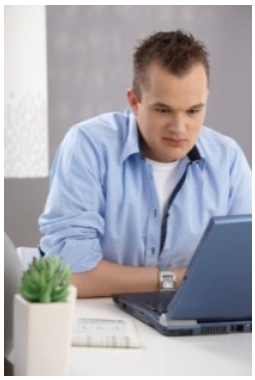
And, that's not what we're talking about here.

**032 Now one thing that I want to emphasize here is that we are talking about measurement that is collected by you and for you. They're for your benefit. They're not something that you really need to share with others; unless you want to.

Tool Support For Data Collection

Collecting the four core measures would be impractical in the absence of software support.

A number of tools are available that make it easy to collect this type of data.

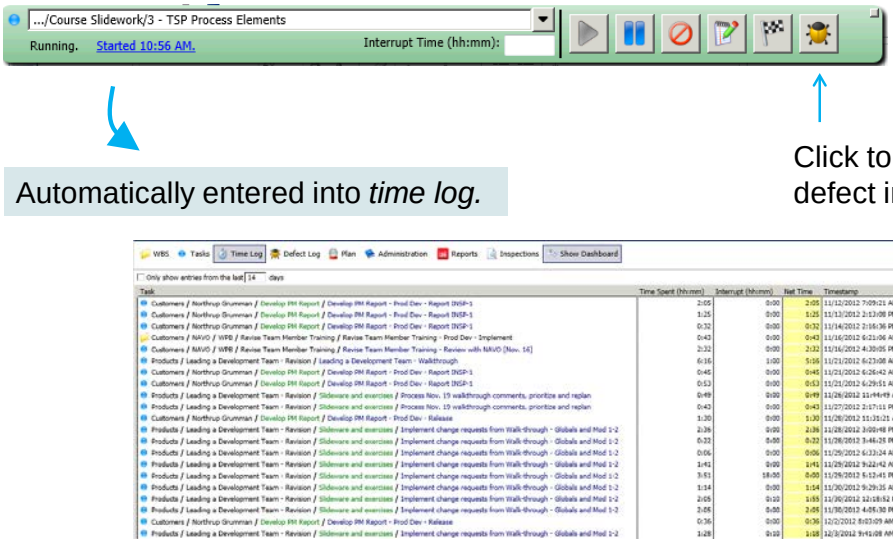


Software Process Dashboard: http://www.processdash.com/

**033 Now data collection like this would be impractical.-- I think we all agree about that-- without tool support. And tools have been developed to support this type of data collection, facilitating the entry and analysis of the data.

As an example is the Software Process Dashboard. It's a free downloadable tool that is specifically geared for collecting and processing the four basic measures. The tool converts the data you collect into a variety of useful derived measures and graphic indicators.

Collecting Personal Data



**034 The way it works is you enter your task into your personal plan, into the tool. You then collect data that is automatically logged against the task and it's collected in the tool. What you're usually working with is that skinny dashboard up at the top of your screen.

Tracking Your Time

All activities that contribute to the value chain are listed as tasks in your plan. Work against any task in your plan is timed.

- When you begin work on a task, you start the timer in the tool.
- When you stop work on a task, you stop the timer.
- The tool calculates durations automatically.
- If your task is interrupted, you can stop and then restart your timer to resume.
- If you forget to use your timer, you estimate your time-on-task, and enter it manually.

**035 To collect time data you are just clicking on a dashboard control; click to start or resume a task and click to pause or complete a task. In each case the tool is calculating the durations automatically and entering it into your time log.

Now like I said before, the most important attribute of any measure is that it drives action; otherwise it's of no value.

Okay. So why do you care about tracking your time? Thank you, I thought you'd never ask.

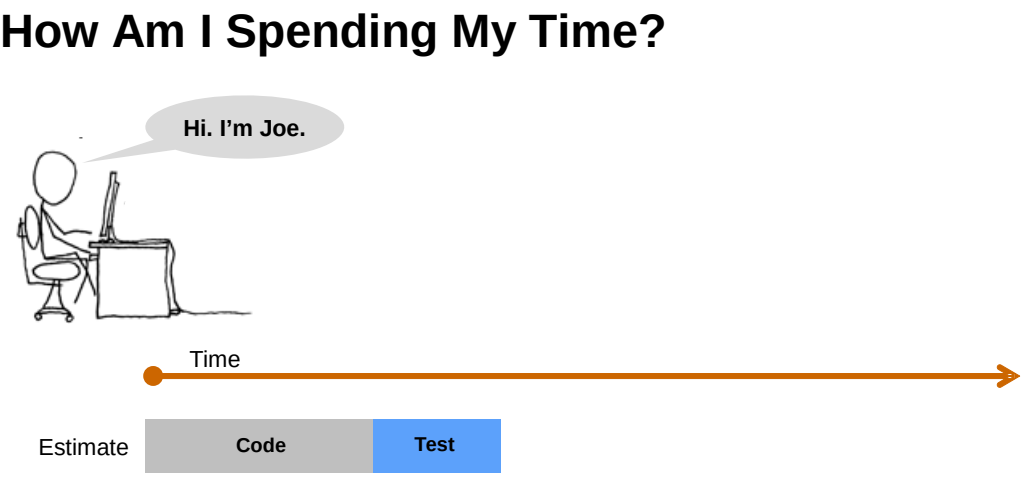
Time

**036 Look, time is your most valuable commodity. When you track your task time, you can learn from the data to improve your estimating and forecasting. Estimates become more accurate and more meaningful.

When you make a commitment, you can feel confident that you will actually meet the commitment.

Being trustworthy, being someone that meets their commitments; it's very important. It's the type of person that you want to work with. Right? It's a valuable team member.

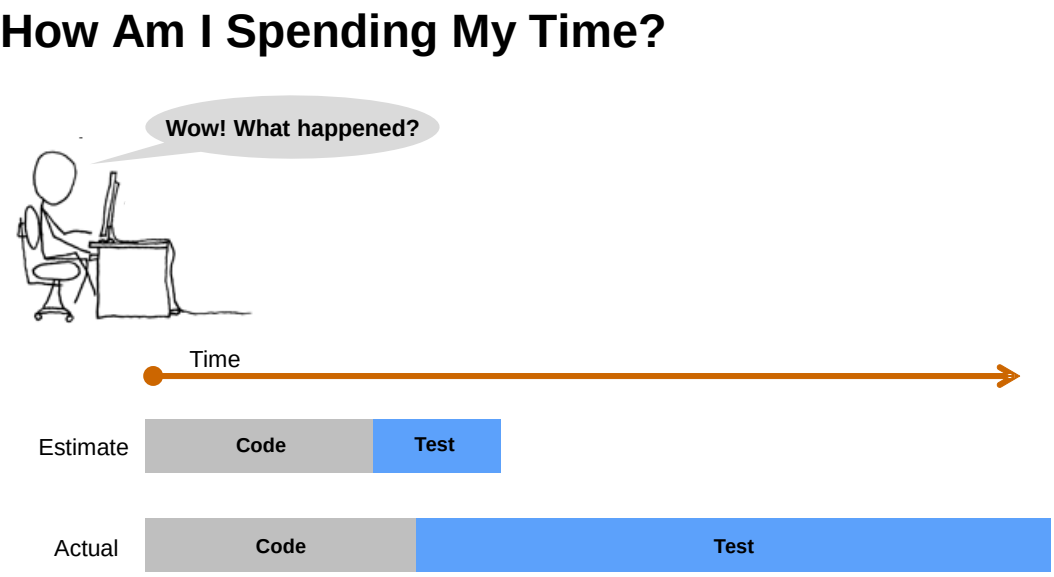
How Am I Spending My Time?



**037 Let's shift into a very simple example of how tracking time can drive action.

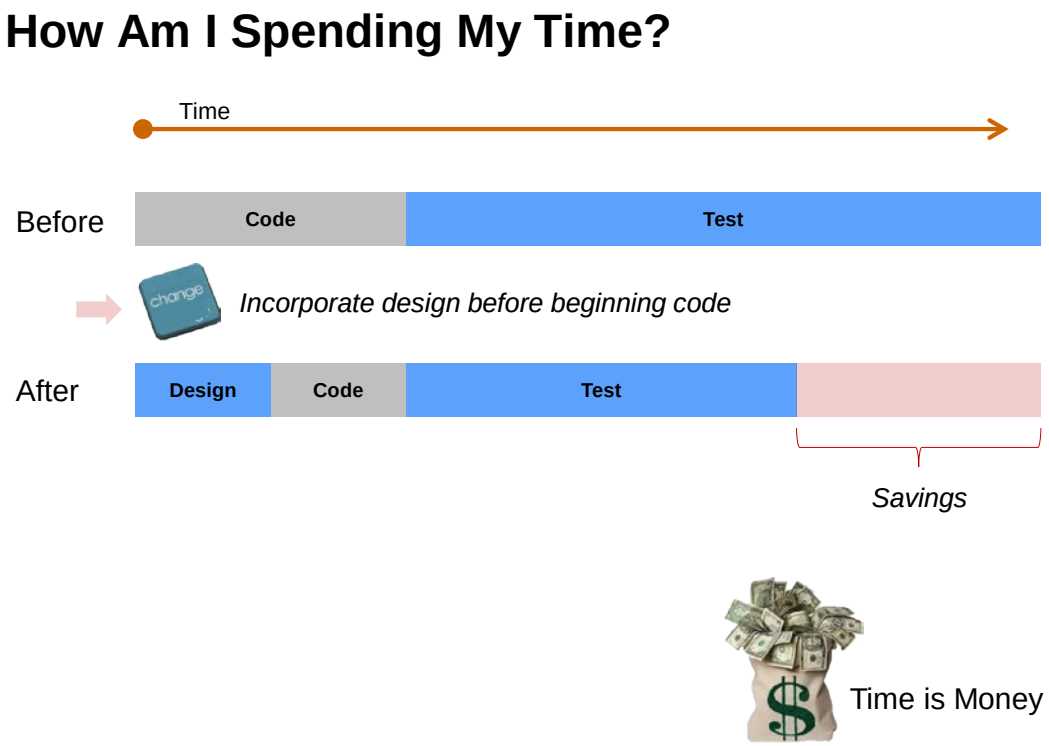
Meet Joe the software engineer. Joe has been coming in late with estimates on a regular basis. So he's going to see if tracking his time will help him. So he looks at what he needs to do and comes up with an estimate for how long he thinks it'll take to code and test. He then starts the work, he completes the task, records the actual code size and then examines his time log.

How Am I Spending My Time?



**038 Okay? Well Joe is pretty surprised and not so happy with the results. He knows that he tends to just jump into coding because he feels that he can figure things out as the code emerges. But the next time he is going to sit down with paper and pencil to map things out before he begins coding. He'll measure again to see if the change makes a difference.

How Am I Spending My Time?



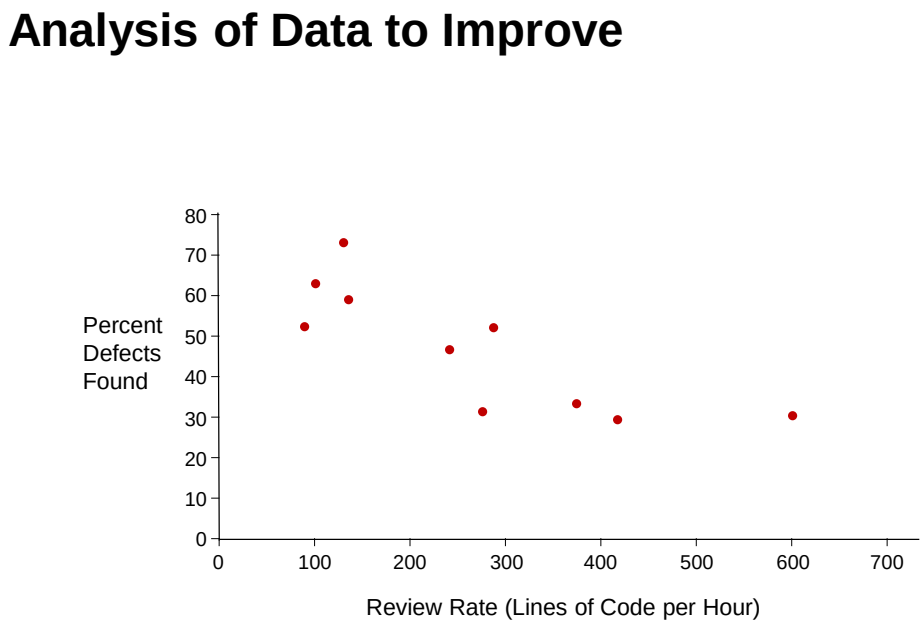
**039 This is the result. Okay?

Compared to his previous performance, Joe learned that by incorporating some design at the beginning, before he started coding, things went much smoother; and there weren't so many difficult problems to fix during test.

The one thing that occurred to Joe is that he would never have been able to compare his performance like this before because he had never tracked his time before. But with this type of data feedback, he is now feeling much more control.

And that's the thing about collecting personal data like this. It actually makes you feel empowered and it makes you feel in control.

Analysis of Data to Improve

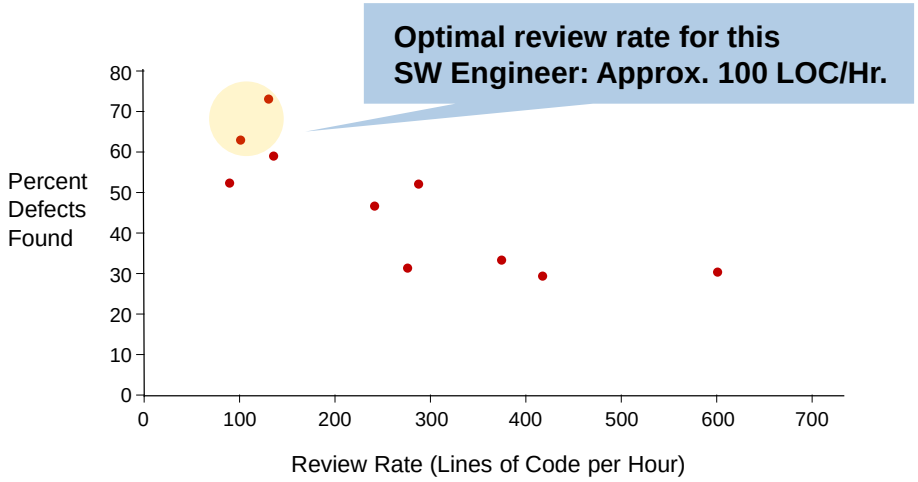


[Humphrey 2005, page 192]

**040 Joe took a further step; he started tracking his time doing a personal review of his code.

Here is his data. He tracked his review rate against the percent defects that he found.

Analysis of Data to Improve



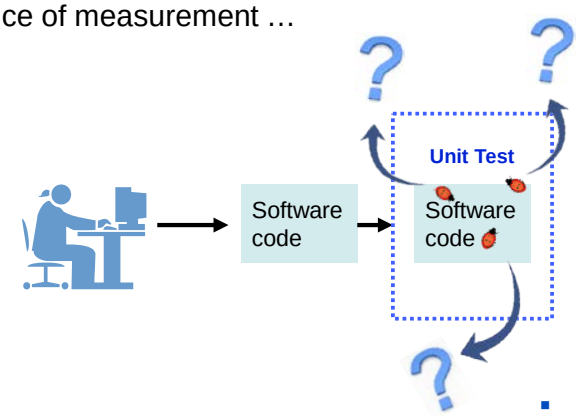
[Humphrey 2005, page 192]

**041 And what he found is there's this sweet spot. It looks like he should be reviewing at a rate at about 100 lines of code per hour. If he goes faster than that, then the review isn't as effective.

But how would he have known this without measurement?

What About Quality Performance?

In the absence of measurement …



- No feedback loop
- No learning
- Mistakes will happen again

**042 He wouldn't have.

So what about quality performance? We were talking about schedule.

This slide is depicting a typical approach that we encounter all the time with software engineers: The software programmer being led around by the nose of their debugger. Rather than using data that is readily available from the work process, to learn and to avoid making the same mistakes, rework over and over again. In this case the debugger is in control of the work; not the developer.

Defect Tracking

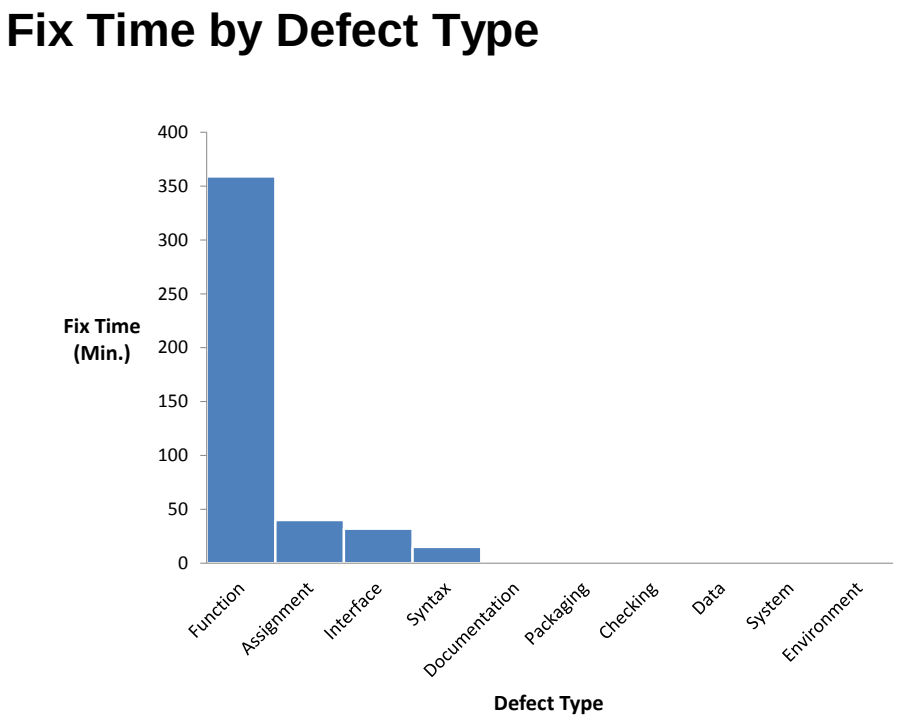
Most defects are discovered during personal reviews, inspections, and other quality control activities.

