Week:
3 of 12

# COMP 2120 / COMP 6120

# METRICS

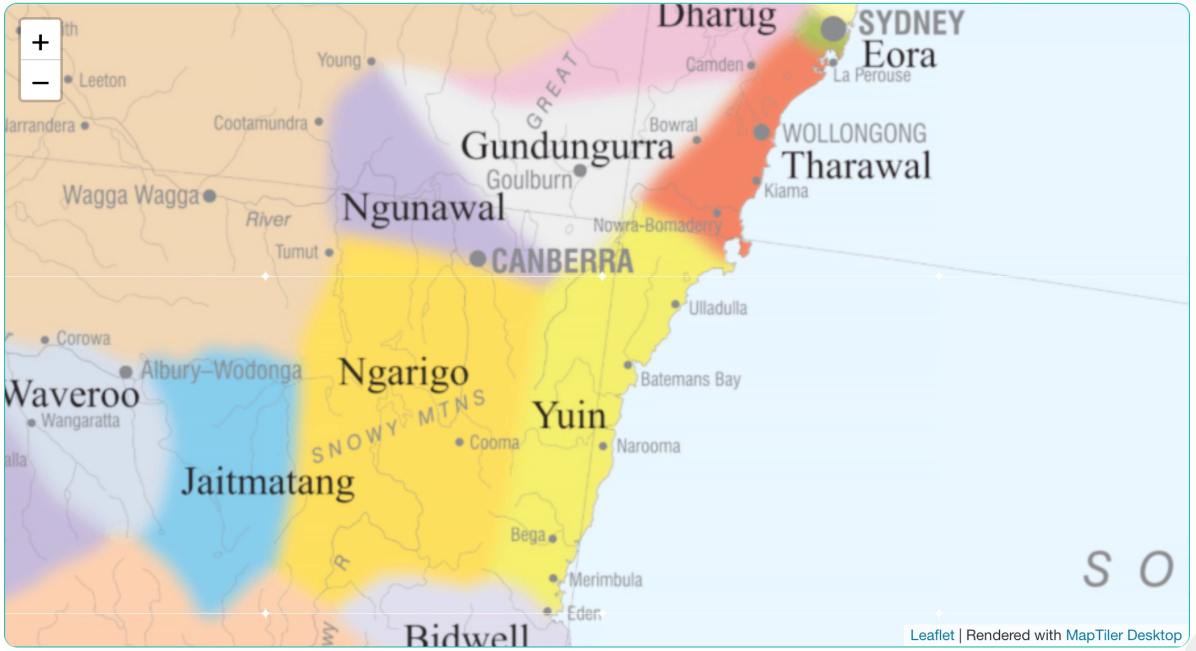
A/Prof Alex Potanin and Dr Melina Vidoni

CRICOS PROVIDER #00120C

# ANU Acknowledgment of Country

AIATSIS Explore Family history Collection Research Education What's new About Shop Search

"We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present."




https://aiatsis.gov.au/explore/map-indigenous-australia



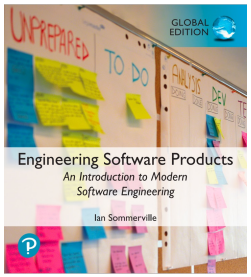
CRICOS PROVIDER #00120C

# SOFTWARE QUALITY



CRICOS PROVIDER #00120C

# Software Quality

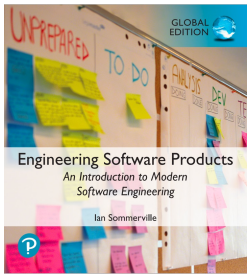



- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- The focus may be ‘fitness for purpose’ rather than specification conformance.



CRICOS PROVIDER #00120C

# Software fitness for purpose

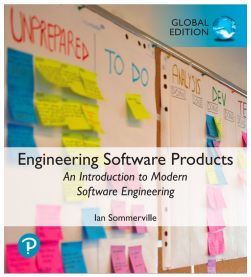



- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?
- Have programming and documentation standards been followed in the development process?



CRICOS PROVIDER #00120C

# Non-functional characteristics

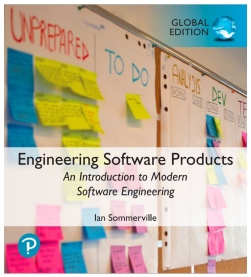



- The subjective quality of a software system is largely based on its non-functional characteristics.
- This reflects practical user experience – if the software's functionality is not what is expected, then users will often just work around this and find other ways to do what they want to do.
- However, if the software is unreliable or too slow, then it is practically impossible for them to achieve their goals.



CRICOS PROVIDER #00120C

# Software quality attributes

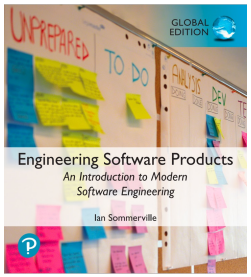



| | | |
|---|---|---|
| Safety | Understandability | Portability |
| Security | Testability | Usability |
| Reliability | Adaptability | Reusability |
| Resilience | Modularity | Efficiency |
| Robustness | Complexity | Learnability |



CRICOS PROVIDER #00120C

# Quality conflicts

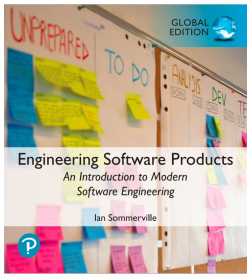



- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.



