Measuring the Extent of Testing

Cem Kaner, J.D., Ph.D.
Professor: Computer Science
Florida Institute of Technology

About These Slides

- There are more slides here than I will use in the presentation, and more points per slide than I will speak.
- This is not an accident and it is not poor time planning. It is my style. I use slides ***twice***:
  - Once to highlight the lecture
  - Once to help you work through the paper that comes with this talk.
- Please refer to these slides as you read the paper. These will provide highlights, summaries, and illustrations while reminding you of the lecture that you heard at this conference.

Question

- Imagine being on the job. Your local PBH (pointy-haired boss) drops in and asks

  "So, how much testing have you gotten done?"

- Please write down an answer. Feel free to use fictitious numbers but (except for the numbers) try to be realistic in terms of the type of information that you would provide.

The Question is Remarkably Ambiguous

Common answers are based on the:

Product ➢ We've tested 80% of the lines of code.

Plan ➢ We've run 80% of the test cases.

Results ➢ We've discovered 593 bugs.

Effort ➢ We've worked 80 hours a week on this for 4 months. We've run 7,243 tests.

The Question is Remarkably Ambiguous

Common answers are based on the:

Obstacles	➢ We've been plugging away but we can't be efficient until X, Y, and Z are dealt with.
Risks	➢ We're getting a lot of complaints from beta testers and we have 400 bugs open. The product *can't be* ready to ship in three days.
Quality of Testing	➢ Beta testers have found 30 bugs that we missed. Our regression tests seem ineffective.
History across projects	➢ At this milestone on previous projects, we had fewer than 12.3712% of the bugs found still open. We should be at that percentage on this product too.

How do we measure extent of testing?

- Before we can measure something, we need some sense of what we're measuring. It's easy to come up with "measurements" but we have to understand the relationship between the thing we want to measure and the statistic that we calculate to "measure" it.
- **If we want to measure the "extent of testing", we have to start by understanding what we mean by "extent of testing."**

What is measurement?

- Is measurement really “the assignment of numbers to objects or events according to a clear cut rule”?
  - No, it can’t be. If it was, then many inappropriate rules would do.

Surrogate measures

- "Many of the attributes we wish to study do not have generally agreed methods of measurement. To overcome the lack of a measure for an attribute, some factor which can be measured is used instead. This alternate measure is presumed to be related to the actual attribute with which the study is concerned. These alternate measures are called surrogate measures."
  - » Mark Johnson's MA Thesis
- "Surrogates" provide unambiguous assignments of numbers according to rules, but they don't provide an underlying theory or model that relates the measure to the attribute allegedly being measured.

What is measurement?

- **Measurement is the assignment of numbers to objects or events according to a rule <u>derived from a model or theory</u>.**

A framework for measurement

A measurement involves at least 10 factors:

- Attribute to be measured
  - appropriate scale for the attribute
  - variation of the attribute
- Instrument that measures the attribute
  - scale of the instrument
  - variation of measurements made with this instrument
- Relationship between the attribute and the instrument
- Likely side effects of using this instrument to measure this attribute
- Purpose
- Scope

Framework for measurement

Attribute ➢ Extent of testing – *What does that mean?*

Instrument ➢ What should we count? *Lines? Bugs? Test cases? Hours? Temper tantrums?*

Mechanism ➢ How will increasing "extent of testing" affect the reading (the measure) on the instrument?

Side Effect ➢ If we do something that makes the measured result look better, will that mean that we've actually increased the extent of testing?

Purpose ➢ Why are we measuring this? What will we do with the number?

Scope ➢ Are we measuring the work of one tester? One team on one project? Is this a cross-project metrics effort? Cross-departmental research?

Attributes and Instruments

Length	Ruler
Duration	Stopwatch
Speed	Ruler / Stopwatch
Sound energy	Sound level meter
Loudness	Sound level comparisons by humans
Tester goodness	??? Bug count ???
Code complexity	??? Branches ???
Extent of testing???	???
----Product coverage	?? Count statements / branches tested ??
----Proportion of bugs found	?? Count bug reports or graph bug curves??