Whenever a defect is found, you open the defect log of the planning/tracking tool and record the following:

- the start time when defect was found
- defect type
- the process phase where the defect was injected
- the process phase where the defect was removed
- a brief description of the defect
- the stop time (when you have completed *fixing* the defect)

**043 There is a better way; and that better way is to take an engineering approach and collect data that we can learn from, so that we can become better at what we do; using the same tracking tool that we talked about earlier. You can find a defect and start the timer, select the defect type and record a brief description of the defect. The timer is stopped when the defect is fixed. It's all recorded in a tools defect log.

Fix Time by Defect Type



**044 Now you can view this type of data in various ways as generated by the tool. Here the data is organized to show you which types of errors, which types of defects, are causing you the most amount of time to find and fix.

**045 Again, it's a waste of time to collect measures if it doesn't drive action. The idea behind collecting the defect data is to help you find ways to improve your performance.

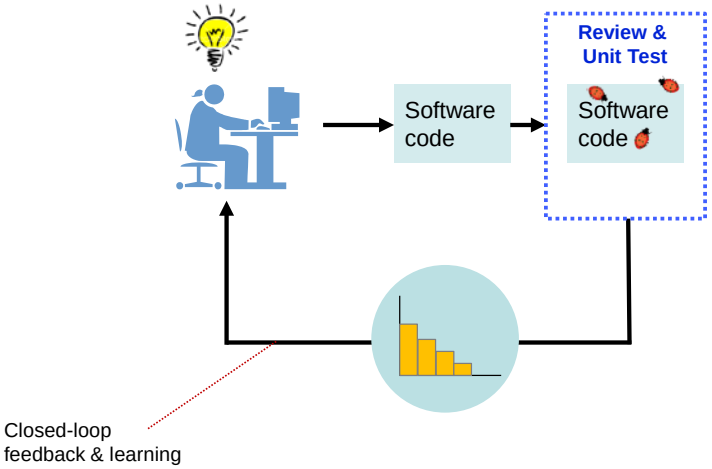
A good way of using the data that we just talked about is to update your review checklist, based on the defects that are typically being injected into your code and the ones that are causing you the most problems. That way when you are in your reviews, you'll be sure to look for those types of defects in the future.

And why do you want to catch them during your reviews? Why not just test them out? Well our empirical data shows us that if you can find the defect and fix it in review, it's 20 times more efficient than finding and fixing it in unit test.

Using Measurement To Understand…

Using Measurement To Understand …

… and, to get better



- Defects recorded by type and time-to-fix
- Closed-loop feedback
- Learn from mistakes so they don't happen again

**046 And again, time is money.

So that's the idea here. You use your personal data to learn and improve the quality of your work. You learn from your mistakes so that they don't keep happening. This is what performance improvement is all about. You use measurement to help you get better.

Taking Responsibility

Others cannot manage how you estimate your work and how you manage the quality of our work.

Knowledge workers manage themselves with data.

Software engineers are knowledge workers.

Only four basic measures are need to manage your work.

**047 So in this section that I've talked about, we've emphasized that you the software engineer are a knowledge worker. As a knowledge worker, you need to take responsibility for managing your work; including how you make your commitments and how you control the quality of your work product.

Measurement can most certainly help you. There are only four basic measures that you need to manage your work.

Software Engineers and Athletes

Can software engineers leverage goal-setting and measurement the way that star athletes do?

Yes! Absolutely.

**048 So coming up, Bill is going to take the reins and describe how teams can benefit when they collectively agree to estimate and collect the four basic measures that we have just covered.

Shane McGraw: From Matt asking: How can you trust the data you measure when no two tasks are the same?

Mark Kasunic: Hum. Well- you know, I wish we had more time to go into the details of how this data is collected.

But what we're doing is we're collecting time against tasks that are in our plan; and those tasks are

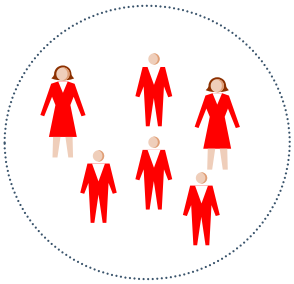
typically organized into phases. We're grouping those into phases. And so what you're learning about is how much time do I spend on a task that's in, for example, the review phase; or how much time am I spending coding? How much time am I spending in test? More or less getting a handle over those types of things.

Shane McGraw: Stuff that's repeatable that you'll be doing over and over.

Mark Kasunic: Right, stuff that's repeatable. You're learning each time you're doing it. As you do this over time, you're actually collecting information, establishing a historical database that you can go back and take a look at, analyze and make better decisions with.

Measurement On Your Team

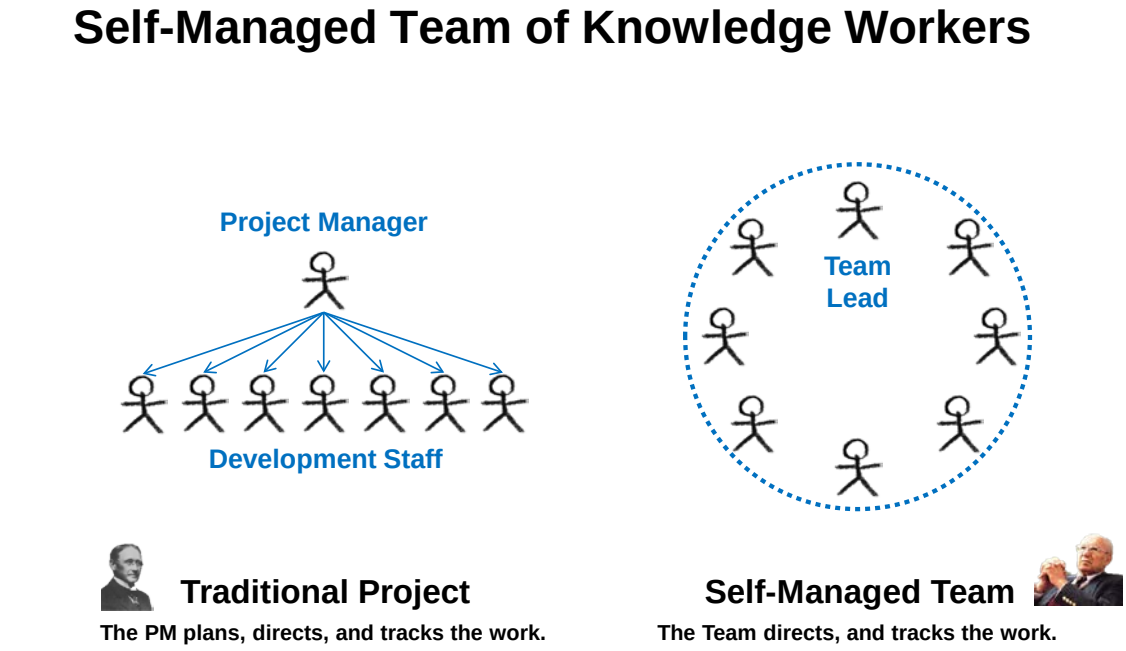
The Team



**050 Bill Nichols: So what I'm going to talk about is measurement at the team level. You know, Mark talked a good bit about the measurement, the personal level; and we had a question about measuring tasks that are different all the time.

Guess what? Variation happens. There's going to be a lot of variation from task to task, from instance to instance, from team to team.

So part of what I'm going to show are some data that our partners have very graciously shared with us from their own personal data.



**051 So you can get a sense as to what are the real ranges of performance out there.

Before we go any farther, let's talk about the differences of what we can measure now using this paradigm.

If we look at the traditional project management, you could only measure what the boss could see. The boss had to be able to see it to measure it.

By having the empowered team, the people who define the process, who define the practices and measure themselves, measure the work, a whole new range of activities become

able to be measured for the first time; because the people doing the work can actually measure it themselves and they can see things that the boss could never see.

Self-Managed Teams Plan Their Work

Self-Managed Teams Plan Their Work

Management provides the goals and constraints for the project.

The team then develops its plan for meeting management's objectives.

If necessary, the team negotiates with management to arrive at a mutually agreeable outcome.

But it's up to the team to manage their work! Not someone else.

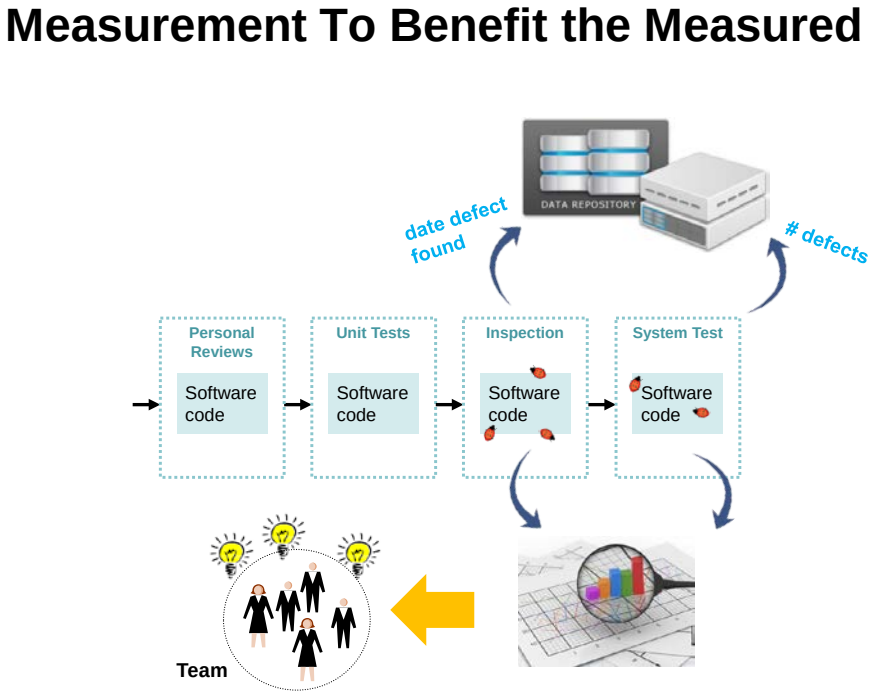
The team must have data to manage the work, to meet their commitments.

Measurement helps teams manage their commitments.

**052 Now it becomes more important, the more the work involves creativity, responding to change instead of following a specific fixed plan but adapting your plan, intangible outcomes that can't really be seen, the greater the importance for the worker autonomy. That means the workers have to define the tasks. They have to define the work practices. They have to measure them.

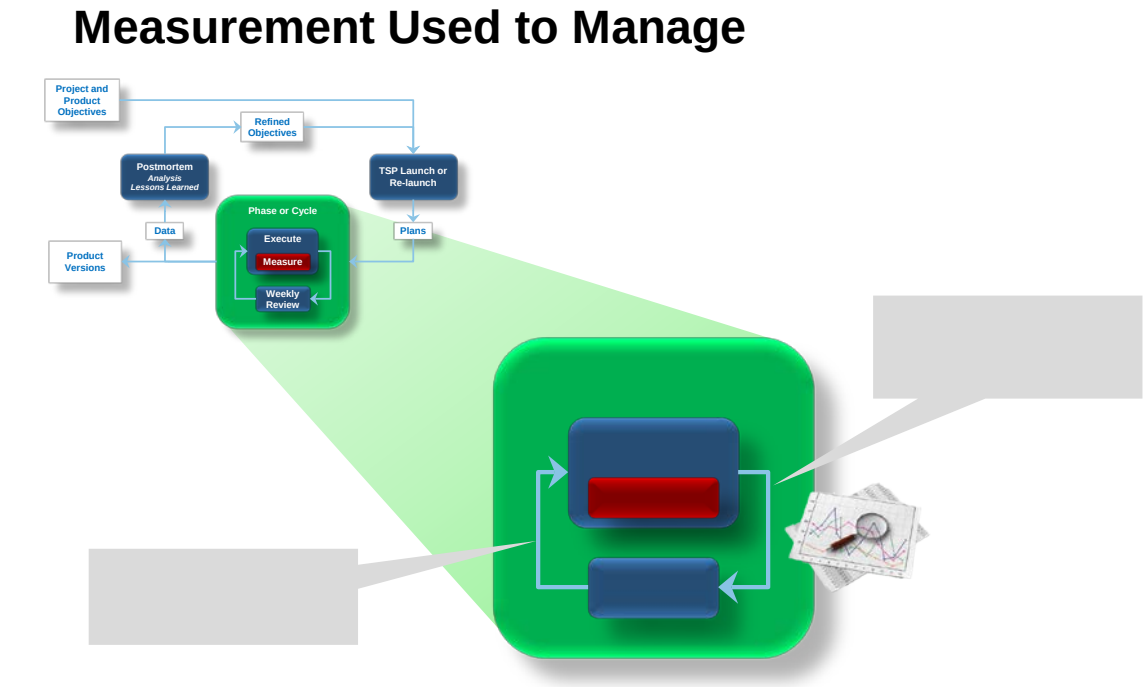
Management is still important. The manager has to provide the resources. The manager needs to have some idea of how many people we're going to need; how long is this going to take; what kind of facilities will they need? And management has to provide the direction: What are we trying to accomplish; what are the goals?

But the teams now are responsible for defining the tasks to achieve those goals; for planning the tasks; for scheduling; for tracking the work.



**053 Now we can see here that measurement now supports a feedback loop. You're going to define activities, you're going to measure the activities, and you're going to determine how do these measurements compare to what we actually expected?

Measurement Used to Manage

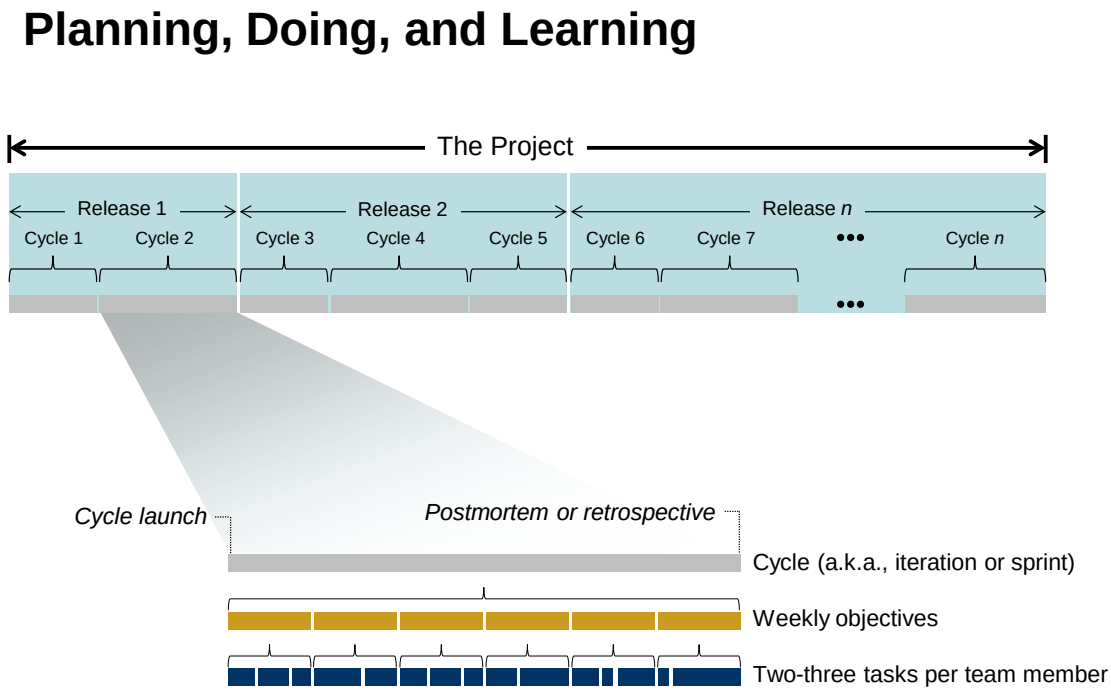


**054 Now meaningful feedback. Meaningful feedback requires measurement. Measurement is precise- precise and meaningful feedback against your expectations. That is, if you don't have some expectation going in, you can't be surprised. You can't- if you can't be surprised you can't learn anything.

Now it also requires that you define your process and you have enough consistency in what you are doing that the measurements are at least comparable; that you're comparing likes to likes. The outcomes don't have to be the same but they have to be similar enough that the comparisons are meaningful.

Well you can take these measurements at a number of different points in the process.

Planning, Doing, and Learning



**055 And that's what we're talking about here.

In a typical development process you will have release cycles. You're going to have multiple releases of a product very commonly.