CRICOS PROVIDER #00120C

# Process and product quality

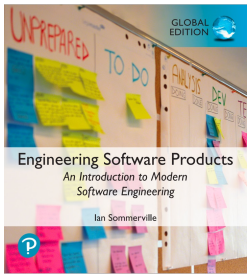


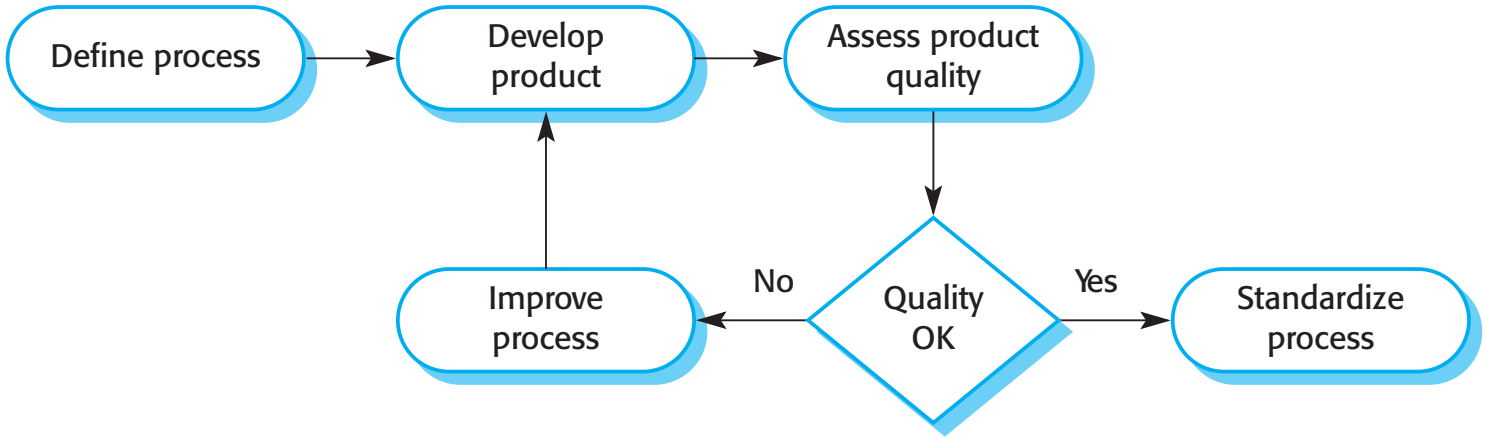

- The quality of a developed product is influenced by the quality of the *production process*.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.



CRICOS PROVIDER #00120C

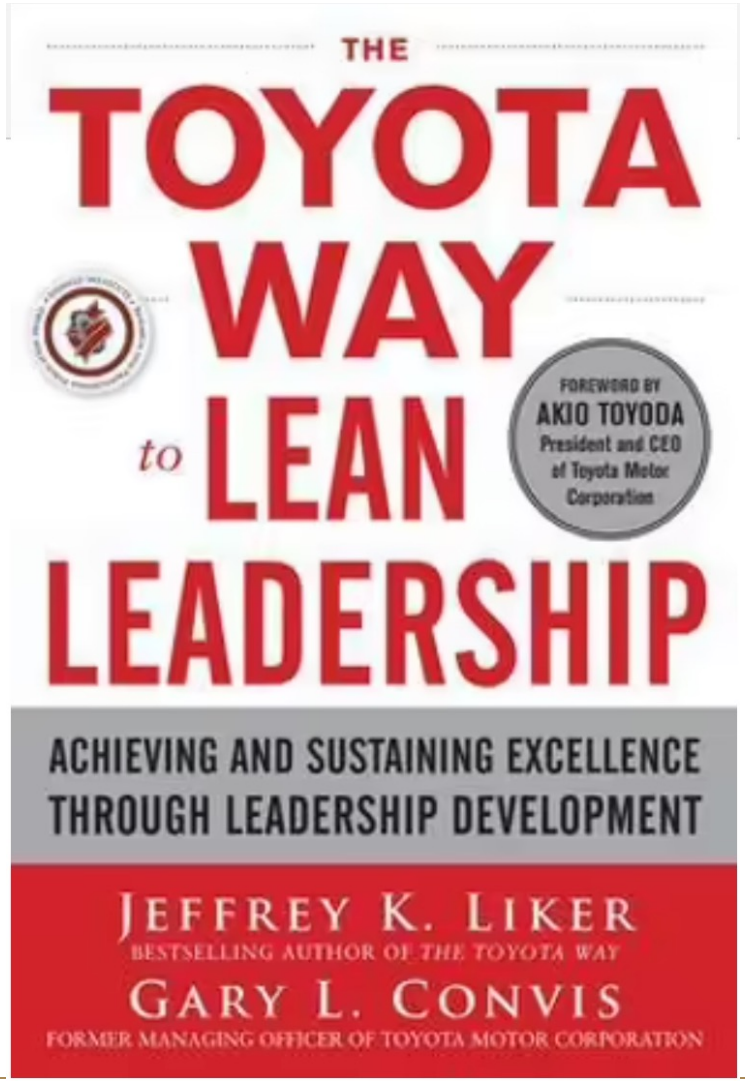
# Process-based quality






CRICOS PROVIDER #00120C






CRICOS PROVIDER #00120C

DRIVE

News

# Tesla boss Elon Musk admits car quality flaws, says mass production is “hell”

NEWS

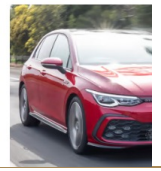
**Tesla CEO and founder Elon Musk has admitted the production quality, panel fit, and finish of his cars has at times been hit and miss, describing vehicle mass-production as “hell”.**

**Joshua Dowling** 06:02 04 February 2021 **108 comments** **0 shares**

In a rare interview – and the first with automotive industry veteran and vehicle “tear-down” expert Sandy Munro – Musk was surprisingly candid about where Tesla could make improvements and promised the company would do better as it matured and ramped up production to a more stable level.

On the YouTube channel hosted by Sandy Munro – a former manufacturing expert with big name car companies who now runs a business which pulls apart cars to find out how

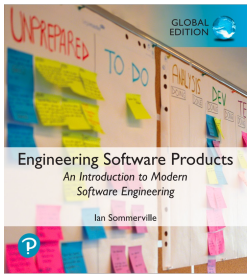
Trending N





CRICOS PROVIDER #00120C

# Quality Culture




- Quality managers should aim to develop a ‘quality culture’ where everyone responsible for software development is committed to achieving a high level of product quality.
- They should encourage teams to take responsibility for the quality of their work and to develop new approaches to quality improvement.
- They should support people who are interested in the intangible aspects of quality and encourage professional behavior in all team members.