Simple measurement

- You have a room full of tables that appear to be the same length. You want to measure their lengths.
- You have a one-foot ruler.
- You use the ruler to measure the lengths of a few tables. You get:
  - 6.01 feet
  - 5.99 feet
  - 6.05 feet
- You conclude that the tables are “6 feet” long.

Simple measurement (2)

- Note the variation
  - Measurement errors using the ruler
  - Manufacturing variation in the tables
- Note the rule:
  - We are relying on a direct matching operation and on some basic axioms of mathematics
    - The sum of 6 one-foot ruler-lengths is 6.
    - A table that is 6 ruler-lengths long is twice as long as one that is 3 ruler-lengths long.
- These rules don't always apply. What do we do when we have something hard to measure?

Next measure: A race

Sandy, Joe and Susan run in a race. Sandy comes in first, Joe second, and Susan third.

- We assign Sandy the number 1 (first place) and give her $10,000.
- We assign Joe the number 2 and give him $1000.
- We assign Susan the number 3 and give her $100.

Questions:

- Is Sandy twice as fast as Joe and three times as fast as Susan?
- Is Sandy 10 times as fast as Joe and 100 times as fast as Susan?
- We assigned these numbers to Sandy, Joe and Susan according to a rule. Isn't that assignment based on their speed?

Did we measure their speed in this race or not?

The race (slide 2)

- The rankings (1st, 2nd, 3rd) could be called measures of speed but they are on an **ordinal scale**. What you can tell from them is that Sandy (1st place) was faster in this race than Joe, who was faster than Susan. Sandy might have been lots faster or a little faster—we can't tell.
- There are five main types of scales (categorical, ordinal, interval, ratio, absolute). A tremendous amount of mathematical arm-waving can be (and has been) done about the measurement problems and properties of scales.
- The scaling problem is moderately interesting, but it is hardly the most difficult problem in a theory of measurement.
- ***The much more challenging problem lies in the relationship between the measure and the underlying attribute.***

Measurement scales

- ***Absolute scale:*** You have four children, I have two.
- ***Ratio scale:*** (You have $200, I have $100. The relationship is the same if we switch to Canadian dollars, $300CDN vs $150CDN. Multiplying doesn't affect the relationship.)
- ***Interval scale:*** (Temperatures of 70, 75 and 80 degrees (Fahrenheit) differ from each other by 5 degrees. The difference (the interval) between 10 and 15, 70 and 75, and 75 and 80 is the same. But a 70 degree day is not 7 times as hot as a 10 degree day.
- ***Ordinal:*** (Position in the race. Bug severity: fatal, serious, moderate, minor)
- ***Categorical:*** (This is a design bug (category 1); that is a control flow bug (category 2).)

Consider bug counts

- Do bug counts measure testers?
- Do bug counts measure thoroughness of testing?
- Do bug counts measure the effectiveness of an automation effort?
- Do bug counts measure how near we are to being ready to ship the product?

How would we know?

Bug counts and testers

To evaluate an instrument that is supposed to measure an attribute, we have to ask two key questions:

- What ***underlying mechanism***, or fundamental relationship, justifies the use of the reading we take from this instrument as a measure of the attribute? ***If the attribute increases by 20%, what will happen to the reading?***
- What can we know from the instrument reading? How tightly is the reading traceable to the underlying attribute? ***If the reading increases by 20%, does this mean that the attribute has increased 20%.*** If the linkage is not tight, we risk serious ***side effects*** as people push the reading (the "measurement") up and down without improving the underlying attribute.

Bug counts and testers: mechanism?

Suppose we could improve testing by 20%.

This might mean that:

- We find more subtle bugs that are important but that require more thorough investigation and analysis
- We create bug reports that are more thorough, better researched, more descriptive of the problem and therefore more likely to yield fixes.
- We do superb testing of a critical area that turns out to be relatively stable.