Within the releases however, you're often going to have development cycles; you know, pre-release you might have potentially shippable increments.

Within the cycles you're going to have individual components that you are building and maybe integrating into the system. Drive this all the way down and you have work that you're performing week to week, day to day. You can measure at each of these points and mark your progress.

Comparing Estimates to Actuals

Comparing Estimates to Actuals

For both schedule and quality …

Is the project on track?

**056 Now as I said, it's essential to have estimates with which you can compare to actual measures.

Comparing Estimates to Actuals

Closed loop feedback

For both schedule and quality …

Is the project on track?

**057 This provides you a closed feedback loop. Your estimate provides you a sort of set point. Your actual is what is actually happening.

Comparing Estimates to Actuals

For both schedule and quality …

Closed loop feedback

Learning

Is the project on track?

**058 This is critical to the learning process. People learn by comparing what actually happened to what they expect; because when they get deviations from their expectation, they know that they can change their performance in certain ways by changing their behaviors.

Comparing Estimates to Actuals

For both schedule and quality …

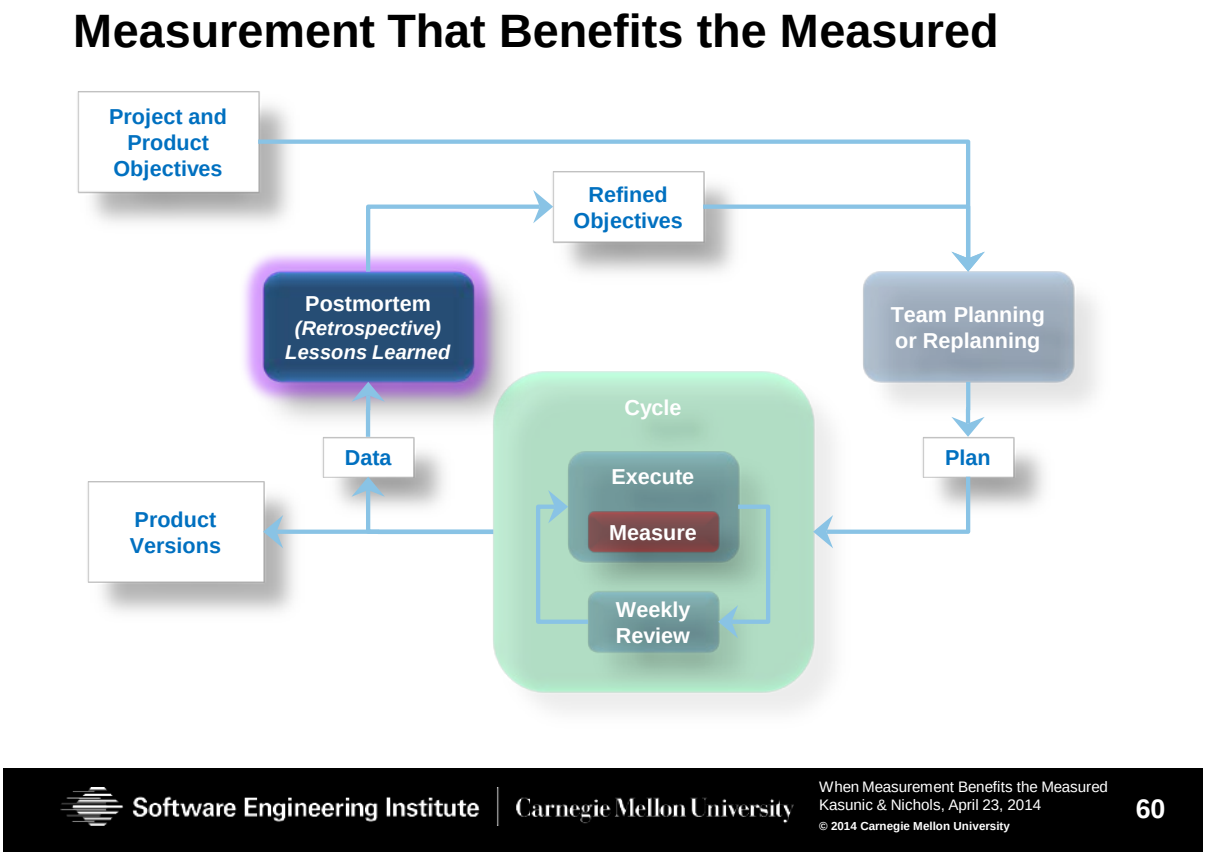
Closed loop feedback

Learning

Performance improvement

Is the project on track?

**059 And this is what leads to performance improvement. This is how people achieve expert- expertise and mastery of a subject.



**060 I will sum that up. We have an overall system; and we can measure it. We can estimate at the beginning. We can measure at different points; and we can do comparisons to see: Are we on track; have we achieved our objectives?

Again … Only Four Basic Measures Needed

Software engineers only need to collect four basic measures to mange their schedule performance and the quality of their work.

Time on Task

Size

Defects

Schedule

**061 Now the good news is, as Mark told us, we only need a few simple measures to do all of this.

If we measure the time on task. Think of time on task as the stopwatch time, when you actually have like fingers to keyboard, doing certain types of activities.

The Schedule is the big picture.

Milestones: When did something actually get done; time on the calendar.

The Size: How big was it? Was this a little piece of work, like a page, a sheet of paper? Was this a big piece of work like a book?

Defects: What did I have to go back and change after I thought it was done? What did I do incorrectly or wrong, in a way that's going to affect my customer?

Derived From the Four Basic Measures

Derived From the Four Basic Measures

Many other useful measures and indicators can be easily derived from the four basic measures including:

- estimation accuracy*
- prediction intervals*
- time in phase distribution
- defect injection distribution
- defect removal distribution
- productivity
- reuse percentage
- cost performance index
- planned value
- earned value
- predicted earned value
- defect density
- defect density by phase
- defect removal rate by phase
- defect removal leverage
- review rates
- process yield
- phase yield
- failure cost of quality (COQ)
- appraisal COQ
- appraisal/failure COQ ratio

* Both size and time

**062 From these measures, from these four simple measures, we can derive a number of other meaningful measures that can be applied at different points in the process. I'm not going to discuss all of these right now. But just four simple measures open up a world of other useful metrics.

How Do You Know If It's a *Best Practice?*

Organizations want a way to gauge their performance and to compare their performance with others in their industry.

Data on project performance is needed to provide evidence of what (exactly) constitutes a best practice.

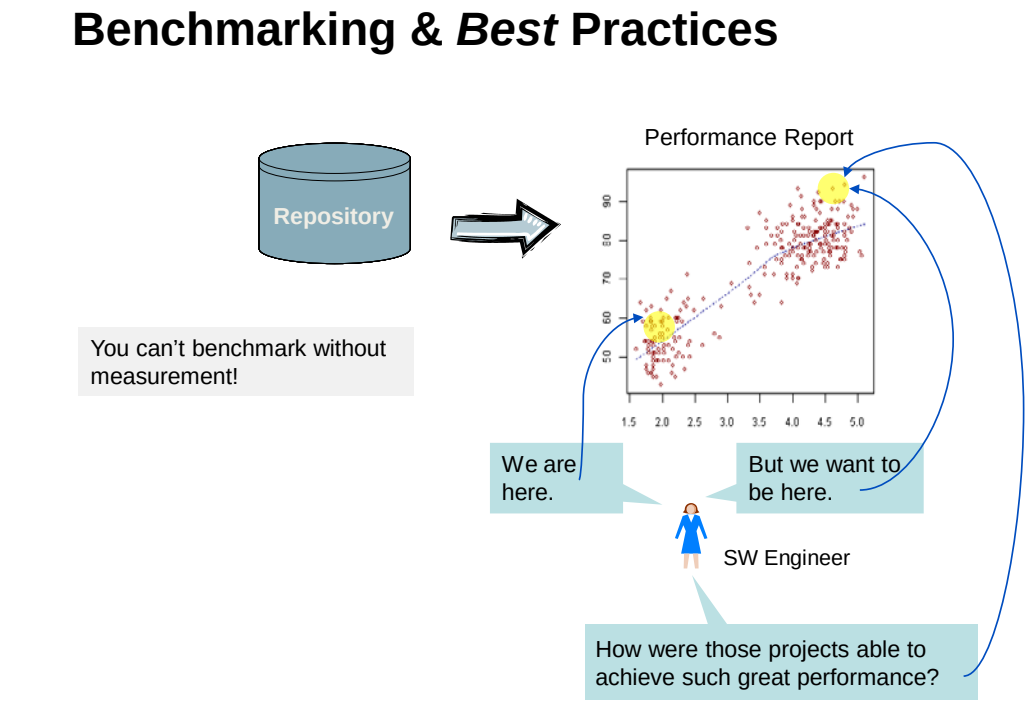
How do you even know what a best practice is unless you measure and compare it to other practices?

Benchmarks provide a reference point for interpreting performance.

**063 Now what are you going to use them for? Ask questions.

How are you going to do your work? What is a best practice? How do you know it's a best practice?

Well best practices should do what? They should get you good results. Compared to what? Compared to what other people are achieving.



**064 And that's what I want to share with you a little bit today, some of the results that we have collected over the last several years.

Description of the Data

The data was submitted to the SEI between 2000 and 2012.

The source data is from 93 projects in the United States and 20 projects from Mexico.

This is data that has been aggregated at the *team* level at the time of a cycle postmortem (retrospective).

Only data from a project's last postmortem is included.

Tests were conducted to ensure that extracted data represented unique projects.

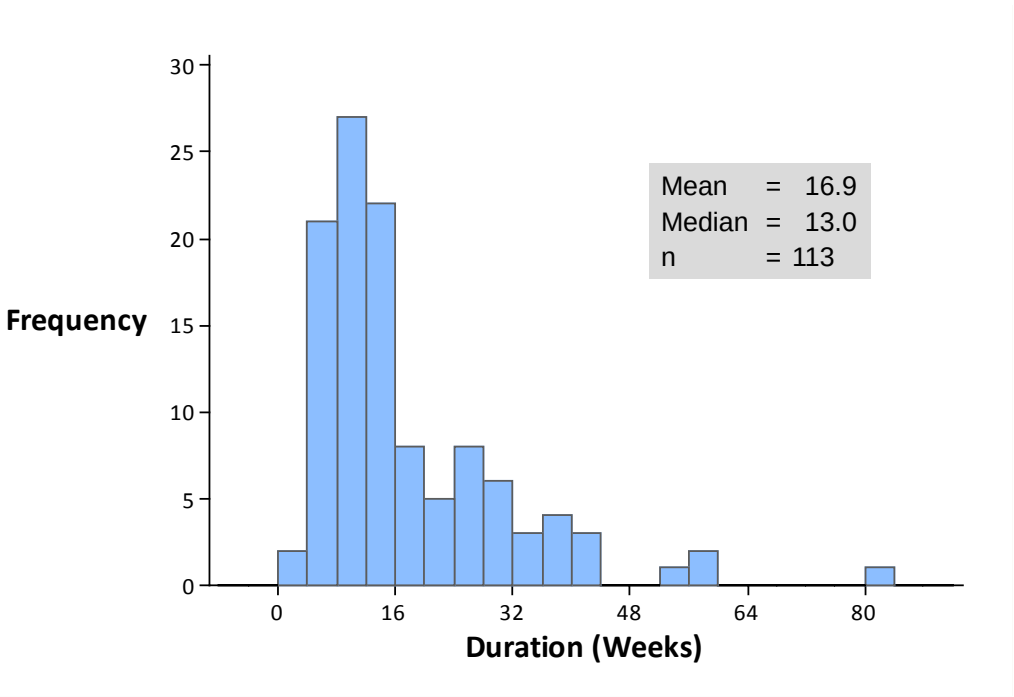
**065 Our partners have been gracious enough to share with us with the data that they have been collecting since about the Year 2000. This particular dataset includes data collected from roughly 2000 to 2012.

We have in this set about 90- 93 sets of projects or portions of projects from primarily the United States and Mexico. I'd be surprised if we don't have a few other projects in there as well; because we have some active teams in South Africa and Japan, Korea. But this particular set may be limited.

We have tried to narrow the scope on these projects, the projects that we know completed, for which we have postmortem data; and then we put in a few tests to make sure that we didn't double count, that there are no duplicates.

This data was recorded by people, as Mark suggested, taking their own data-- for their own purposes tracking their projects-- and then sharing with us when they were done; so that we can aggregate the results and give the community results back to them.

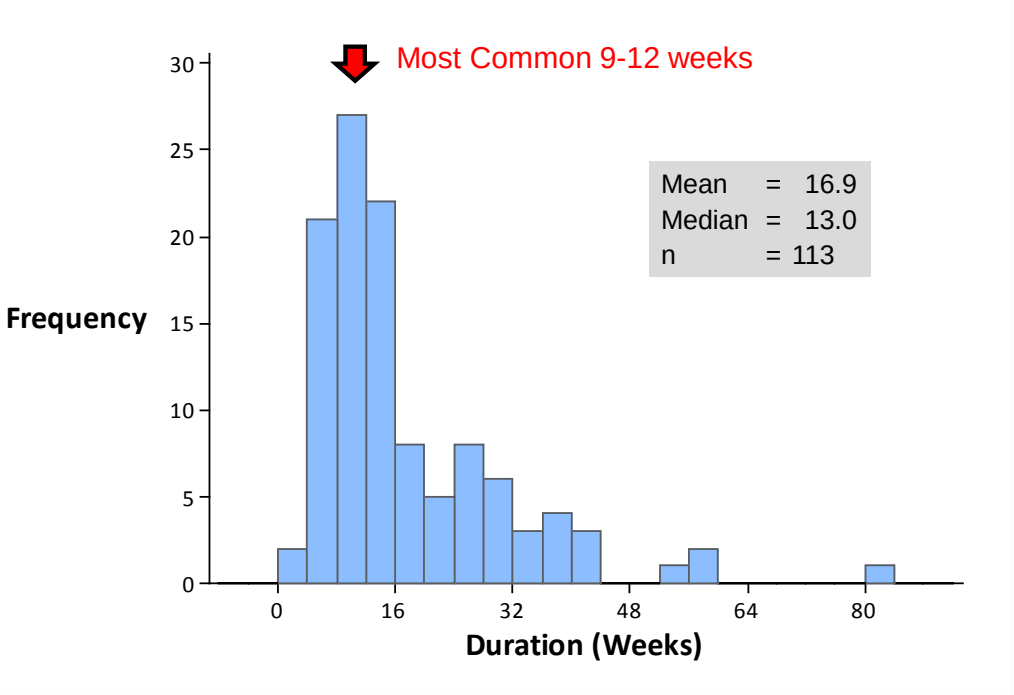
How Long Were Project Durations? [Weeks]



**066 So let's talk a little bit about the demographics of these projects.

How long were they? Well how long should normal projects be? How long can your planning horizons be?

What were Project Durations? [Weeks]

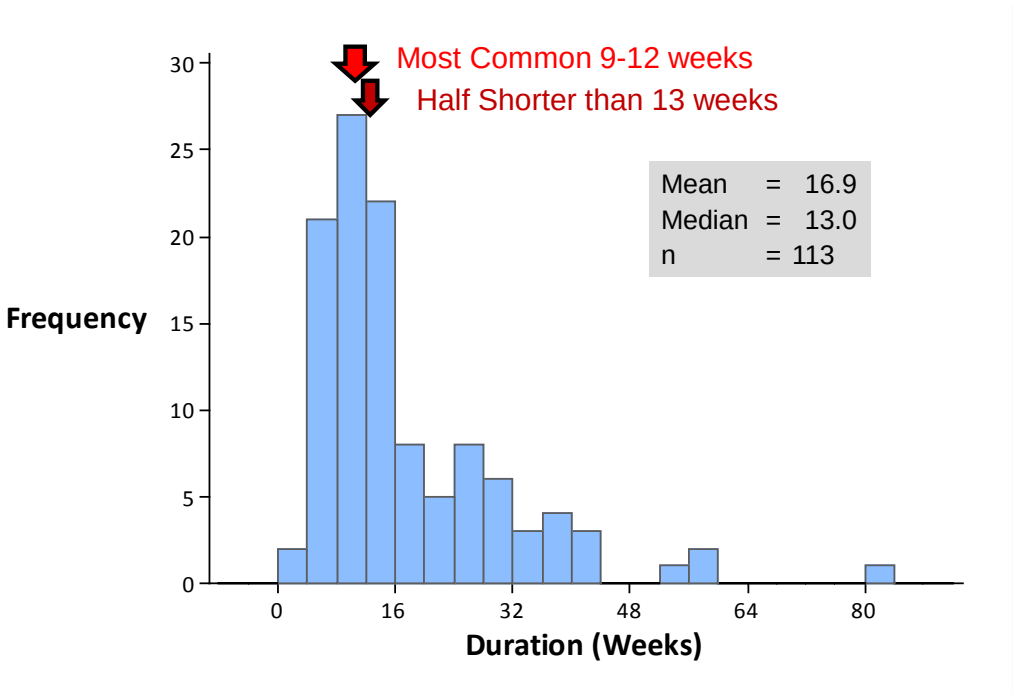


**067 We found that among our customers, the most common planning- planning span was about 12 to nine weeks.

Now think about that. That's a little under three months. That's about how long they thought they could make a credible plan for.

And most of these projects were relatively short duration that would be completed in a single cycle. Now that's not all of them.