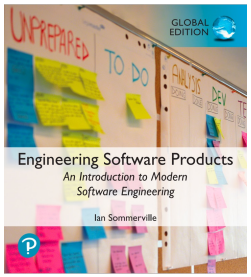
CRICOS PROVIDER #00120C

# SOFTWARE STANDARDS



CRICOS PROVIDER #00120C

# Software standards

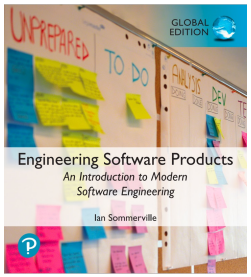



- Standards define the required attributes of a product or process. They play an important role in quality management.
- Standards may be international, national, organizational or project standards.



CRICOS PROVIDER #00120C

# Importance of standards

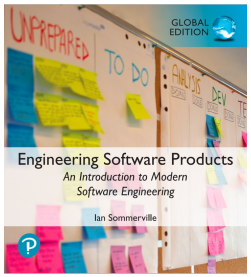



- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.



CRICOS PROVIDER #00120C

# Product and process standards

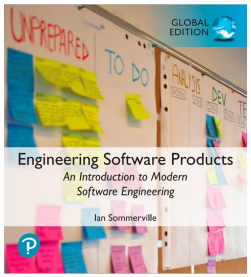



- *Product standards*
  - Apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.
- *Process standards*
  - These define the processes that should be followed during software development. Process standards may include definitions of specification, design and validation processes, process support tools and a description of the documents that should be written during these processes.



CRICOS PROVIDER #00120C

# Product and process standards

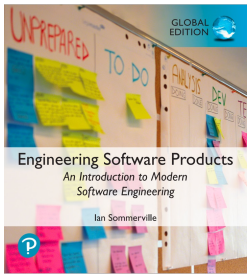



| Product standards | Process standards |
|---|---|
| Design review form | Design review conduct |
| Requirements document structure | Submission of new code for system building |
| Method header format | Version release process |
| Java programming style | Project plan approval process |
| Project plan format | Change control process |
| Change request form | Test recording process |



CRICOS PROVIDER #00120C

# Problems with standards

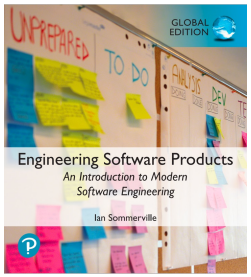



- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.



CRICOS PROVIDER #00120C

# Standards development

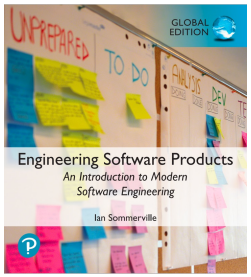



- Involve practitioners in development. Engineers should understand the rationale  underlying a standard.
- Review standards and their usage regularly.
  Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have specialized tool support. Excessive clerical work is the most significant complaint against standards.
  - Web-based forms are not good enough.



CRICOS PROVIDER #00120C

# ISO 9001 standards framework

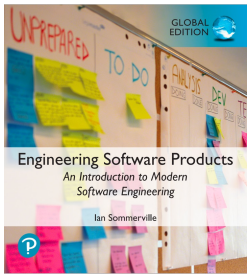


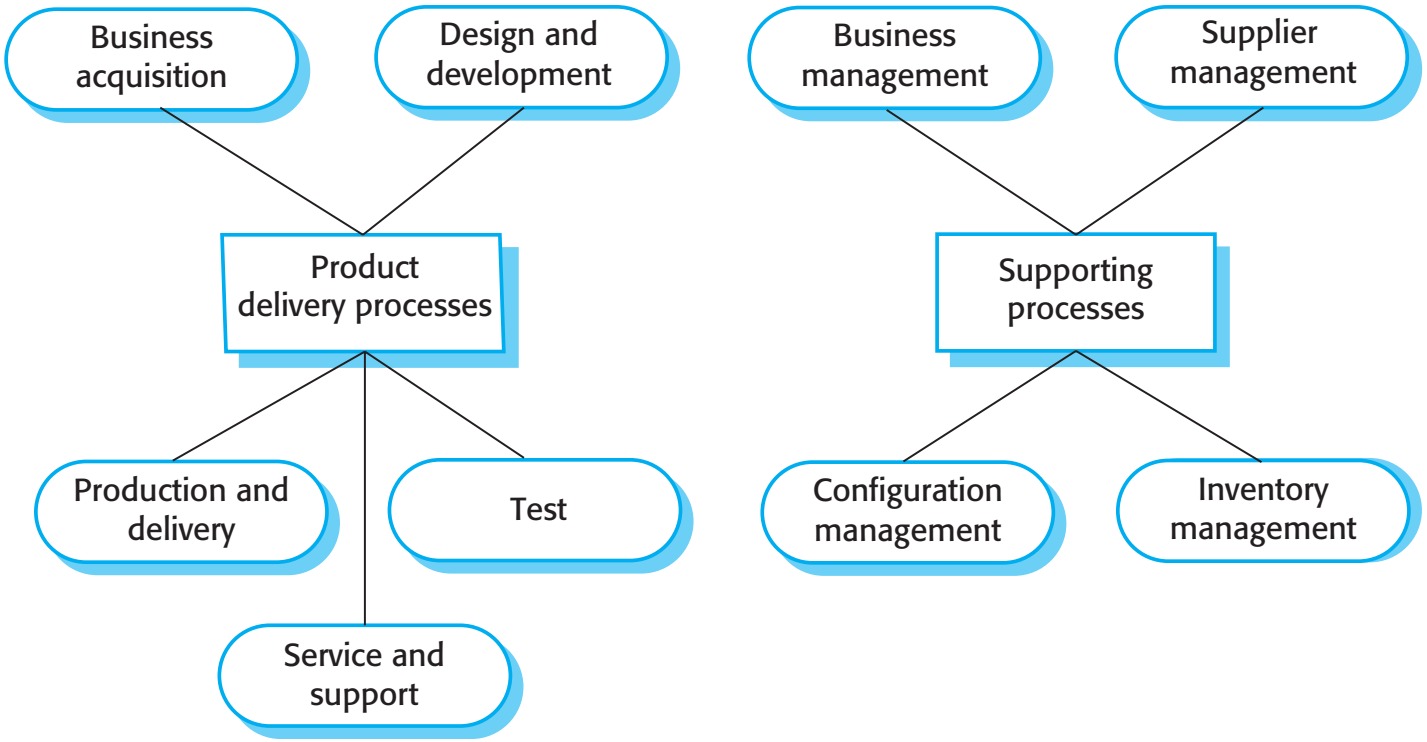

- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
  - It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.