The bug counts might even go down, even though tester goodness has gone up.

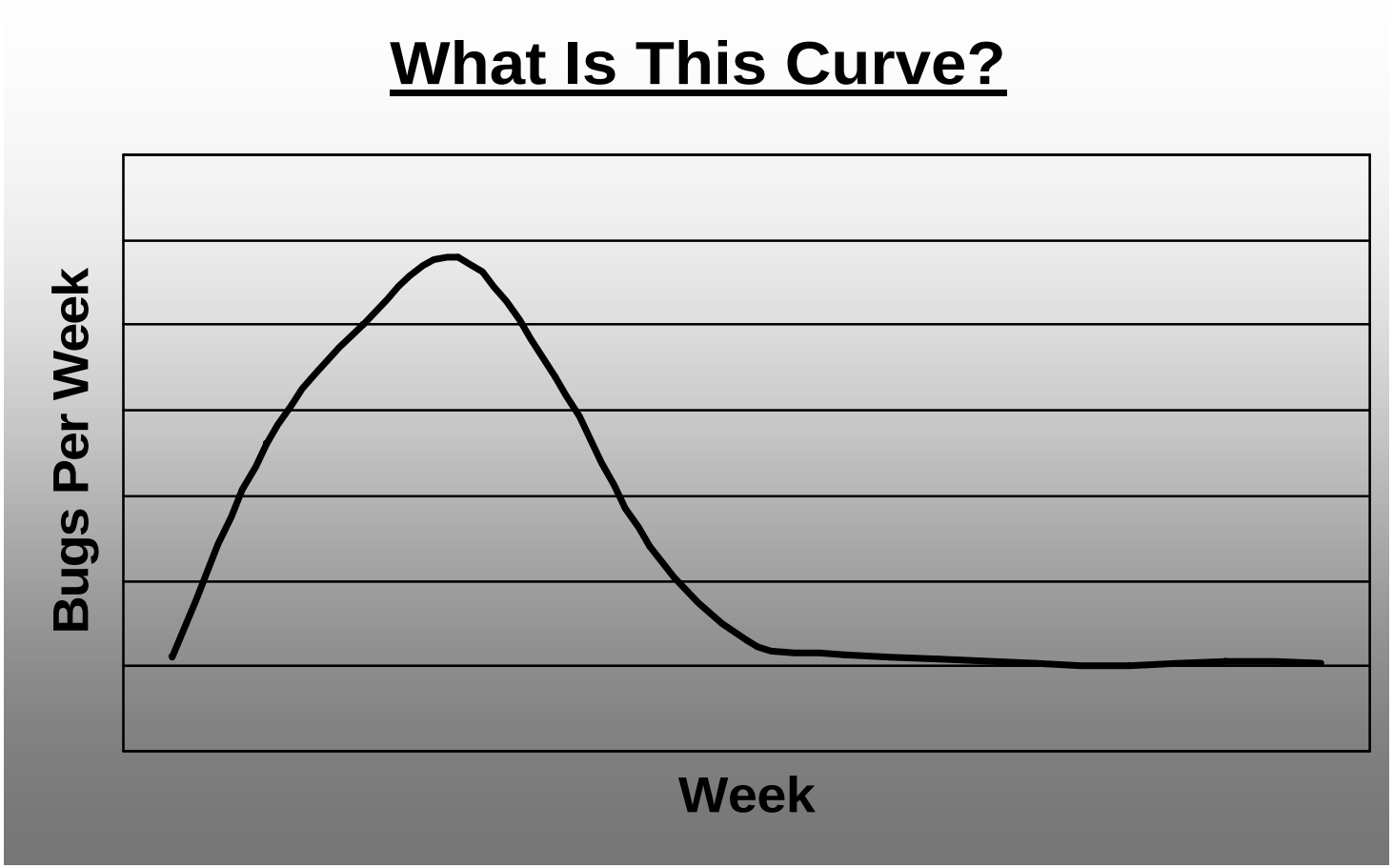
Bug counts & testers: Side effects

What if you could increase the count of reported bugs by 20%?