What were Project Durations? [Weeks]

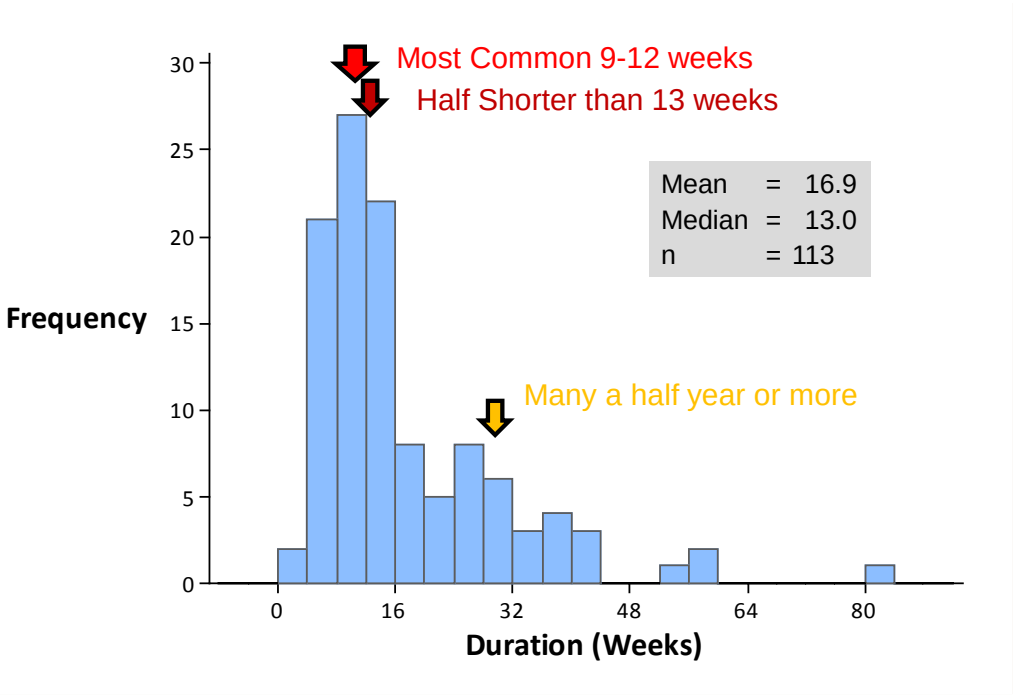


**068 We found that about half the projects were shorter than 13 weeks; 13 weeks or less.

What is 13 weeks? Turns out that is three months; it's about a quarter year.

Half the projects were longer than that.

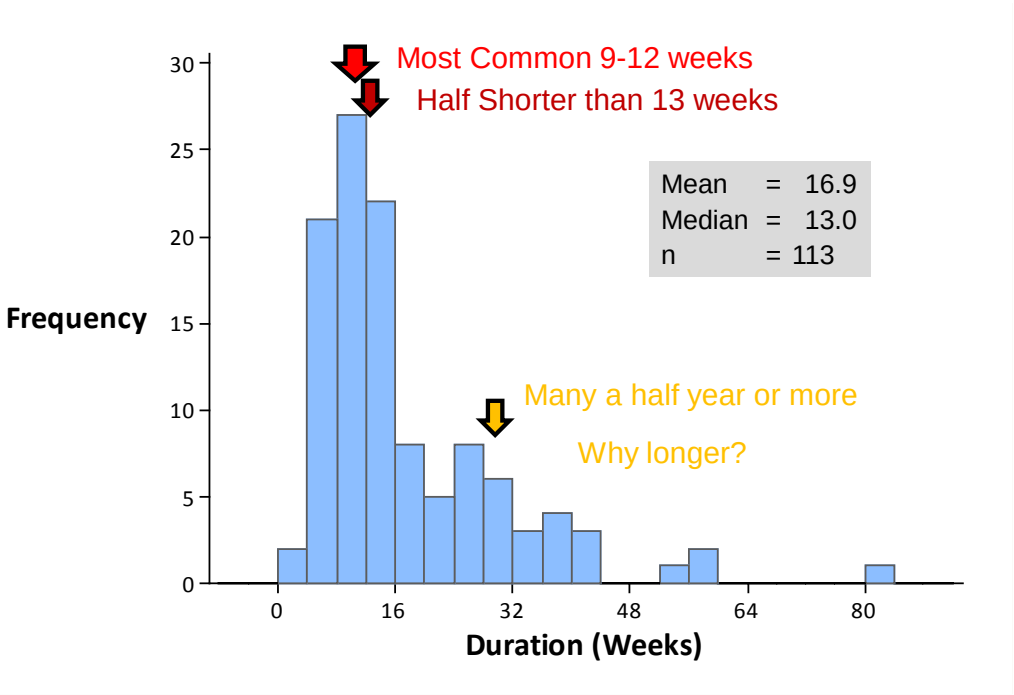
What were Project Durations? [Weeks]



**069 In fact, a significant number of them were more- were a half year or longer.

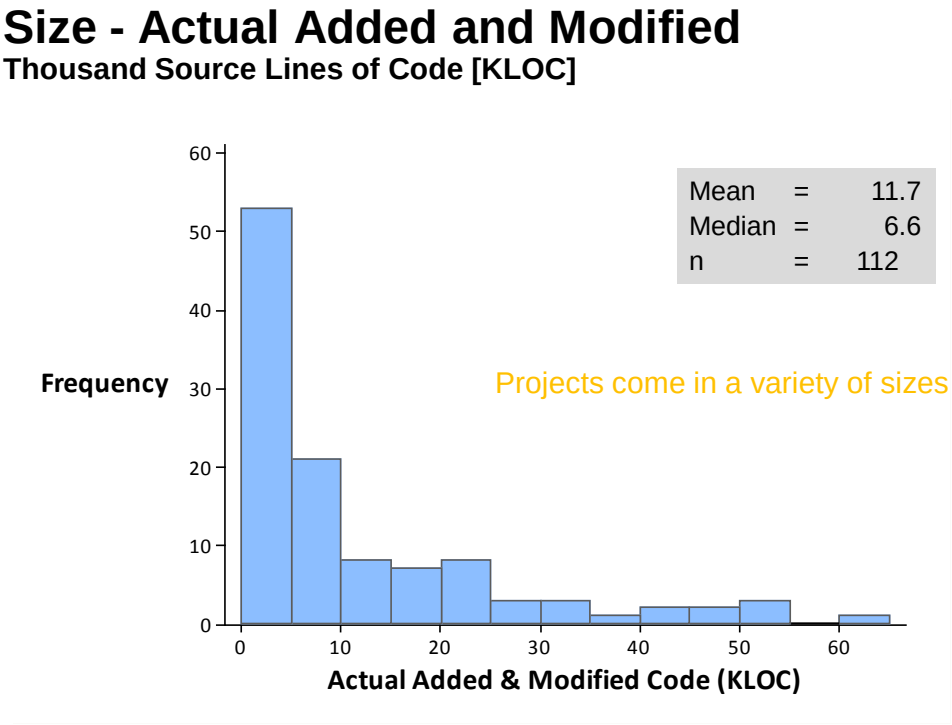
Why do you need longer to build a project?

What were Project Durations? [Weeks]



**070

Size - Actual Added and Modified



**071 Well one of the answers might be the size.

Now we'll use lines of code to talk about project size. It's only one measure; and it's not a perfect measure. But lines of code; because people have to build them. These are only the lines of code that people actually- actually make themselves.

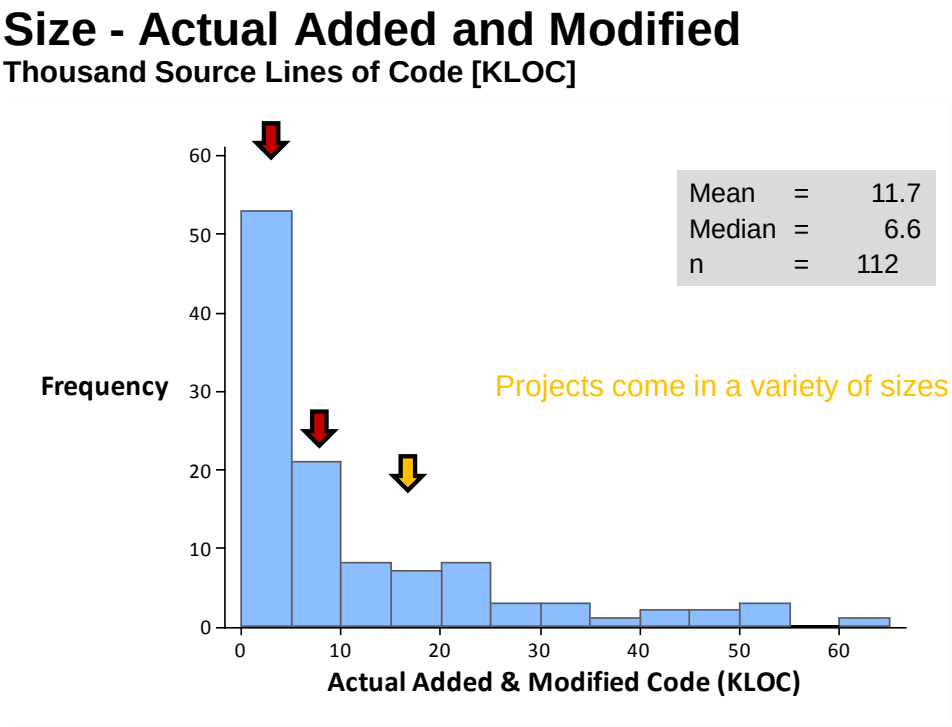
That tends to correlate with effort. You're limited by how fast you can create code.

So it's a mix of different programing languages, different environments. We're seeing global industry

benchmarks; not local numbers- not individual local numbers. This is a collection of all those local numbers.

So take that with a little bit of caution. But what we have is a distribution of project sizes measured in lines of code.

Size - Actual Added and Modified



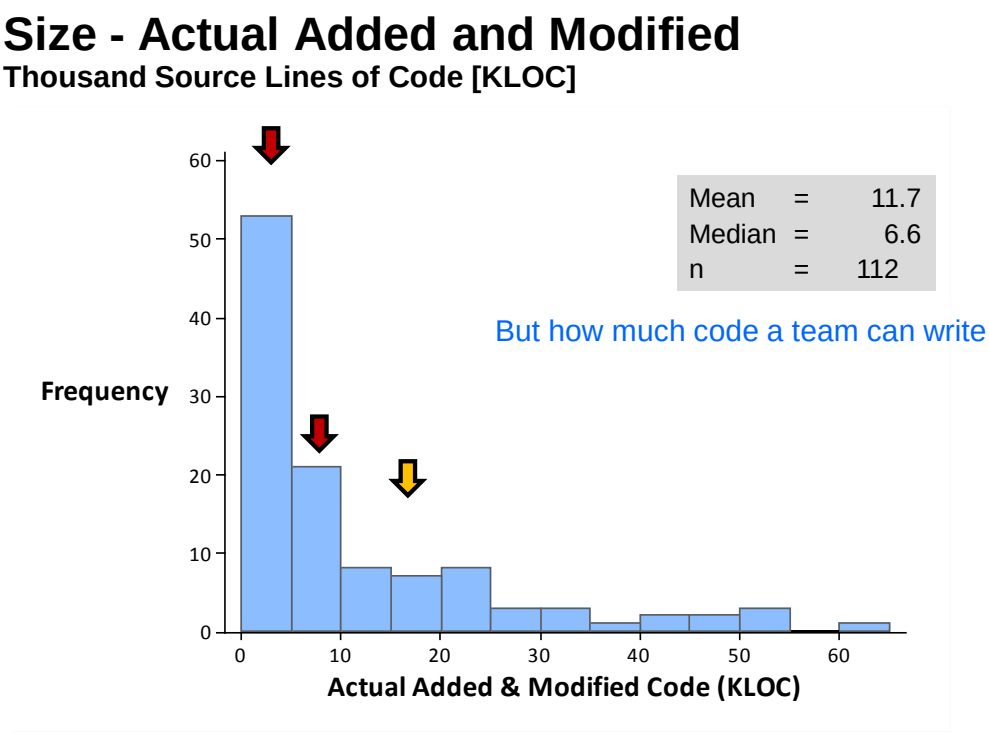
**072 Now what did we find? The median, that is half of these projects, were 6.6-- and this is in thousands of lines of code-- the median and the mode were both in this short, thin here.

Six.6 thousand lines of code. That's not all that big. It would be considered a fairly small project.

To put that into scale, that's probably about what one individual could do somewhere in the year- in the year timespan; on the upper end of a year maybe. If you're writing new code-- I've seen people do 10,000 or more. But that's about a- that's about a man-year of effort.

The mean, the average, was twice that. So we're at about 12,000 lines of code. The average project was actually a couple of man-years of effort.

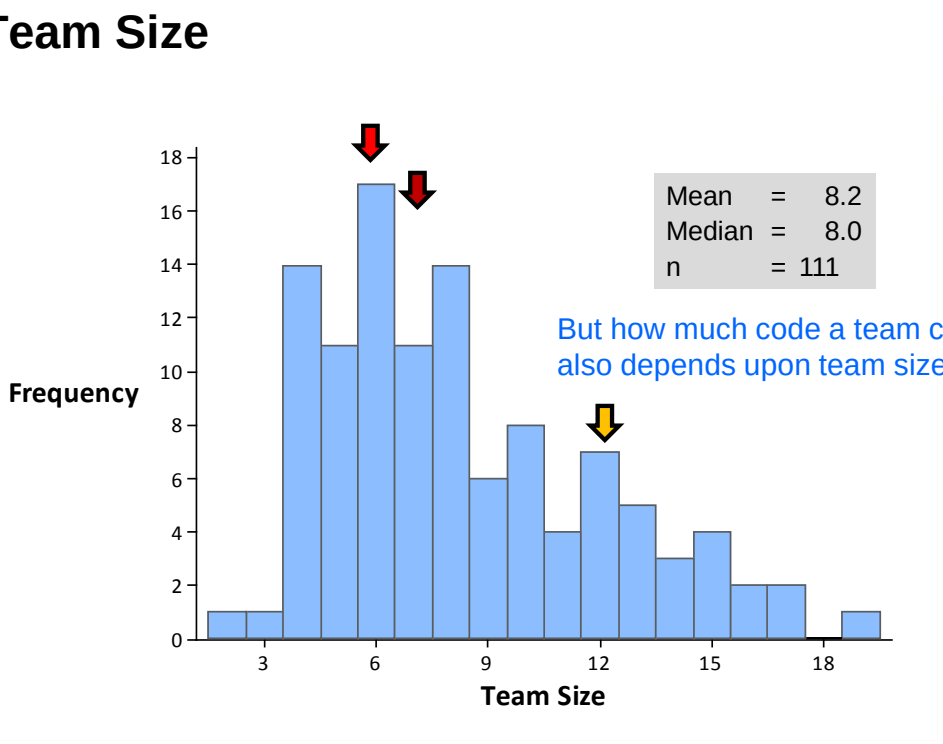
Size - Actual Added and Modified



**073 And we can see that there's a substantial number that come out much, much longer.

Now how much can one team actually write? So we have-- these are the durations. This is the size of the projects. How are you going to staff it?

Team Size



**074 How many people is this going to take?

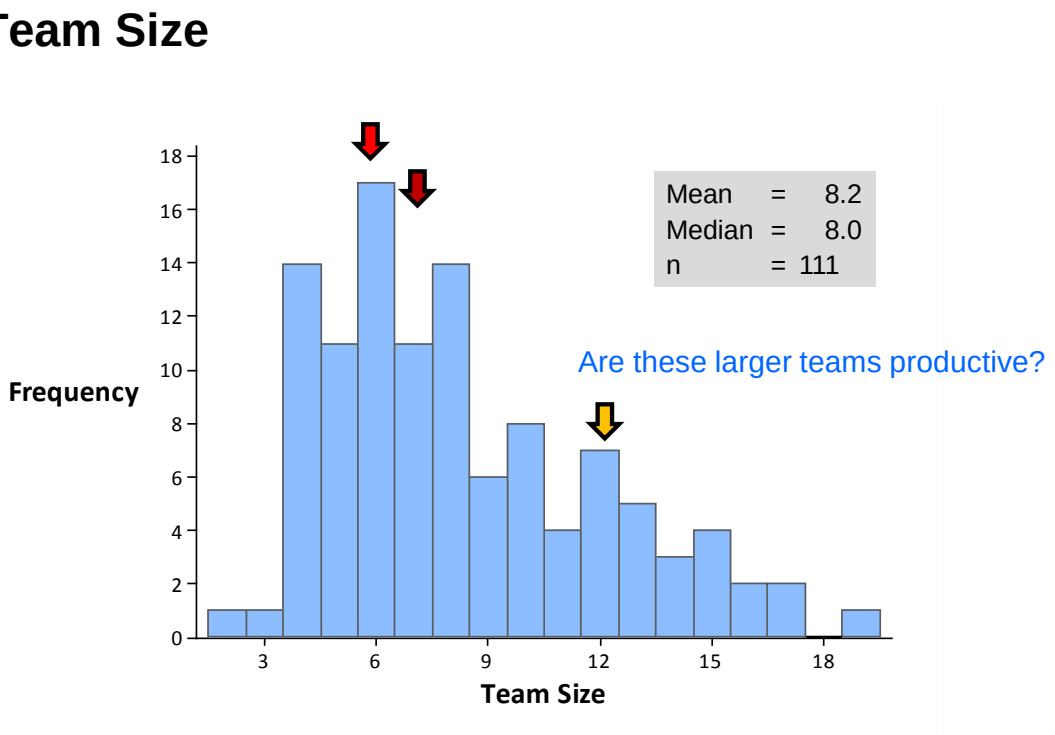
Well let's take a look at team sizes. The most common team size that we observed was six people; a very small team.