CRICOS PROVIDER #00120C

# ISO 9001 core processes

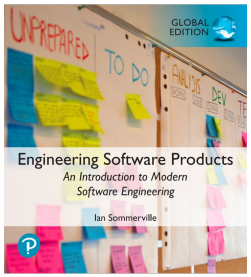


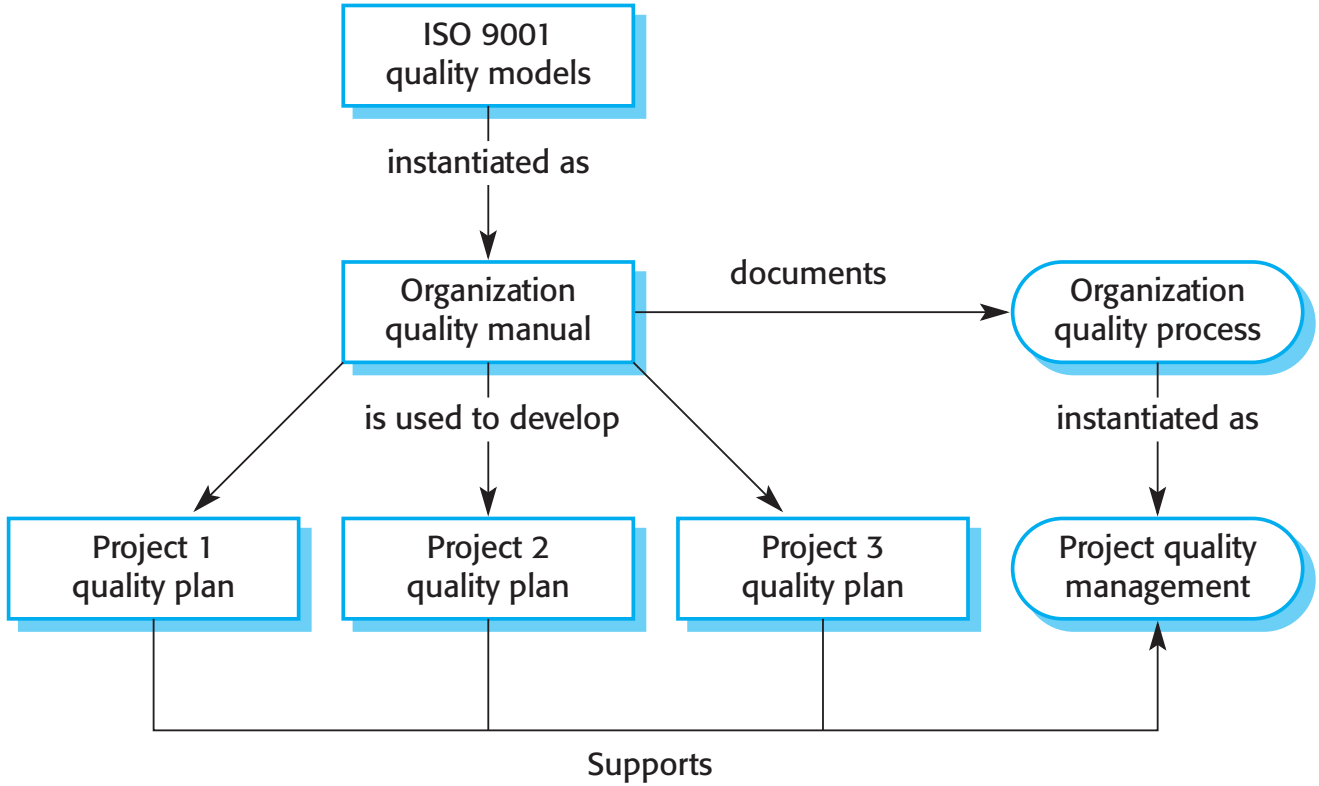






CRICOS PROVIDER #00120C

# ISO 9001 and quality management

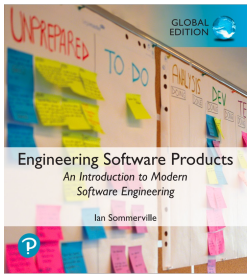





CRICOS PROVIDER #00120C

# ISO 9001 certification

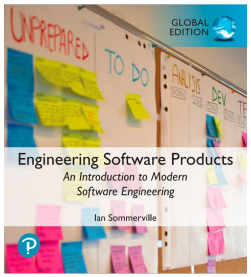



- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.



CRICOS PROVIDER #00120C

# Software quality and ISO9001

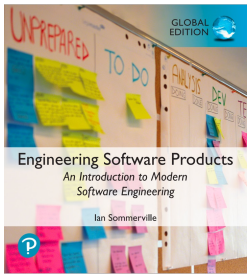



- The ISO 9001 certification is inadequate because it defines quality to be the conformance to standards.
- It takes no account of quality as experienced by users of the software. For example, a company could define test coverage standards specifying that all methods in objects must be called at least once.
- Unfortunately, this standard can be met by incomplete software testing that does not include tests with different method parameters. So long as the defined testing procedures are followed and test records maintained, the company could be ISO 9001 certified.



CRICOS PROVIDER #00120C

# Software quality management

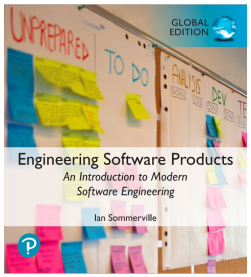



- Concerned with ensuring that the required level of quality is achieved in a software product.
- Three principal concerns:
  - At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.