If you reward testers for higher bug counts, won't you make changes like these more likely?

- Testers report easier-to-find, more superficial bugs
- Testers report multiple instances of the same bug
- Programmers dismiss design bugs as non-bugs that testers put in the system to raise their bug counts
- No one will work on the bug tracking system or other group infrastructure.
- Testers become less willing to spend time coaching other testers.

Bug counts and extent of testing?



Bug counts and extent of testing

Attribute ➢	Not sure. Maybe we're thinking of percentage found of the total population of bugs in this product.
Instrument ➢	Bugs found. (Variations: bugs found this week, etc., various numbers based on bug count.)
Mechanism ➢	If we increase the extent of testing, does that result in more bug reports? *Not necessarily.*
Side Effect ➢	If we change testing to maximize the bug count, does that mean we've achieved more of the testing? ***Maybe in a trivial sense, but what if we're finding lots of simple bugs at the expense of testing for a smaller number of harder-to-find serious bugs?***

Bug curves and extent of testing

Attribute	➢	We have a model of the rate at which new bugs will be found over the life of the project.
Instrument	➢	Bugs per week. A key thing that we look at is the agreement between the predictive curve and the actual bug counts.
Mechanism	➢	As we increase the extent of testing, will our bug numbers conform to the curve? *Not necessarily. It depends on the bugs that are left in the product.*
Side Effect	➢	If we do something that makes the measured result look better, will that mean that we've actually increased the extent of testing? *No, no, no. See side effect discussion.*

Side effects of bug curves

Earlier in testing: (Pressure is to increase bug counts)

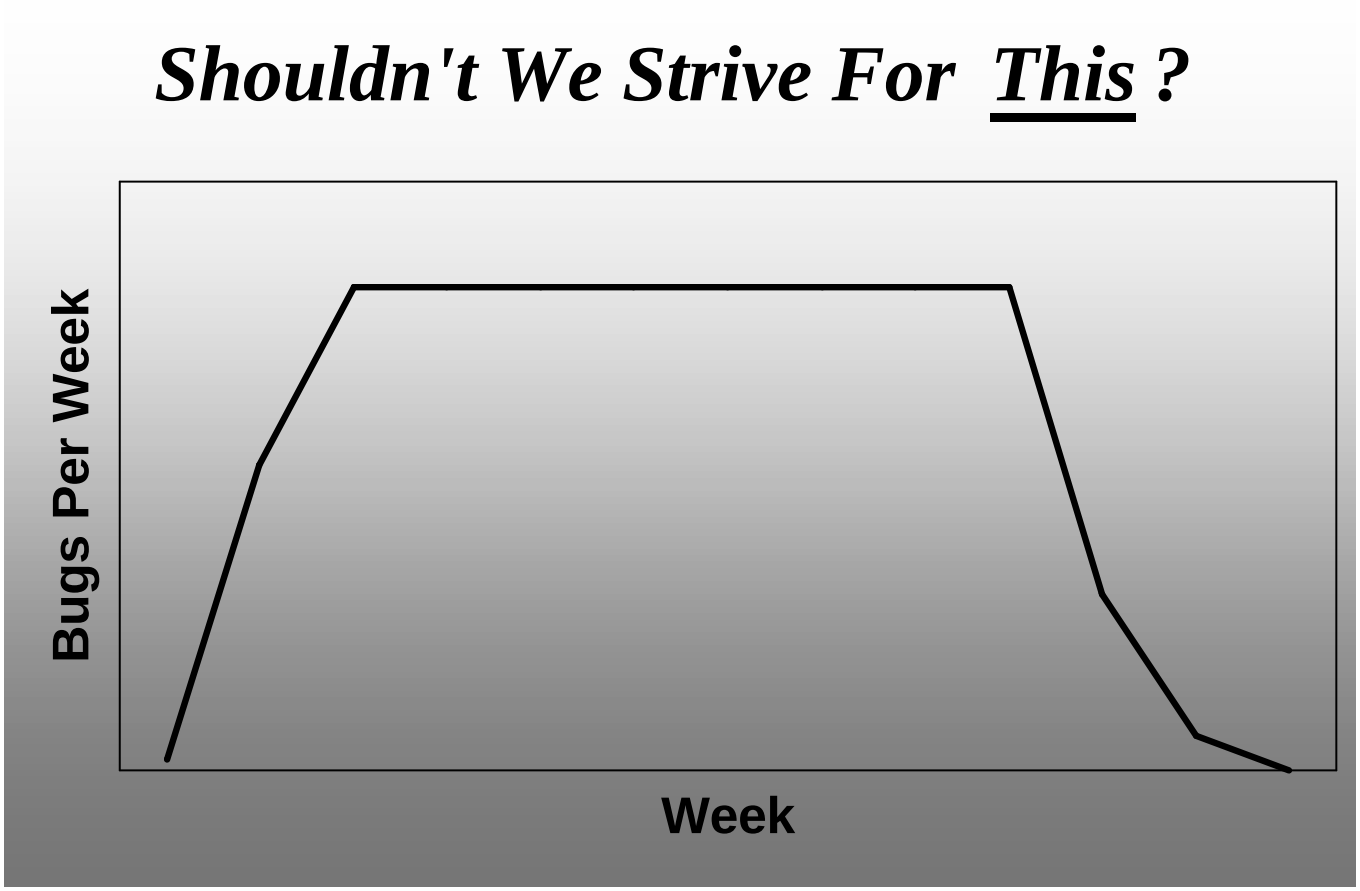
- Run tests of features known to be broken or incomplete.
- Run multiple related tests to find multiple related bugs.
- Look for easy bugs in high quantities rather than hard bugs.
- Less emphasis on infrastructure, automation architecture, tools and more emphasis of bug finding. (Short term payoff but long term inefficiency.)