Many of you have probably heard some recommendations on seven plus or minus two is the ideal time sizes.

Do people really behave as though that's true? Well the most common team sizes in that range, six. The mean size, the average, is on the upper end of that at eight. But about half the teams-- a large number anyway, about a third of them-- are much longer- are much larger, peaking at about 19.

So we have a lot of fairly sizable teams out there that people have- that people have been trying to organize.

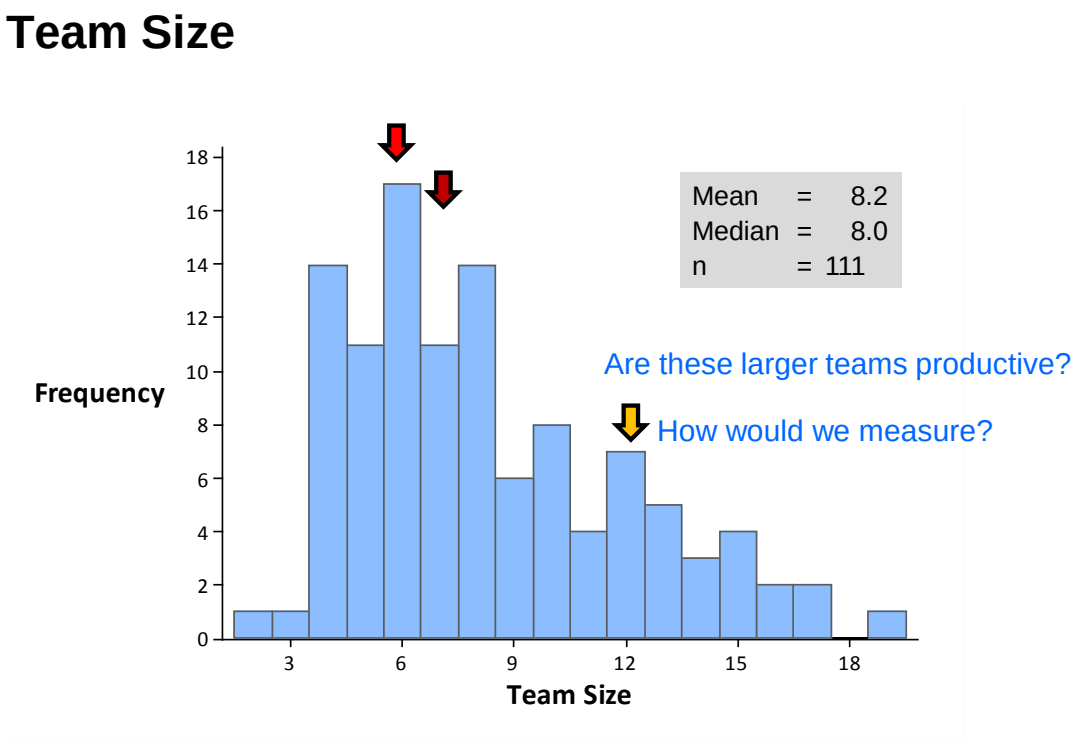
Team Size



**075 So it's a fair question: Does the rule of seven plus or minus two really make any sense here? Are

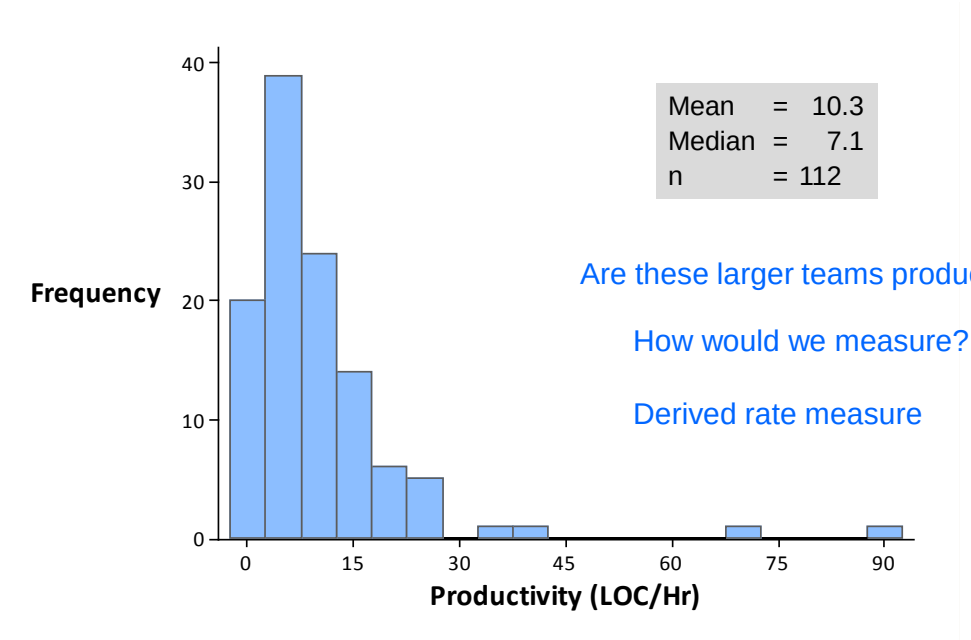
these people hurting themselves by putting together bigger teams to try to get these projects done- bigger projects done in shorter amounts of time? How would you know? What kind of question could you ask?

Team Size



**076 Well we have some benchmark data that might help you think about that.

Code Production Rate



**077 What would you measure? Well one measure would be a code production rate. How much code do people actually write? And this is the variance. We found a pretty substantial variation in how much code people are able to write in a given time. So we're going to measure this in lines of code per hour.

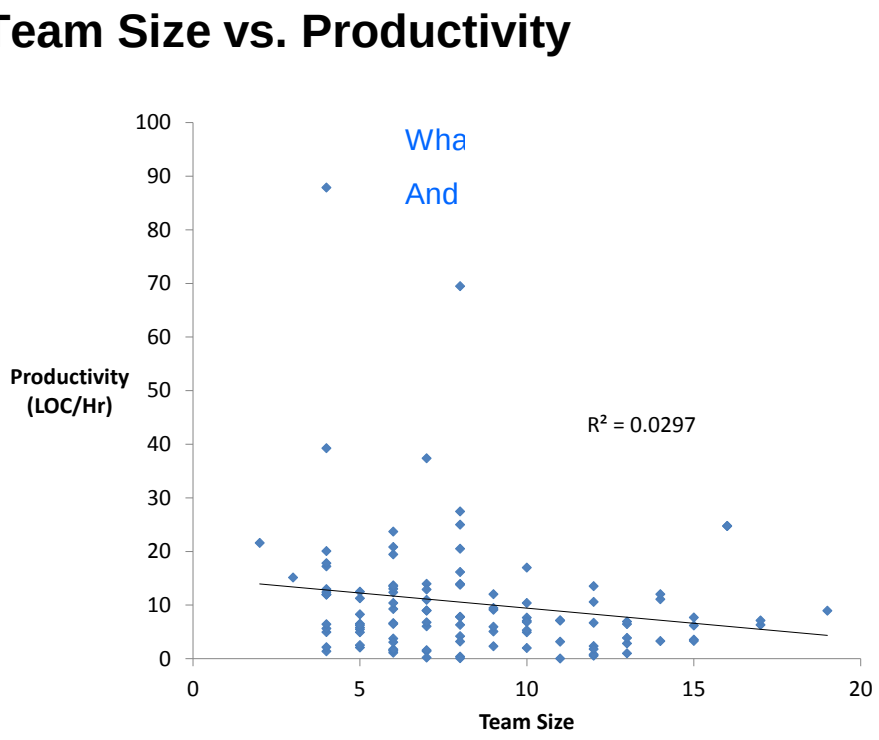
So what is a reasonable number for lines of code per hour? It turns out the average is about 10; just industry benchmark, 10 lines of code per hour. That's presumably from primarily in the development phases; but it could be from requirements through testing.

The median was 7. So if you're organizing a team, and you don't have any other information, this is a good place to start; if you have some idea of how big the project is.

This is an average value. But here are the span of ranges. We see some that are much slower, much faster- much slower and some that are much, much faster.

What does your data look like in your environment? What kind of ranges can you expect? Here is some benchmark data.

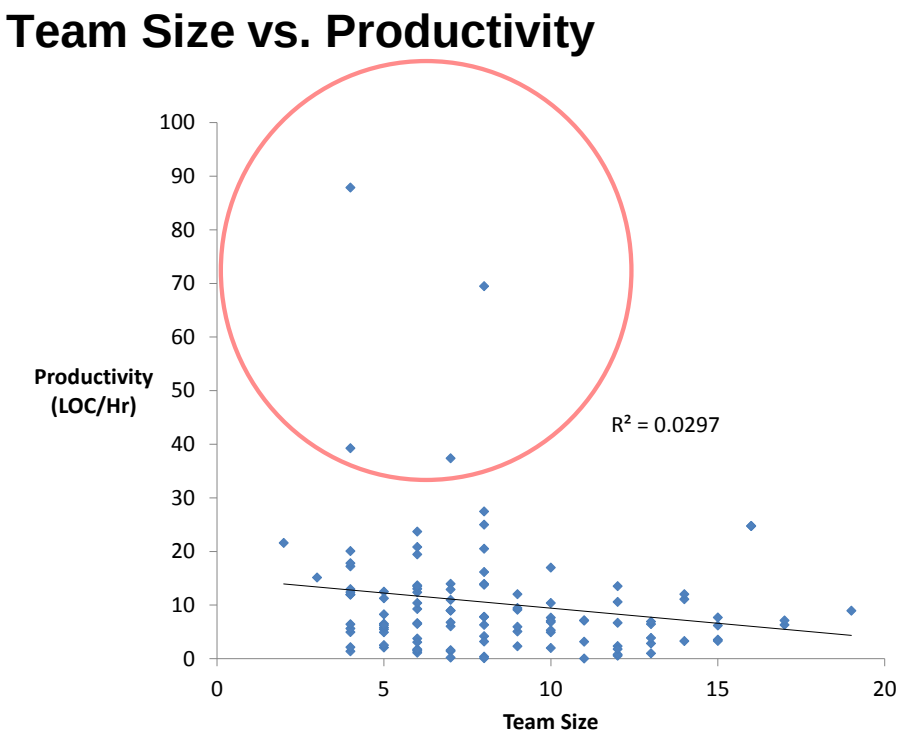
Team Size vs. Productivity



**078 But let's get back to the question. Do large teams slow you down? If so, how much?

We did a scatter plot of team size along the X axis versus the code production rate along the Y axis. So each point represents one team and the rate at which that team was able to build code. We have the team size along this axis.

Team Size vs. Productivity



**079 Let's look at the data a little.

We have a few very high productivity points. I kind of suspect some of that data might not be counted right. It's possible that some teams were counting automatically generated lines of code; or they had a very small project where they were able to build out something like html code very quickly.

Those sorts of outliers don't happen with the larger teams. So I suspect these are smaller teams, certain classes of code, or some mistakes.

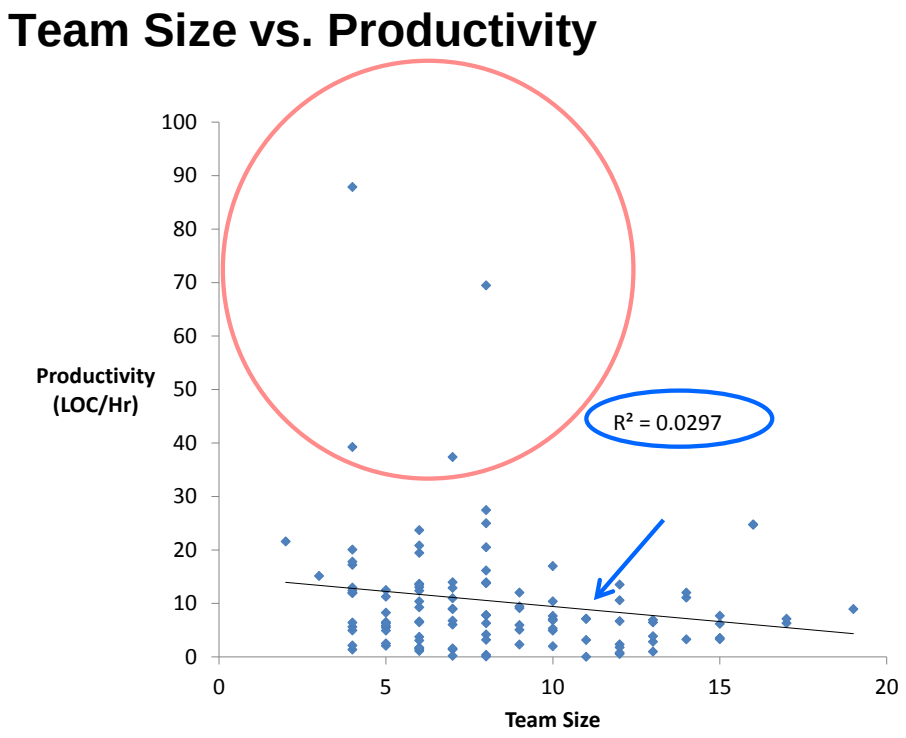
But even with those accounted for, we did a regression fit; and we have a regression line, an average line, a best fit, that looks like it does slope downward a little bit, and we might have lower productivity.

But what do these statistics say? The R-square, the correlation coefficient, is a measure of how predictable- how much predictability this line gives us. R-squared of 0.03 says 3 percent of the variation in production can be explained by the team size. So the answer is 3 percent; not very predictive.

Whatever these teams are doing, the larger teams are more or less as productive as the smaller teams per person.

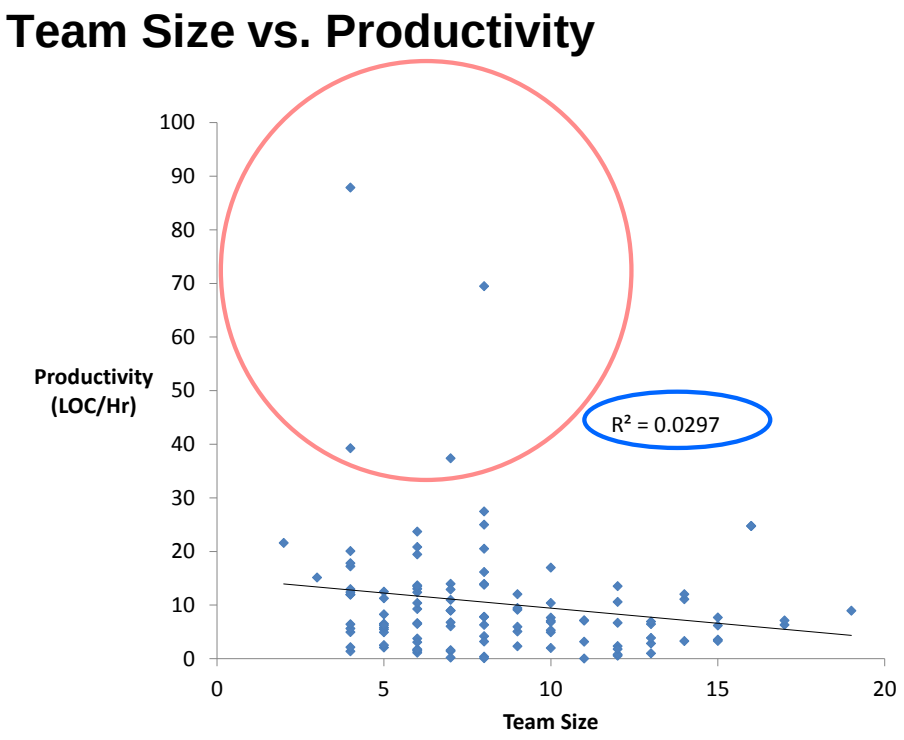
So there you have it. That is the answer from what is actually happening in the field from this data.

Team Size vs. Productivity



**080

Team Size vs. Productivity



**081 So what else? What else might matter?

Bill, any speculation what that answer will be?

Bill Nichols: Well task hours is finger to keyboard. I don't want to bias the people in their guesses because I know the answer.

Shane McGraw: Okay.