CRICOS PROVIDER #00120C

# Quality management activities

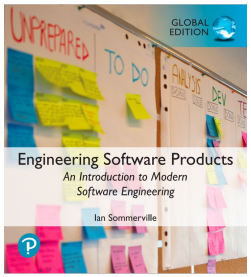


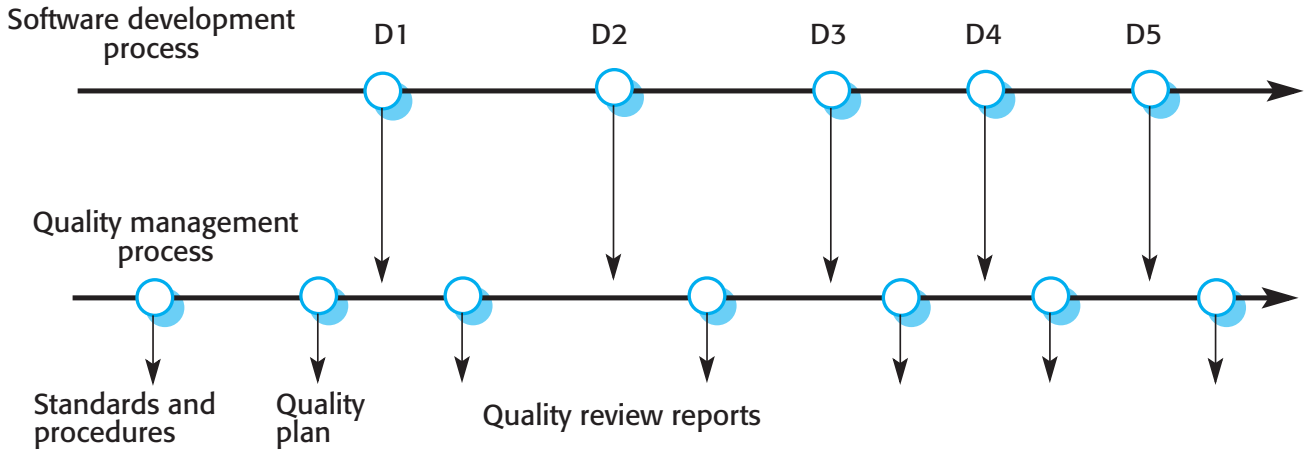

- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals
- The quality team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.



CRICOS PROVIDER #00120C

# Quality management and software development

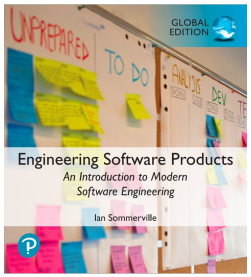





CRICOS PROVIDER #00120C

# Quality planning

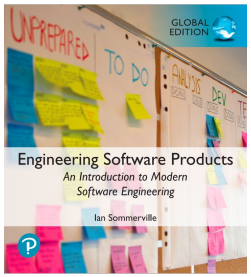



- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.



CRICOS PROVIDER #00120C

# Quality plans

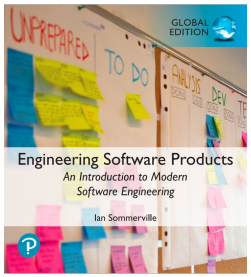



- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them.



CRICOS PROVIDER #00120C

# Scope of quality management




- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.
- Techniques have to evolve when agile development is used.



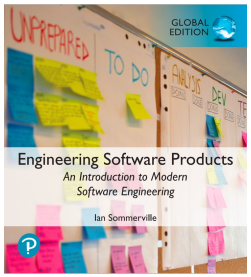
CRICOS PROVIDER #00120C

# SOFTWARE MEASUREMENT



CRICOS PROVIDER #00120C

# Software measurement

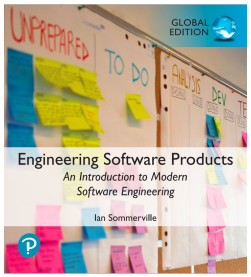



- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don’t make systematic use of software measurement.
- There are few established standards in this area.



CRICOS PROVIDER #00120C

# Software metric

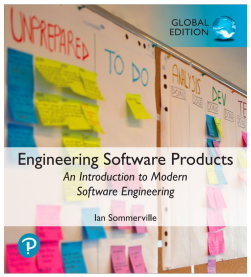



- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.



CRICOS PROVIDER #00120C

# Types of process metric

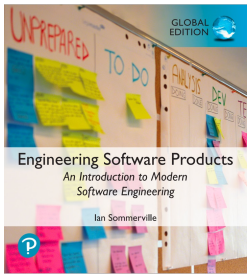


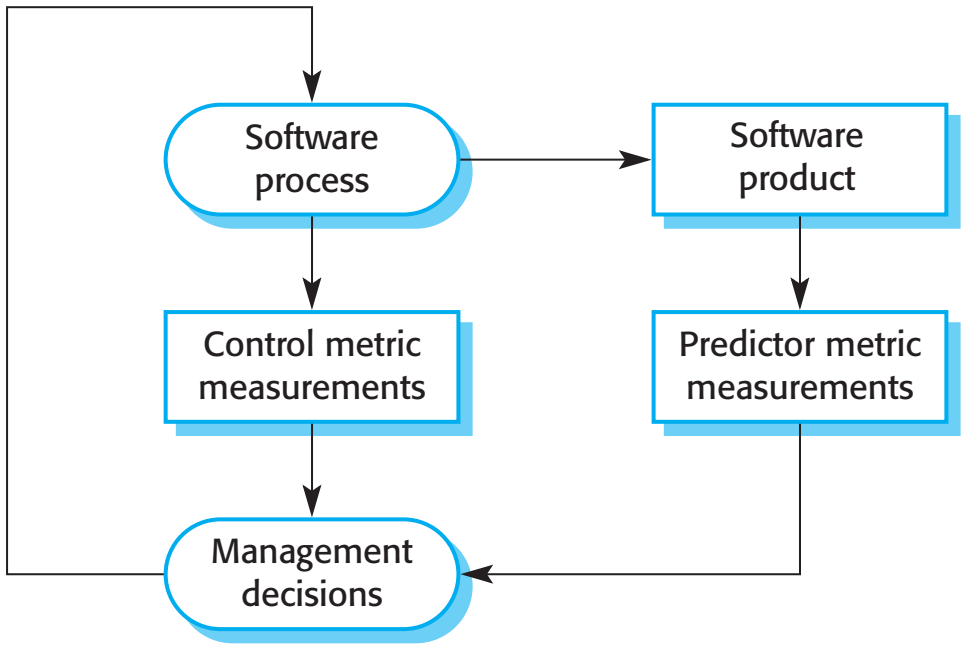

- *The time taken for a particular process to be completed*
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- *The resources required for a particular process*
  - Resources might include total effort in person-days, travel costs or computer resources.
- *The number of occurrences of a particular event*
  - Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, the number of bug reports in a delivered system and the average number of lines of code modified in response to a requirements change.