Side effects of bug curves

Later in testing: ***Pressure is to decrease new bug rate***

- Run lots of already-run regression tests
- Don't look as hard for new bugs.
- Shift focus to appraisal, status reporting.
- Classify unrelated bugs as duplicates
- Class related bugs as duplicates (and closed), hiding key data about the symptoms / causes of the problem.
- Postpone bug reporting until after the measurement checkpoint (milestone). (Some bugs are lost.)
- Report bugs informally, keeping them out of the tracking system
- Testers get sent to the movies before measurement checkpoints.
- Programmers ignore bugs they find until testers report them.
- Bugs are taken personally.
- More bugs are rejected.

Bug curve counterproductive?



Code coverage

- Coverage measures the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:
  - ***100% line coverage*** means that you tested for every bug that can be revealed by simple execution of a line of code.
  - ***100% branch coverage*** means you will find every error that can be revealed by testing each branch.
  - ***100% coverage*** should mean that you tested for every possible error. This is obviously impossible.

The problem of coverage

- Several people seem to believe that complete statement and branch coverage means complete testing. (Or, at least, sufficient testing.)
- Part of the rationale comes from IEEE Std. 982.1-1988, § 4.17, “Minimal Unit Test Case Determination”
- IEEE Unit Testing Standard is

  100% Statement Coverage

  and100% Branch Execution
- Most companies don’t achieve this (though they might achieve 100% of the code they actually write.)

Benefits of coverage

Before I attack coverage measures, let me acknowledge that they are often useful.

- Many projects achieve low statement coverage, as little as 2% at one well-known publisher that had done (as measured by tester-hours) extensive testing and test automation. The results from checking statement coverage caused a complete rethinking of the company's automation strategy.
- Coverage tools can point to unreachable code or to code that is active but persistently untested.

Coverage tools can provide powerful diagnostic information about the testing strategy, even if they are terrible measures of the extent of testing.