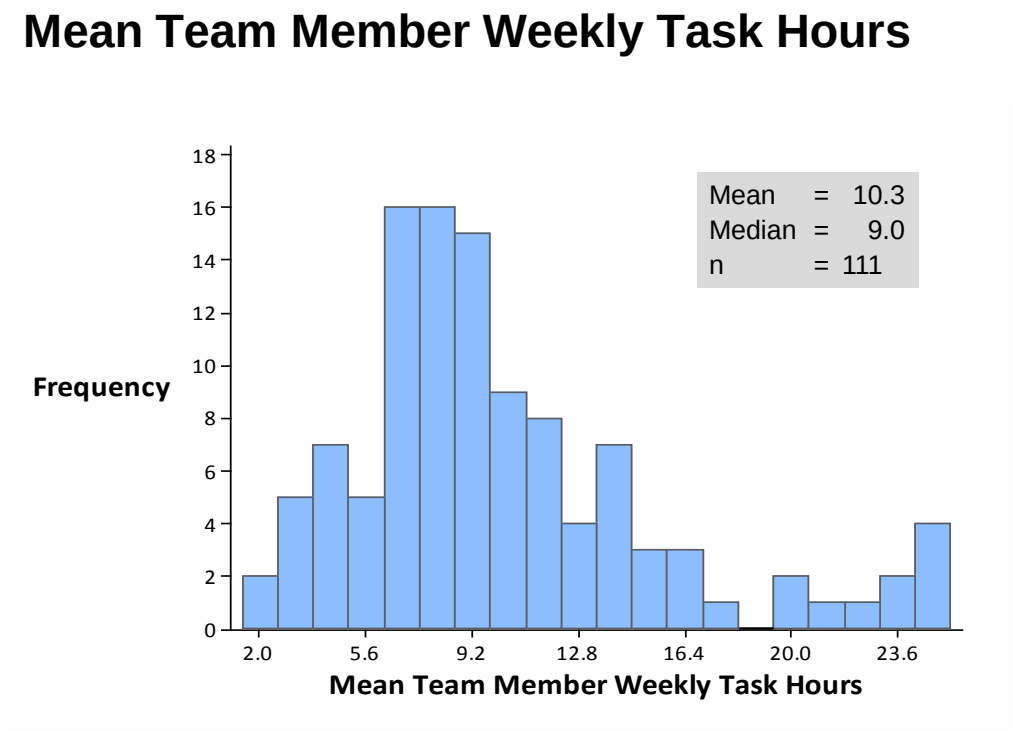
Bill Nichols: But how much time- the question is: How much time, fingers to keyboard, shoulder to the wheel, nose to the grindstone if you will, can you actually get on project work in a given week; let's say a 40-hour week?

Shane McGraw: And we will share it now with our audience. Let me pull it back up to read them off.

Thirty to 40 hours we have at 5 percent; 30 to 35 hours we have at 16 percent; 25 to 30 hours we have at 27 percent; 20 to 25 hours we have at 26 percent; 15 to 20 hours we have at 12 percent; and less than 15 hours we have 13 percent. So quite a mix.

Bill Nichols: Okay. Well some of you may not be surprised.

Mean Team Member Weekly Task Hours



**084 But I think a lot of you will be shocked at what the real data is telling us.

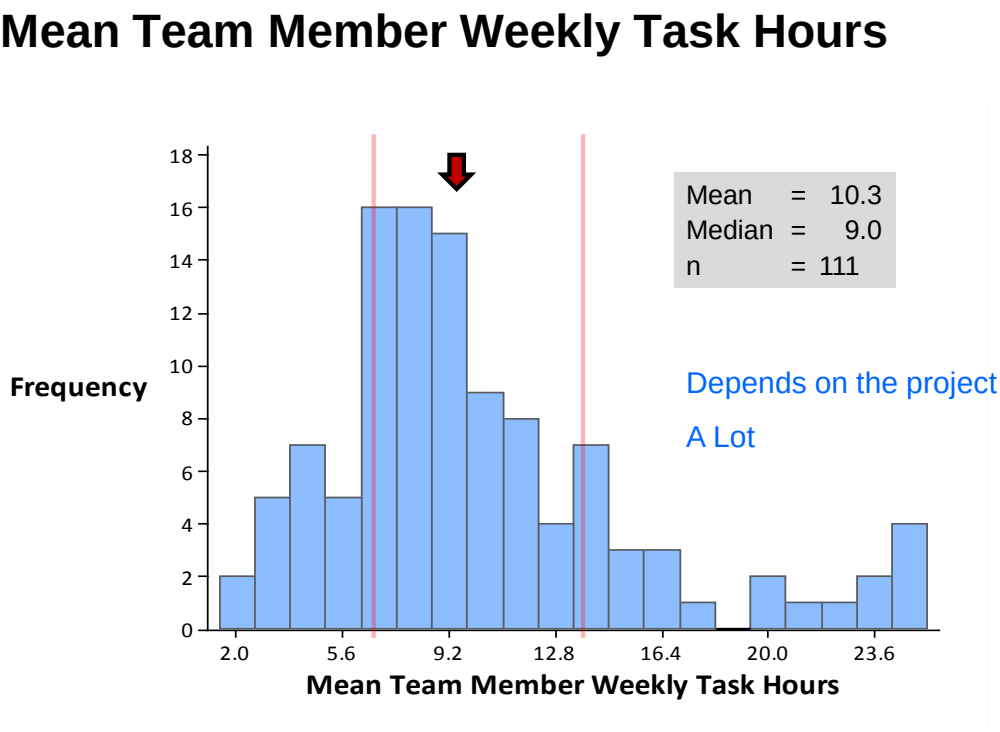
Where is the peak of this curve? This is my mean time, the average time per person, per week. And this is my count; how many? So of these hundred- let's see, 111 data points. We took the project. We took the hours, divided them by the number of people: How many hours per week did they really get?

The most common- the most common was between seven and nine hours per week, on task, on project. So less than one-quarter of the working week was actually on time- on task time.

Now the mean value. Not much difference from the median. They really are both clustered at around 9 to 10 hours.

We can see that there are a fair number in the 12 to 15 hour range; not so many. We're actually running quite a bit out of this.

Mean Team Member Weekly Task Hours



**085 Let me show you the lines. These two lines represent 50 percent of the data. Half the data is essentially between 6- 7 to 14 hours. That means that about a quarter of them are lower and another quarter are higher. Now think about that for a minute.

We can see that there are only a handful that are able to get 20 to-- 25 is the peak number here. Only a handful can achieve these sorts of on task hour performances.

Now I've been there myself and taken my own data. In order to achieve these sorts of numbers, you have to lock yourself in a room with almost no interruptions and work with shoulder to the wheel all the time. You would be amazed, even in those conditions, how difficult it is to approach 30 hours per week on task time.

What do you have to do? You have to do things like you have to spend a certain amount of time at the water cooler. You have to get up and stretch. You have to attend mandatory staff meetings. You might have to attend mandatory training.

By the time all is said and done, there are a lot of demands on your time. Unless you understand those demands, you can't manage the task time.

The problem is that task time is the only time that really helps you directly advance your project.

So what we find again and again repeatedly is when teams start falling behind, there's a desire sometimes to just move the goal post; well this is as fast as we can work.

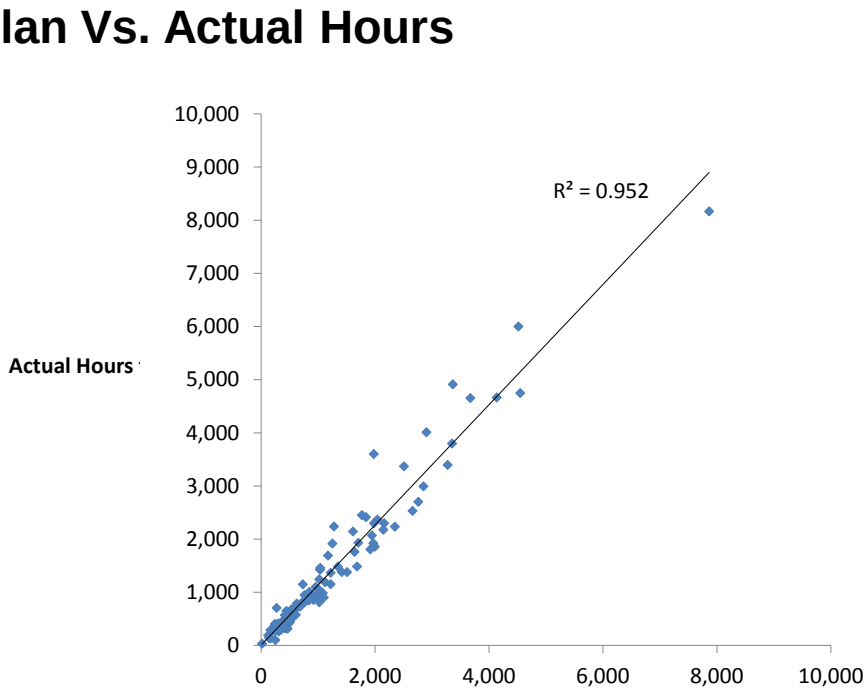
But if you understand where the time has gone, with a little time management you can actually get some help. And one of the things that's been very helpful is we've asked managers to do things like take the project team and put them in the back of the building, away from everyone; just isolate them for awhile.

What happens if you're working on two or more projects? That's a killer.

What do you have to do to be on a project? To be on a project you have to attend staff meetings. You have to communicate.

What we're typically finding is that if you're on a couple of projects, that's how you get down here. By the time you're on three projects, you're lucky to get any time at all to work at all. That's what the data says; and it's really kind of surprising.

Plan Vs. Actual Hours



**086 Now the good news-- and there is some good news in all of this-- is that if you've got a little historic data- if you have a little local historic data on what your environment is like, you can predict how many task hours you'll get. In fact you can even work to improve it.

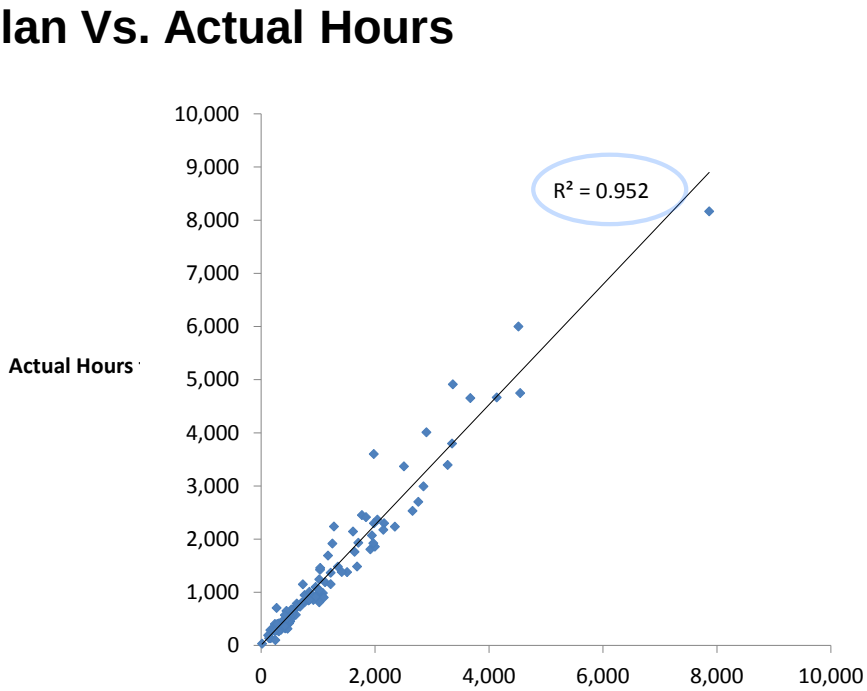
And you got to be careful about that because task hours sort of-- you aren't- task hours aren't your objective; but they give you a lot of insight into where the work is going, where the time is going.

This R-squared is the correlation coefficient for this regression line. R-

square: 95 percent. For teams that have measured their time, have some historic data and have made some projections and managed their time, their actual effort has a 95 percent correspondence to their planned effort.

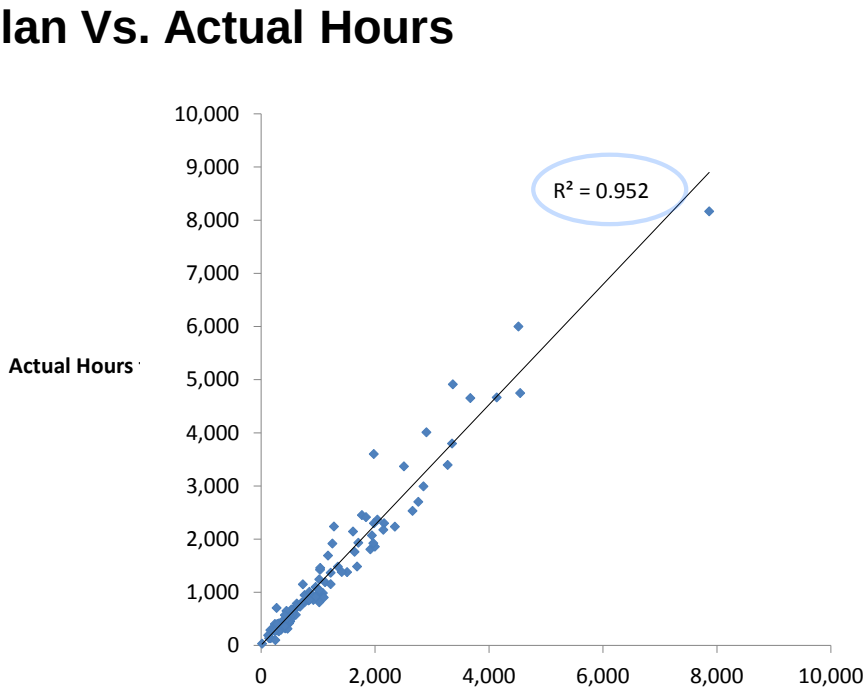
So they bring this under control. So that gives you a whole new dimension with personal measurement of how you can take control of your schedule and stop being tossed about by the winds.

Plan Vs. Actual Hours



**087

Plan Vs. Actual Hours



**088 So the sum of that is the actual hours really go into your project; and you do have some ways of measuring and controlling them.

Let's Looks at Some Quality-Based Profiles

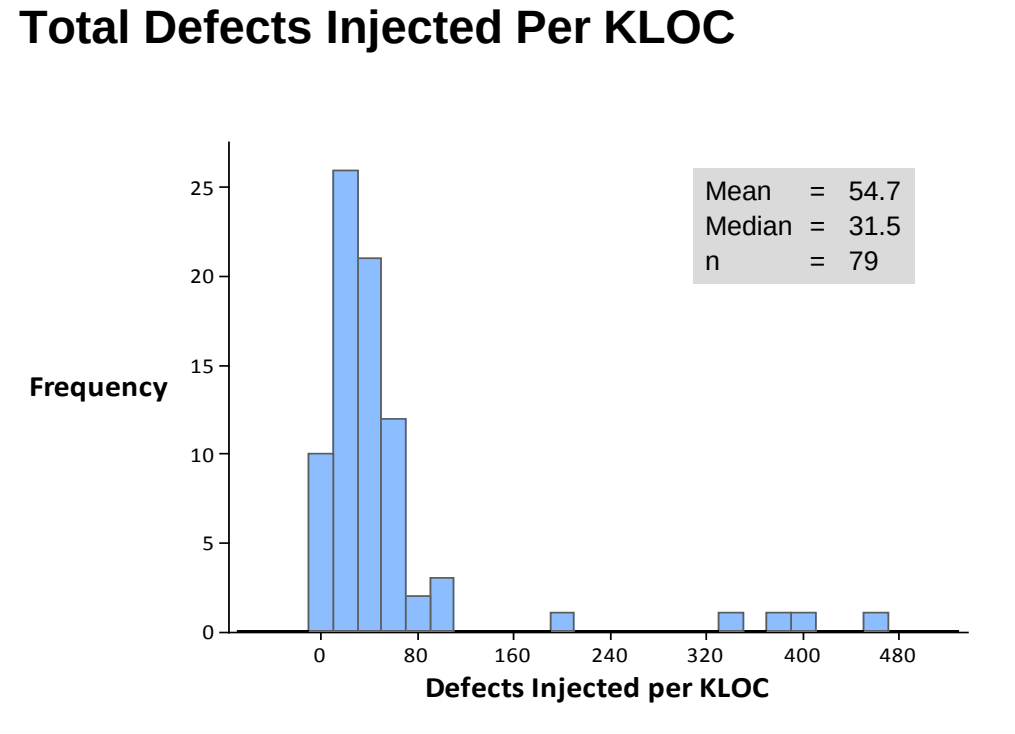
Let’s Looks at Some Quality-Based Profiles

**089 Now that takes us to the issue of quality. And it looks like I do have enough time to go through some quality slides.

So I'm to talk a little bit about quality. One should never talk about schedule or production without speaking to quality when you're talking about knowledge work.

Knowledge work is not a commodity. We're now talking about widgets coming off an assembly line. Every widget is different. That means you have to evaluate the quality of every widget. Was it correct? And if it wasn't correct, you have to go back and rework it.