CRICOS PROVIDER #00120C

# Predictor and control measurements

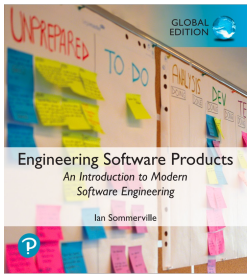





CRICOS PROVIDER #00120C

# Use of measurements

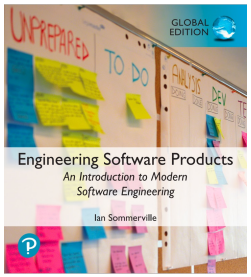



- To assign a value to system quality attributes
  - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
- To identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.



CRICOS PROVIDER #00120C

# Metrics assumptions

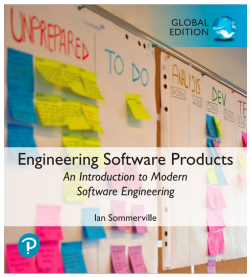


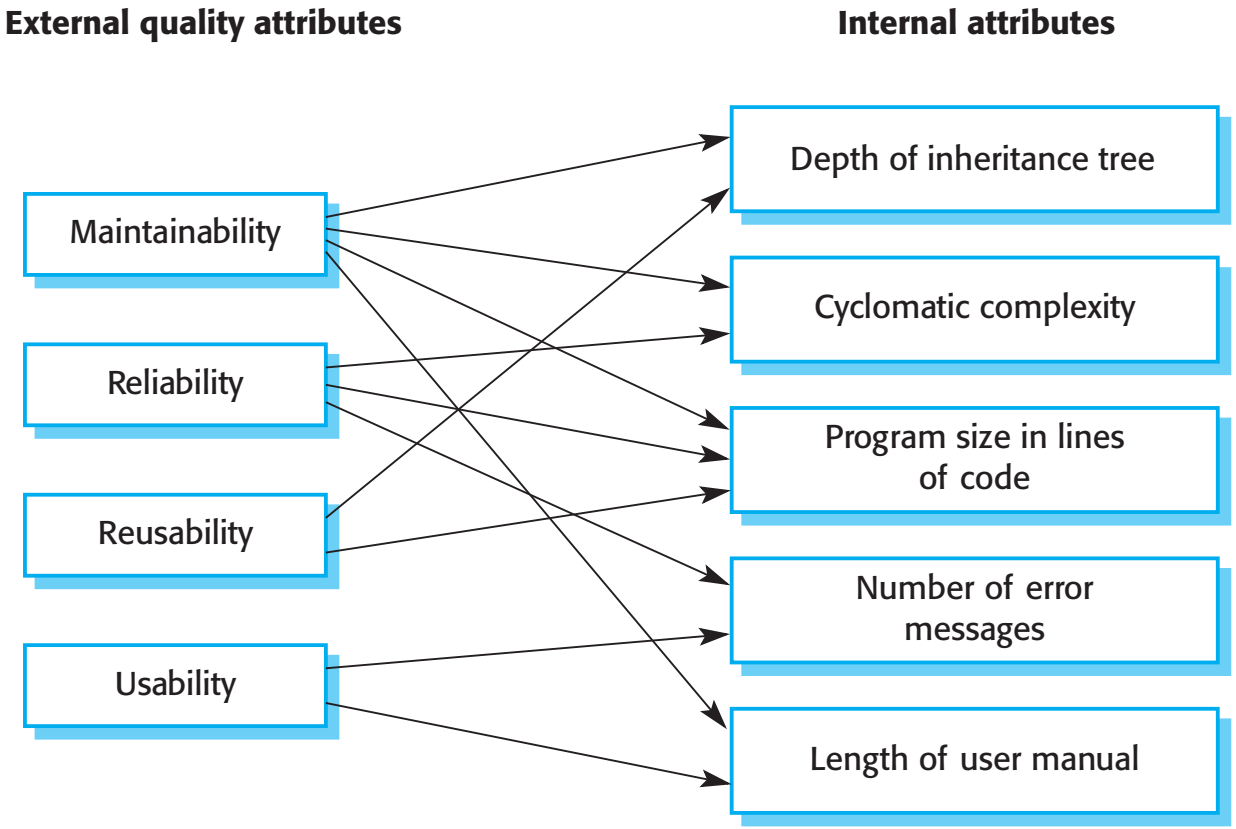

- A software property can be measured accurately.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.



CRICOS PROVIDER #00120C

# Relationships between internal and external software

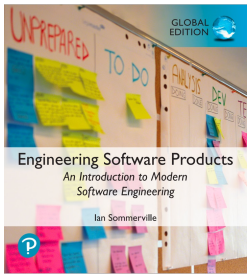





CRICOS PROVIDER #00120C

# Problems with measurement in industry

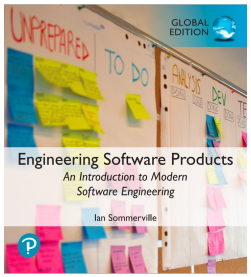



- It is impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.