Analysis: Statement/branch coverage

Attribute	➢	Extent of testing – ***How much of the product have we tested?***
Instrument	➢	Count statements and branches tested
Mechanism	➢	If we do more testing and find more bugs, does that mean that our line count will increase? ***Not necessarily. Example—configuration tests.***
Side Effect	➢	If we design our tests to make sure we hit more lines, does that mean we'll have done more extensive testing? ***Maybe in a trivial sense, but we can achieve this with weaker tests that find fewer bugs.***
Purpose	➢	Not specified
Scope	➢	Not specified

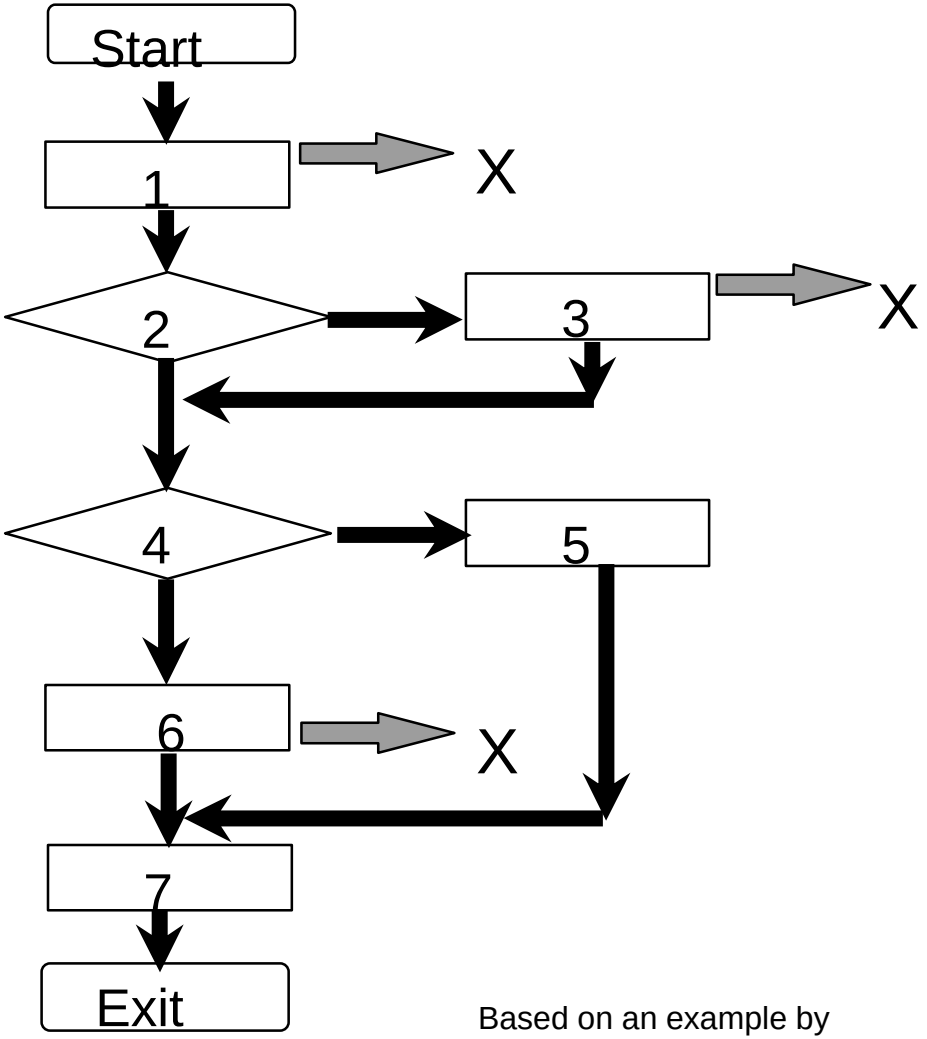
Statement / branch coverage
just test the flowchart

You're not testing:

- data flow
- tables that determine control flow in table-driven code
- side effects of interrupts, or interaction with background tasks
- special values, such as boundary cases. These might or might not be tested.
- unexpected values (e.g. divide by zero)
- user interface errors
- timing-related bugs
- compliance with contracts, regulations, or other requirements
- configuration/compatibility failures
- volume, load, hardware faults

Statement / branch coverage and data flows

→ X
means this routine changes variable X



1(x) 2 3(x) 4 5 7
1(x) 2 4 6(x) 7
Now we have 100% branch coverage, but where is 1(x) 7?
1(x) 2 4 5 7

Based on an example by Richard Bender

If we use “coverage”?