Total Defects Injected Per KLOC



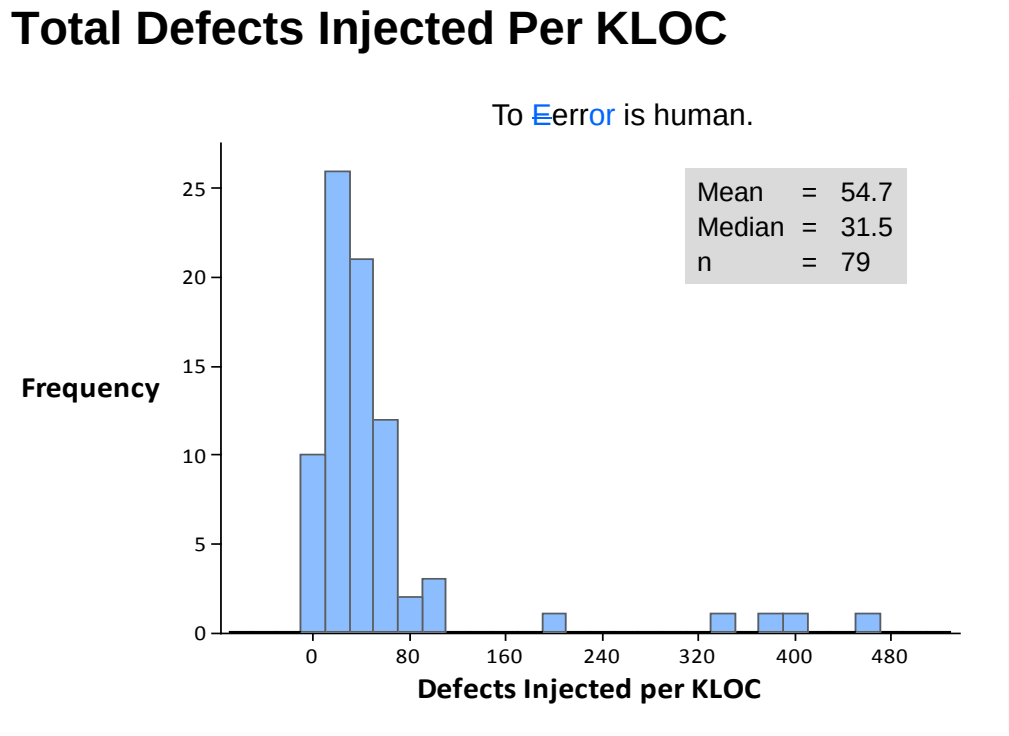
**090 Now what we find in the software world is you measure that primarily through defects.

Guess what the most consistent process in software engineering is? Injecting defects. Humans make mistakes. We make mistakes remarkably consistently. We do the same things over and over. If you're building a project, you are going to have defects.

Removing them, that's another matter. Removing defects is optional. You don't have to take them out until someone complains. But if you're building it, you have to inject them.

In fact, we have data here at how fast the different teams have injected defects; and we can see this range with a peak of about 50- about 55-55 defects per thousand lines of code.

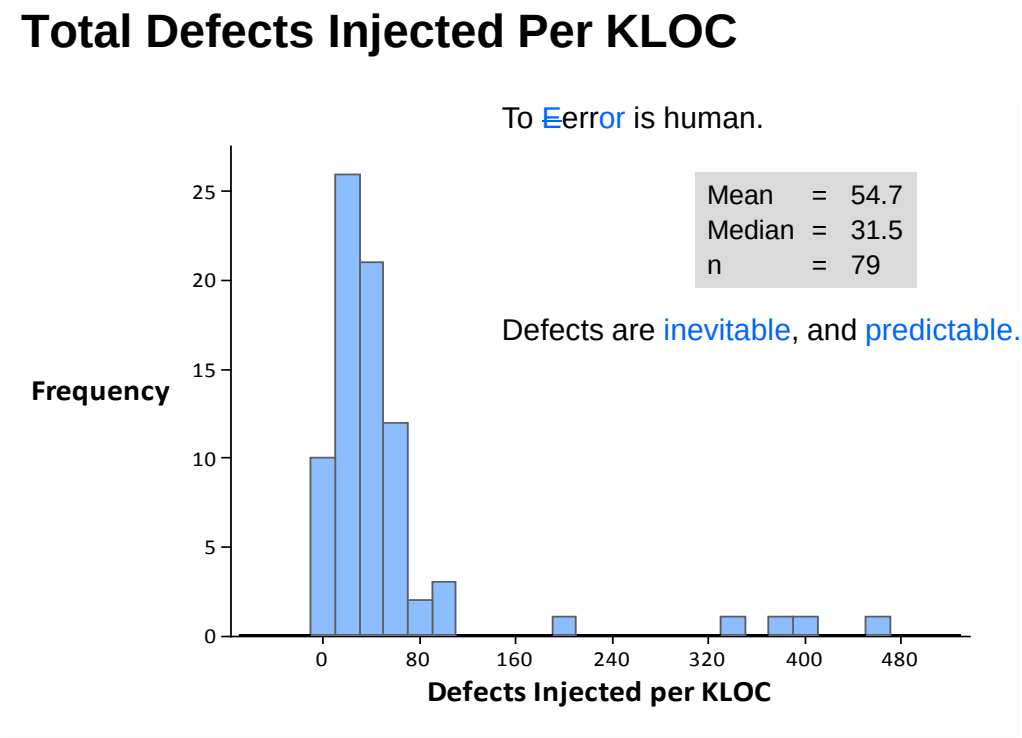
Total Defects Injected Per KLOC



**091 We can do that also in defects per hour.

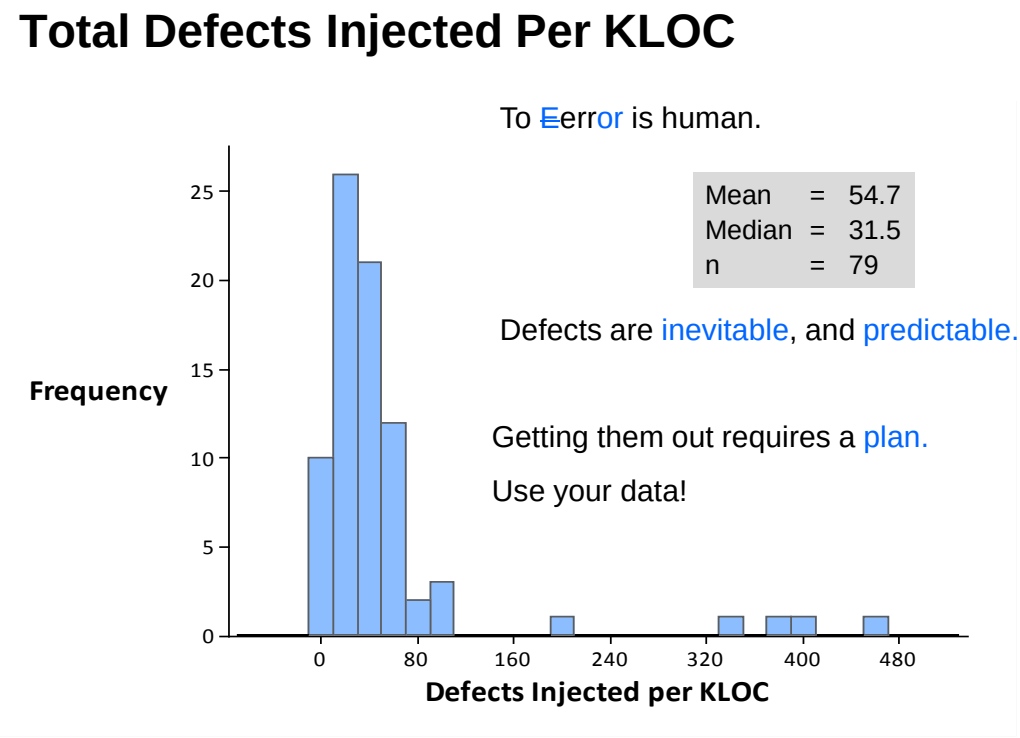
So to err is human. You will make defects.

Total Defects Injected Per KLOC



**092 Live with it.

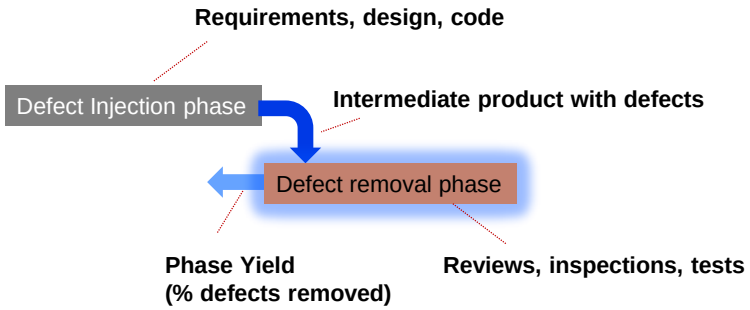
Total Defects Injected Per KLOC



**093 But defects are predictable.

Now you must make a plan. The reason you need to have personal plans. When you know what your defect injection rates are, you have to make a plan to get them out. If you don't plan to get them out, they aren't going to get themselves out.

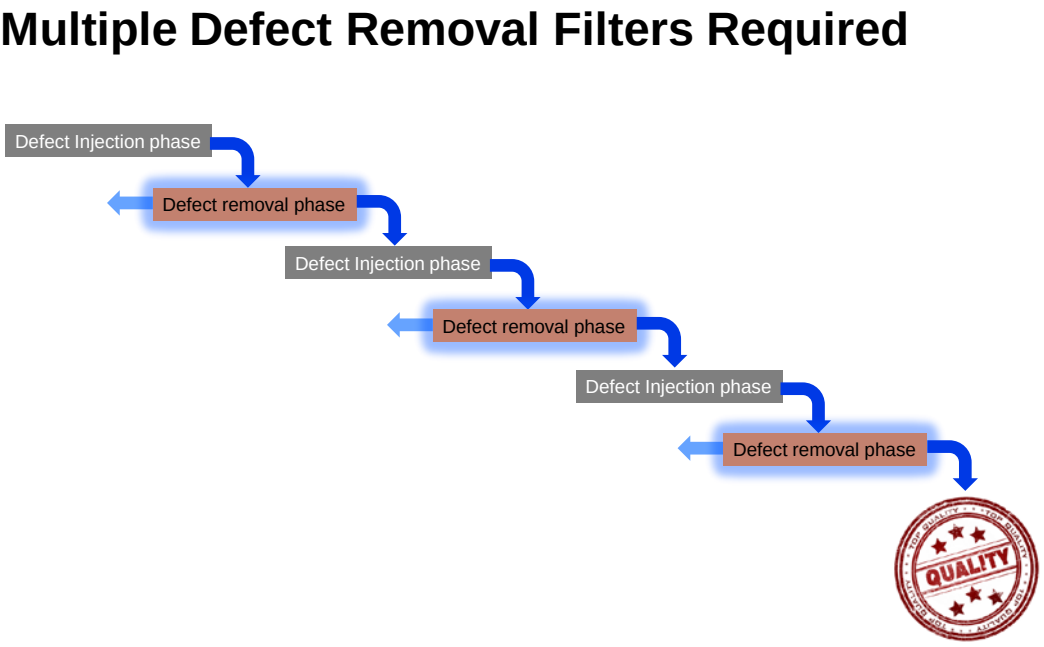
Injection and Removal of Defects



**094 That's why we have creation phases. We're creating product. Separately we're going to have phases to remove. We can remove defects by doing personal inspections or personal reviews. We can have peer reviews or inspection. We can run tests.

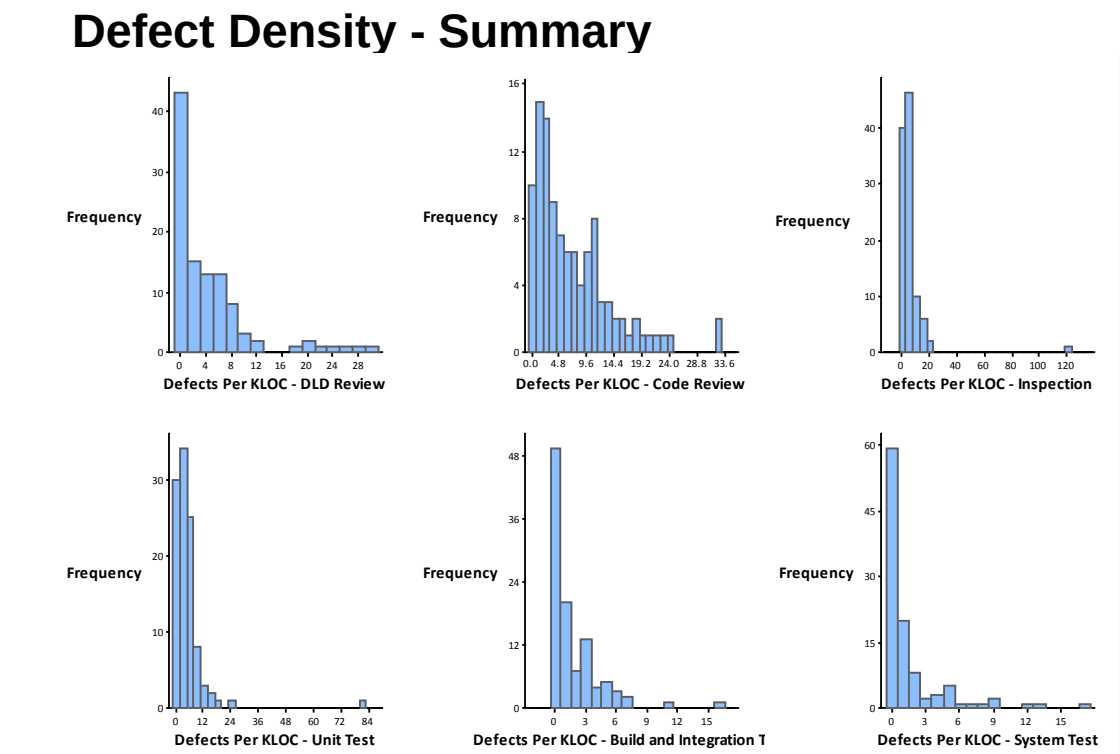
You have to plan for the removal phases.

Multiple Defect Removal Filters Required



**095 And you will have more than one. You'll have- you'll inject defects when you do requirements gathering, when you do designing, when you do coding. So they have to be followed by various types of removal phases.

Defect Density - Summary

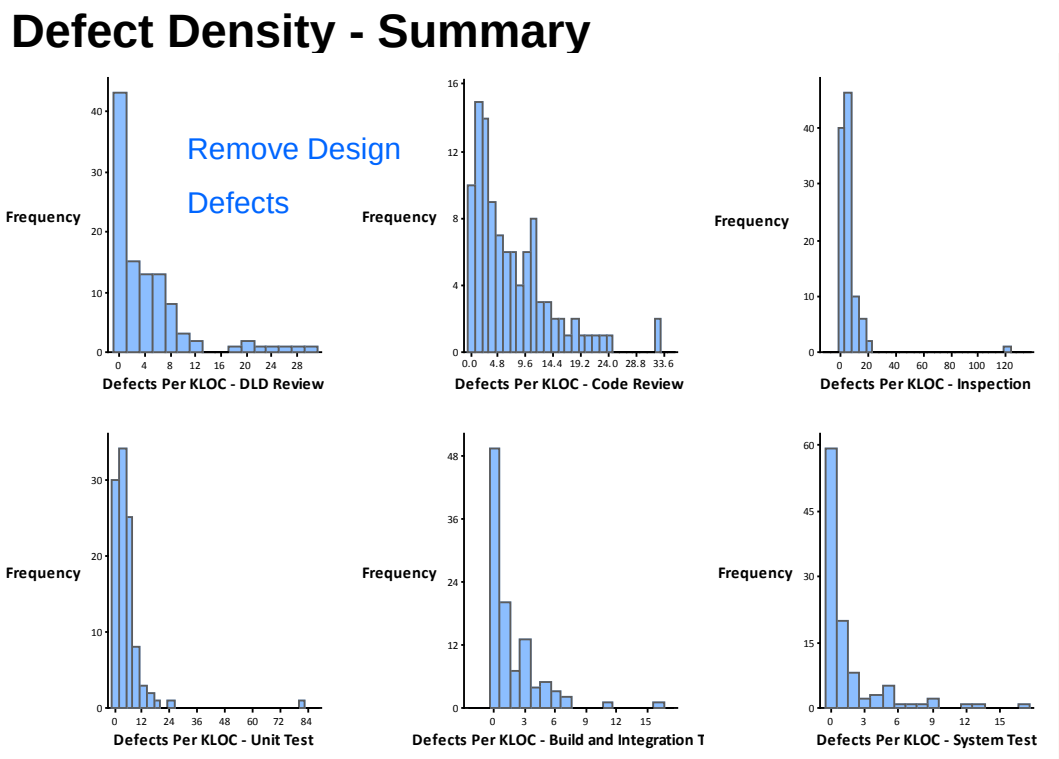


**096 And that's what I'm going to show here.

There are many other ways that we can examine the defects. But I'm going to look only at the defect densities. What are your expectations going in? What are the expectations about how many defects are going to be in your product at different times?

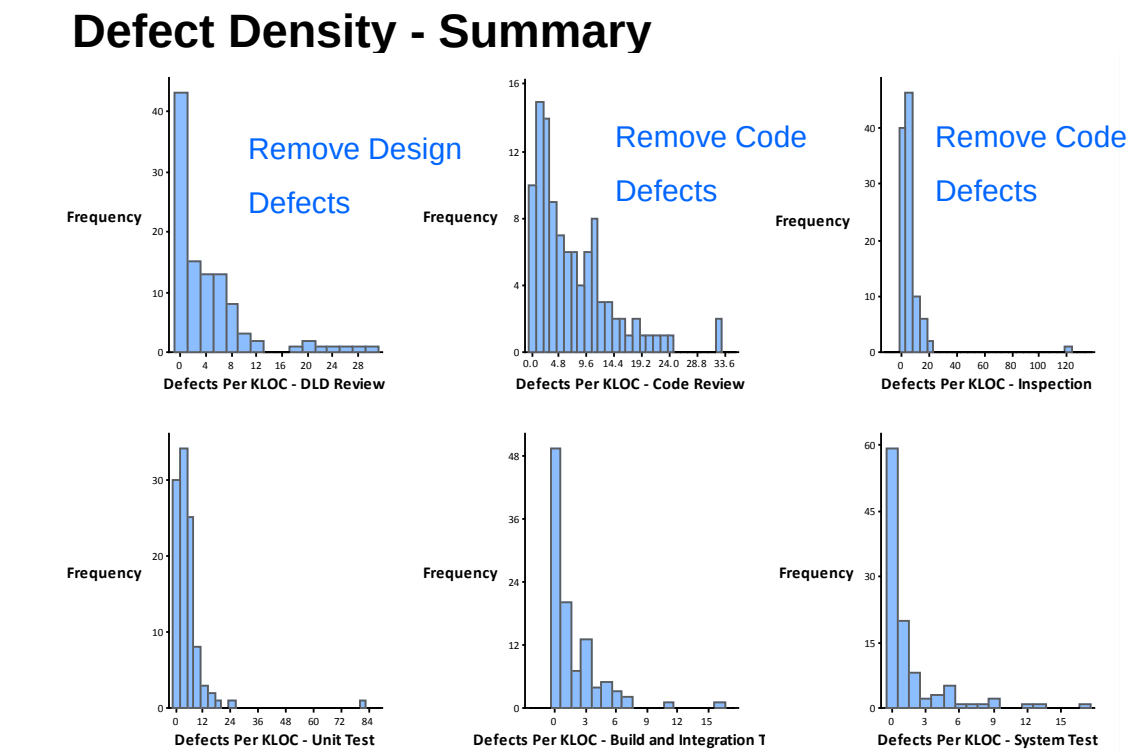
So what I have here, defects that were removed during design reviews.