CRICOS PROVIDER #00120C

# Empirical software engineering

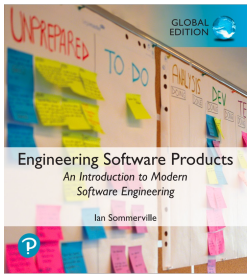



- Software measurement and metrics are the basis of empirical software engineering.
- This is a research area in which experiments on software systems and the collection of data about real projects has been used to form and validate hypotheses about software engineering methods and techniques.
- Research on empirical software engineering, this has not had a significant impact on software engineering practice.
- It is difficult to relate generic research to a project that is different from the research study.



CRICOS PROVIDER #00120C

# Product metrics

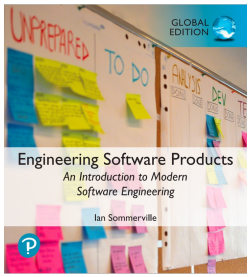



- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.



CRICOS PROVIDER #00120C

# Dynamic and static metrics

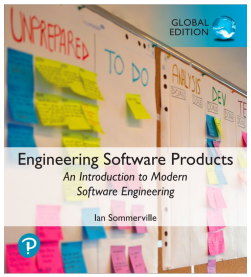



- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.



CRICOS PROVIDER #00120C

# Static software product metrics

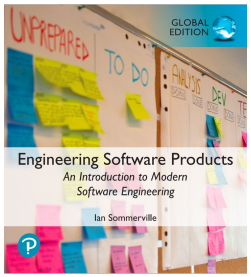



| Software metric | Description |
|---|---|
| Fan-in/Fan-out | Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components. |
| Length of code | This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components. |



CRICOS PROVIDER #00120C

# Static software product metrics

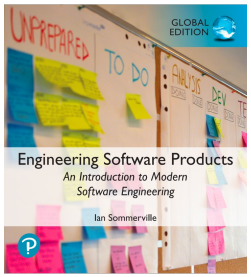



| Software metric | Description |
|---|---|
| Cyclomatic complexity | This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8. |
| Length of identifiers | This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program. |
| Depth of conditional nesting | This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone. |
| Fog index | This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand. |



CRICOS PROVIDER #00120C

# The CK object-oriented metrics suite

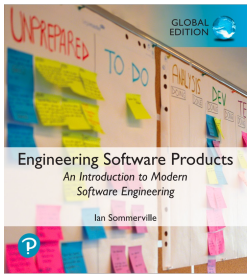



| Object-oriented metric | Description |
|---|---|
| Weighted methods per class (WMC) | This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree. |
| Depth of inheritance tree (DIT) | This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree. |
| Number of children (NOC) | This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them. |



CRICOS PROVIDER #00120C

# The CK object-oriented metrics suite

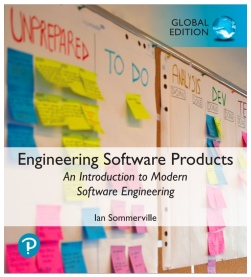



| Object-oriented metric | Description |
|---|---|
| Coupling between object classes (CBO) | Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program. |
| Response for a class (RFC) | RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors. |
| Lack of cohesion in methods (LCOM) | LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics. |



CRICOS PROVIDER #00120C

# Software component analysis

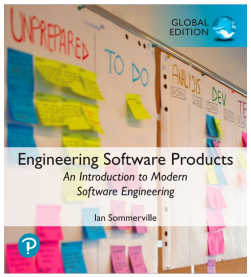


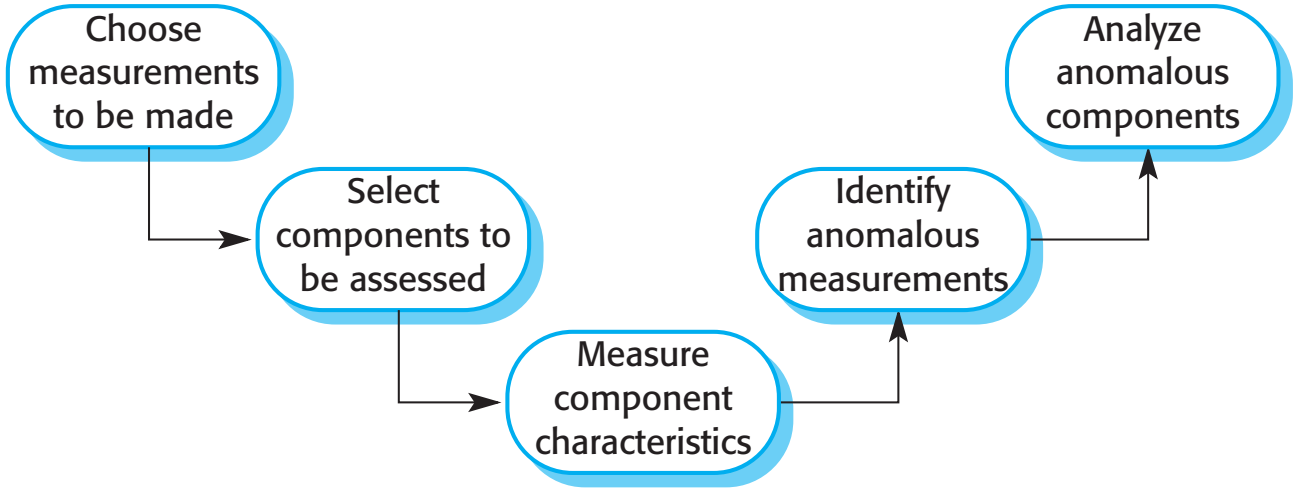

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.



CRICOS PROVIDER #00120C

# The process of product measurement

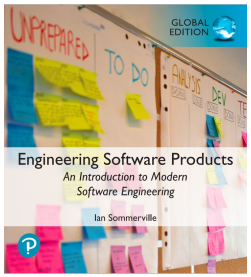





CRICOS PROVIDER #00120C

# Measurement ambiguity

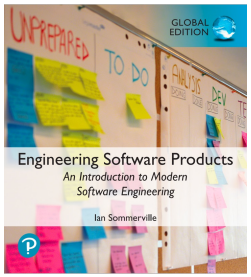



- When you collect quantitative data about software and software processes, you have to analyze that data to understand its meaning.
- It is easy to misinterpret data and to make inferences that are incorrect.
- You cannot simply look at the data on its own. You must also consider the context where the data is collected.