- If we improve testing by 20%, does this result in a 20% increase in “coverage”? Does it necessarily result in ANY increase in “coverage”?
- If we increase “coverage” by 20%, does this mean that there was a 20% improvement in the testing?
- If we achieve 100% “coverage”, do we really think we’ve found all the bugs?

Side effects and "coverage"

- Without a mechanism that ties changes in the attribute being measured to changes in the reading we get from the instrument, we have a "measure" that is ripe for abuse.

- People will optimize what is tracked. If you track "coverage", the coverage number will go up, but (as Marick has often pointed out) the quality of testing might go down.

Having framed the problem . . .

- Do I have a valid, useful measure of the extent of testing?
  - Nope, not yet.
  - The development and validation of a field's measures takes time.
- In the meantime, what do we do?
  - We still have to manage projects, monitor progress, make decisions based on what's going on - - - so ignoring the measurement question is not an option.
  - Let's look at strategies of some seasoned test managers and testers.

One approach: Balanced scorecard

"Coverage" measures are popular because they provide management with essential (if incorrect) feedback about the progress of testing.

Rather than reporting a single not-very-representative measure of testing progress, consider adopting a "balanced scorecard" approach. Report:

- a small number (maybe 10) of different measures,
- none of them perfect,
- all of them different from each other, and
- all of them reporting progress that is meaningful to you.

Together, these show a pattern that might more accurately reflect progress.

One approach: Balanced scorecard

- **For 101 examples of possible coverage measures, that might be suitable for balancing, see "Software Negligence and Testing Coverage" at www.kaner.com. These are merged in a list with over 100 additional indicators of extent of testing in the paper, "Measurement Issues & Software Testing", which is included in the proceedings.**
- **Robert Austin criticizes even this balanced scorecard approach. It will still lead to abuse, if the measures don't balance each other out.**

"Extent" as a Multidimensional Problem

- We developed the 8 aspects (or dimensions) of "extent of testing" by looking at the types of measures of extent of testing that we were reporting.
- The accompanying paper reports a few hundred "measures" that fit in the 8 categories
  - product coverage - plan / agreement
  - effort - results
  - obstacles - risks
  - quality of testing - project history
- **ALL of them have problems**

"Extent" as a Multidimensional Problem

- So, look at testing progress reports actually used in the field. Do we see focus on these individual measures?
  - Often, NOT.
  - Instead, we see reports that show a pattern of information of different types, to give the reader / listener a sense of the overall flow of the testing project.
  - Several examples in the paper, here are two of them:
    - Hendrickson's component map
    - Bach's project dashboard

Project Report / Component Map (Hendrickson)

Page 1 --- Issues that need management attention

Page 2 --- Component map

Page 3 --- Bug statistics

Component	Test Type	Tester	Total Tests Planned / Created	Tests Passed / Failed / Blocked	Time Budget	Time Spent	Projected for Next Build	Notes

We see in this report:

- Progress against plan
- Effort
- Obstacles / Risks
- Results

Bach's Dashboard

Testing Dashboard				Updated 11/1/00	Build 32
Area	**Effort**	**Coverage Planned**	**Coverage Achieved**	**Quality**	**Comments**
File/edit	High	High	Low	😐	1345, 1410
View	Low	Med	Med	☺	
Insert	**Blocked**	Med	Low	☹	**1621**

We see coverage of areas, progress against plan, current effort, key results and risks, and obstacles.