Defect Density - Summary



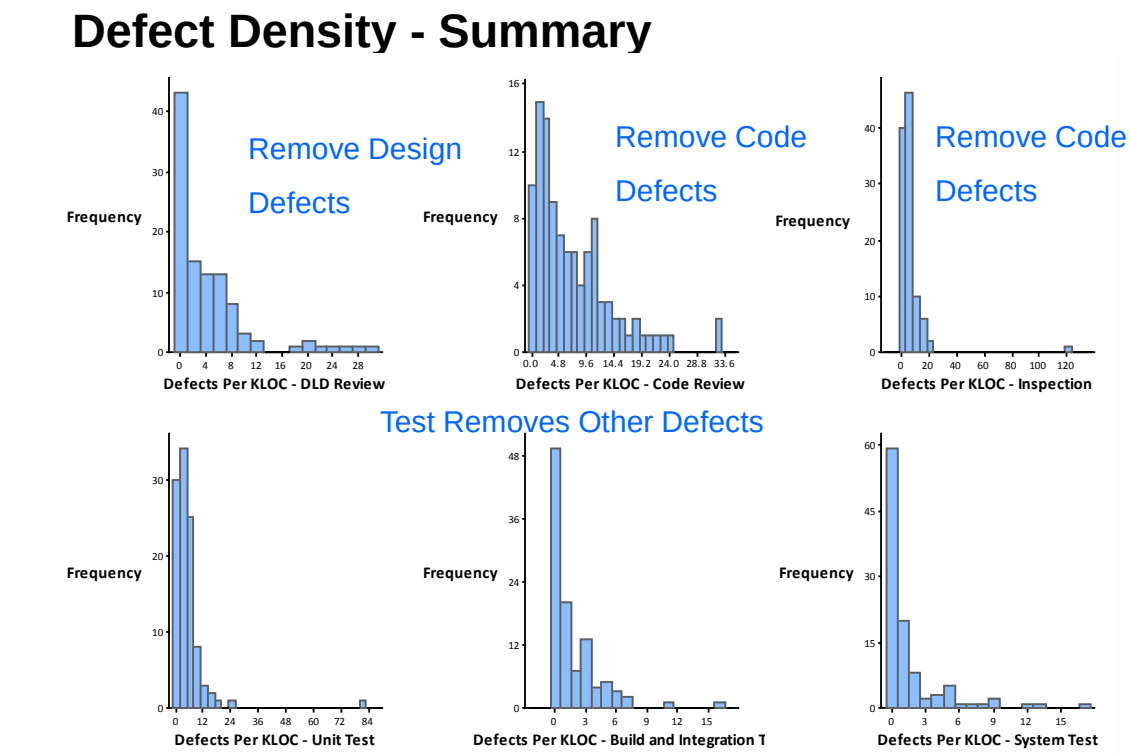
**097

Defect Density - Summary



**098 Here we have defects removed during personal code review, during peer review. And we can see the defect profile, the scales-- I'd prefer if they were all at the same. But we can see the profiles narrowing as we get deeper into the process.

Defect Density - Summary

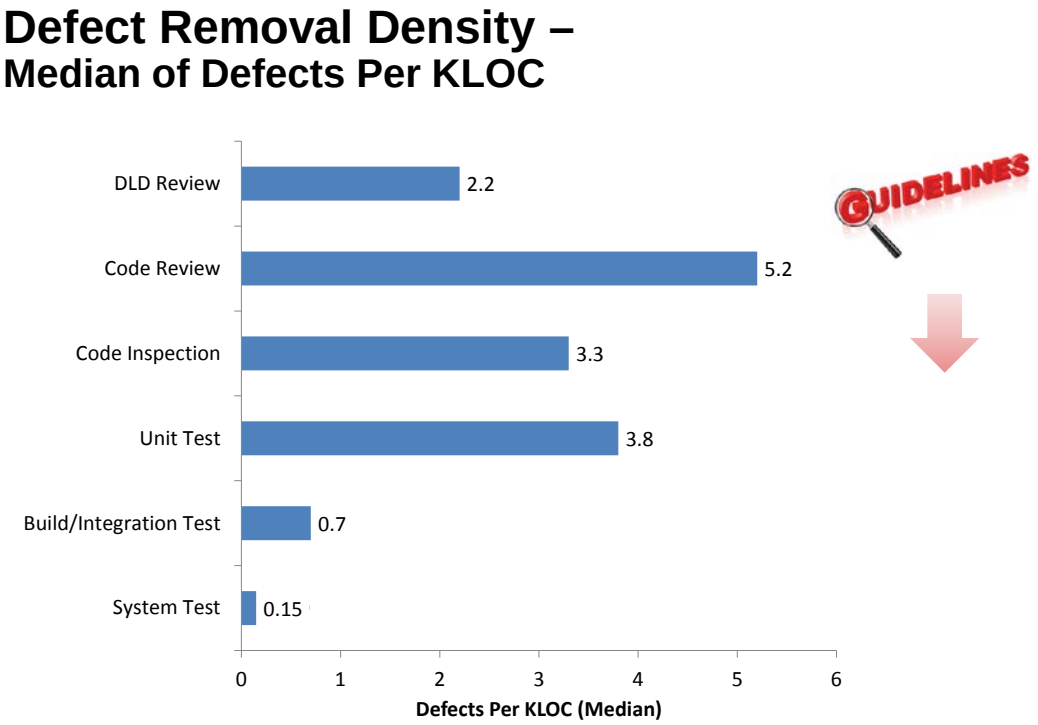


**099 Then we have various types of testing phases. And you can't quite see the scale here clearly; but here we have a peak of about 8.

As we get into later test build and integration, peak is getting closer-closer to one or two defects per thousand lines of code.

And in system test this peak here is actually less than 1, for these particular teams.

Defect Removal Density – Median of Defects Per KLOC



**100 Here are some of the numbers that we have measured as average benchmarks.

So in unit test we suggest that they try to get under five defects per thousand lines of code. In fact, these teams were measuring- as a group averaging 3.8.

We recommend less than half a defect per KLOC. In build and integration these teams on average were around .7.

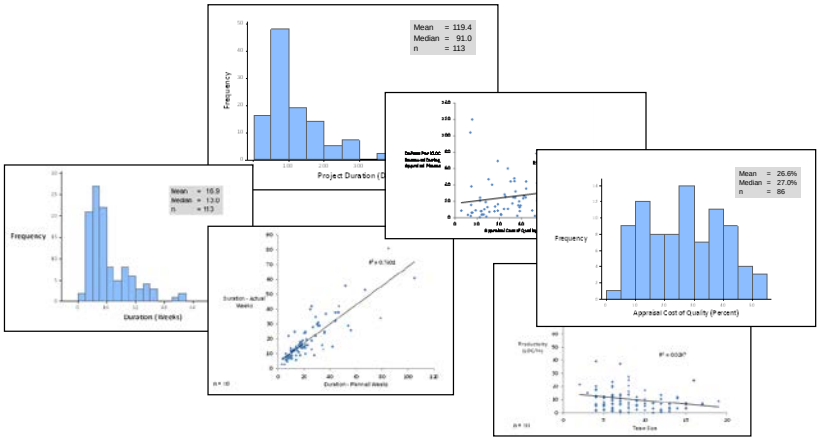
We recommend 2-- that would be 200 defects per million lines of code; to put this in perspective. Industry

standards right now, you typically see 1 to 5 per thousand; that would be 1000 to 5000 defects. We're recommending 200.

And what are the teams achieving? On average 150 defects per thousand lines of code in system test.

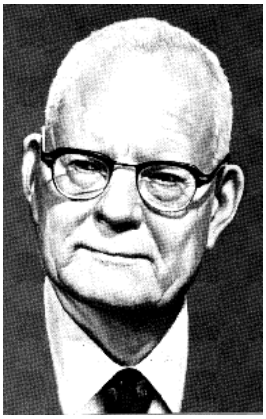
Now again some perspective. What we find is in the field, in industry, shipped/delivered products are typically in the 1 to 5 defects per thousand lines of code. If you have applications more like say your phone apps, probably closer to 10.

Download



**101 But that takes us to the download cycle. We have this benchmark data and much, much more. We can only scratch the surface in this time.

In God we trust…



W. Edwards Deming

In God we trust …

All others bring us

good data

**102 And I'll leave you with this message from Edwards Deming: In God we trust; all others bring data.

Shane McGraw: So a question for Bill, from Michael asking: Does a process methodology-- for example, Agile or Waterfall-- affect time on task?

Bill Nichols: I don't think we've seen any distinct measures on Agile versus Waterfall per se. It would be kind of hard because we haven't really seen- we haven't seen many of them instrumented. We have had some Agile projects instrumented though.

Frankly-- let's see, I guess we could compare-- I can't go into the company names; but I do know some specific ones. I haven't seen anything that jumps out at me. But it's a good question. I'd have to look at that.

Shane McGraw: Okay. One for Mark. I believe this is from Earle- or from Hella asking: For the projects in the sample, were they purely software? In other words, is firmware included? What is the end product; is it purely software?

Mark Kasunic: Purely software, from our projects, yes.

Shane McGraw: Okay. A question from Bradley asking: Are the team counts just for developers or do they include others such as BAs and testers? I believe this was from Earle here: Are the team counts for just developers or do they include others such as BAs and testers?

Bill Nichols: In this particular set it's including all team measures- all team members; depending upon the nature of the project. If it was only the development phase, typically it was developers. But several of the projects do have testing specialists and requirements specialists. So it's a mix right now. We haven't segmented the data down to that level.

Shane McGraw: Okay great. Darren asks: With the higher lines of code with smaller teams was there a correlation with defects?

Bill Nichols: We have some data on that. We haven't looked at the production rates versus defects. But we did not find any correlation with defects and size of team.

Shane McGraw: Okay. Nat asked-- this was from a section with Bill: If the time spent on task is so low, what else is taking up the time? He mentions is it YouTube?

Bill Nichols: No actually I've personally taken my own data; and literally I used to work in a vault-- it was an old- literally an old vault-- in the back of it. So people weren't coming by and interrupting me. And the peak times I was able to hit with absolutely no interruptions-- just coming in in the morning, sitting down at my computer, turning it on and starting to work-- were 28, 29 hours. I might've cracked 30 once. So with absolutely no interruptions, that is like an absolute upper limit.

Mark Kasunic: Yes I really encourage people, just as an experiment try tracking your time for one week. It's really going to blow your mind. You're going to actually see, you know, firsthand that-- you know, where did that time go? You know?

It's a real eye opener; it was for me. I still track my time; and I'm always kind of surprised. It makes you much more aware of how you're using your time. You know?

Bill Nichols: Well and it's more practical. If you have like a staff meeting, it's not just that hour; you have to get ready for the meeting.

Mark Kasunic: Right, right.

Bill Nichols: You have to walk back to your desk and start-- by the time I've started my clocking, it might be a half hour past the meeting time.

Mark Kasunic: Yes.

Shane McGraw: Okay.

Mark Kasunic: Amazing, yes.

Shane McGraw: Next one, from Mike, asking: At the team level or at the division level, what are some best practices in terms of software metrics collected and reported that answers the question: How do you know as a software organization we are getting more efficient?

Bill Nichols: Oh that's a tough question; because you're talking about efficiency. But the single best efficiency measure that I would use would probably be cost to quality. How much rework are we putting into the product?

Shane McGraw: We've got about 30 seconds left. We'll just fire off one more question here. And this is from Hella asking: What is the difference between code review and code inspection? We'll wrap up with that.

Bill Nichols: You want to take that?

Mark Kasunic: Oh sure. Well code reviews are considered personal reviews; that is, before I perform unit test I'm going to sit down, look at my code and go through it, do a personal review of the logic of whatever defects might be in there.

A peer review or inspection is when you have multiple colleagues that's looking at and inspecting your code, looking for defects from their various perspectives.

So that's really the difference, yes.

Shane McGraw: Excellent. Mark, Bill, thank you very much.
Mark Kasunic: Oh yes, thank you.

Shane McGraw: Great presentation.

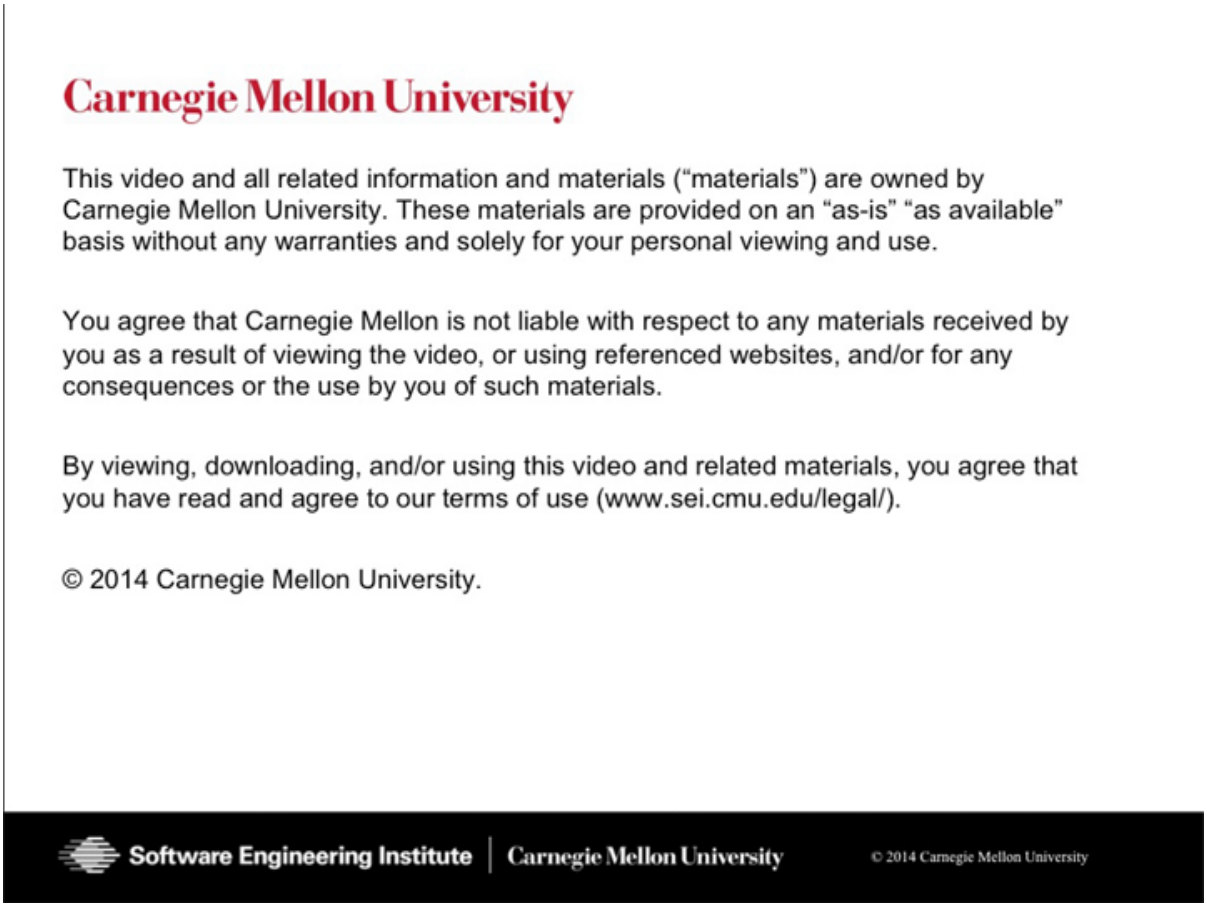
Everyone, that's our time for today. Again, I know we have lots of questions we weren't able to get. Please go to LinkedIn; search under groups for measurement and analysis. Post your questions there. We'll get to as many as we can.

And lastly I just want to invite you all to submit a proposal to the Team Software Process Symposium, which is set to take place in November in Pittsburgh, Pennsylvania.

The tactical program this year will go beyond the core methodology of TSP to encompass a broader range of complementary practices that contribute to peak performance in

software and systems projects; and
the unifying theme this year is
Software Quality.

Copyright



**107