Suggested Lessons

- Bug count metrics cover only a narrow slice of testing progress. There are LOTS of alternatives.
- Simple charts can carry a lot of useful information and lead you to a lot of useful questions.
- Report multidimensional patterns, rather than single measures or a few measures along the same line.
- Think carefully about the potential side effects of your measures.
- Listen critically to reports (case studies) of success with simple metrics. If you can't see the data and don't know how the data were actually collected, you might well be looking at results that were sanitized by working staff (as a side effect of the imposition of the measurement process).

Postscript: Measurement and Science

- Hetzel, "The sorry state of software practice measurement and evaluation" in Fenton, Whitty & Iizuka's *Software Quality Assurance & Measurement*
  - Unable to obtain detailed descriptions of experiments underlying published conclusions
  - Unable to obtain reference data
  - Unable to get answers to questions like, "How was testing effort calculated and measured?" In response to claims like, "Inspections found 1 defect per hour of effort compared to testing that typically was .2 or .3 per hour."
- Much of the data is either proprietary or from toy programs.

Postscript: Measurement and Science

- In other fields, we obtain original data from other researchers in order to:
  - Reanalyze (possibly to reinterpret, possibly to check compatibility with an entirely different theory)
  - Replicate (run the same experiment yourself).
- With most data presented in CS, we cannot obtain original data or thorough description of the original details.
- **Other fields would call most CS data case studies, not scientifically reported experiments.**
- **If engineering involves the application of science, and our "science" lacks a broad, shared data base, then how can we have genuine, licensable, "Software Engineering"?**

Thanks to . . .

Much material is from participants of the *Software Test Managers Roundtable* (STMR) & the *Los Altos Workshop on Software Testing* (LAWST).

- STMR 1 (October 3, November 1, 1999) focused on the question, ***How to deal with too many projects and not enough staff?*** Participants: Jim Bampos, Sue Bartlett, Jennifer Brock, David Gelperin, Payson Hall, George Hamblen, Mark Harding, Elisabeth Hendrickson, Kathy Iberle, Herb Isenberg, Jim Kandler, Cem Kaner, Brian Lawrence, Fran McKain, Steve Tolman and Jim Williams.
- STMR 2 (April 30, May 1, 2000) focused on the topic, ***Measuring the extent of testing***. Participants: James Bach, Jim Bampos, Bernie Berger, Jennifer Brock, Dorothy Graham, George Hamblen, Kathy Iberle, Jim Kandler, Cem Kaner, Brian Lawrence, Fran McKain, and Steve Tolman.
- LAWST 8 (December 4-5, 1999) focused on ***Measurement***. Participants: Chris Agruss, James Bach, Jaya Carl, Rochelle Grober, Payson Hall, Elisabeth Hendrickson, Doug Hoffman, III, Bob Johnson, Mark Johnson, Cem Kaner, Brian Lawrence, Brian Marick, Hung Nguyen, Bret Pettichord, Melora Svoboda, and Scott Vernon.

A few sources

Austin, R.D. (1996) *Measuring and Managing Performance in Organizations*.

Fenton & Pfleeger (1997) *Software Metrics: A Rigorous & Practical Approach*.

Johnson, M.A. *(1996) Effective and Appropriate Use of Controlled Experimentation in Software Development Research*, Master's Thesis (Computer Science), Portland State University.

Kitchenham, Pfleeger, & Fenton (1995) “Towards a framework for software measurement and validation.” *IEEE Transactions on Software Engineering*, vol. 21, December, 929-944.

Schneidewind (1994) “Methodology for Validating Software Metrics,” *Encyclopedia of Software Engineering* (Marciniak, ed.) → IEEE 1061, *Standard for a Software Quality Metrics Methodology*.

Weinberg, G.M. (1993) *Quality Software Management*, *Volume 2*, *First-Order Measurement*.

Weinberg, G.M. & E.L. Schulman (1974) "Goals and performance in computer programming," *Human Factors*, 16(1), 70-77.

Zuse (1997) *A Framework of Software Measurement.*