CRICOS PROVIDER #00120C

# Measurement surprises

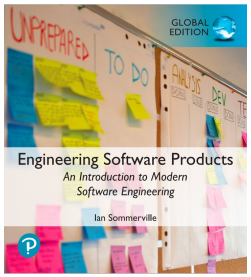



- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.



CRICOS PROVIDER #00120C

# Software context

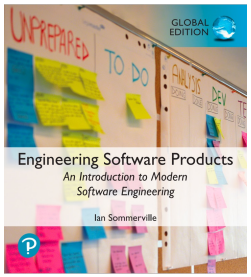



- Processes and products that are being measured are not insulated from their environment.
- The business environment is constantly changing and it is impossible to avoid changes to work practice just because they may make comparisons of data invalid.
- Data about human activities cannot always be taken at face value. The reasons why a measured value changes are often ambiguous. These reasons must be investigated in detail before drawing conclusions from any measurements that have been made.



CRICOS PROVIDER #00120C

# Software analytics

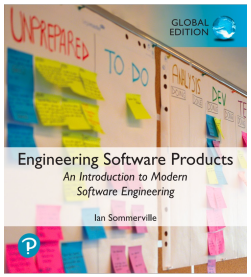



*Software analytics is analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions.*



CRICOS PROVIDER #00120C

# Software analytics enablers

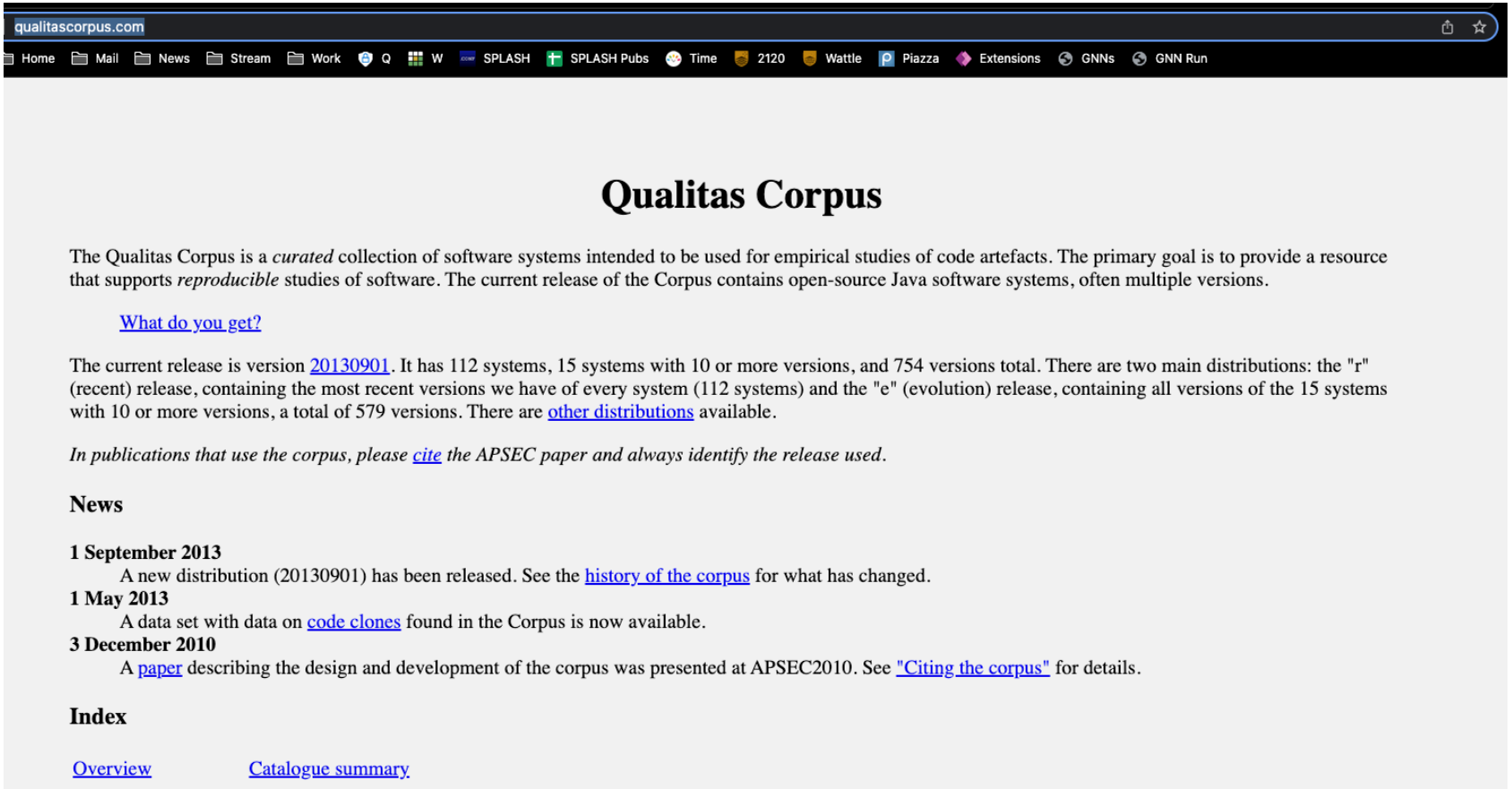



- The automated collection of user data by software product companies when their product is used.
  - If the software fails, information about the failure and the state of the system can be sent over the Internet from the user's computer to servers run by the product developer.
- The use of open source software available on platforms such as Sourceforge and GitHub and open source repositories of software engineering data.
  - The source code of open source software is available for automated analysis and this can sometimes be linked with data in the open source repository.



CRICOS PROVIDER #00120C

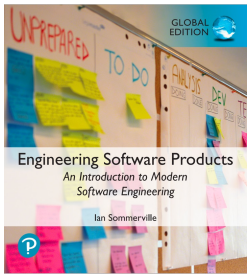
# Qualitas Corpus






CRICOS PROVIDER #00120C

# Analytics tool use

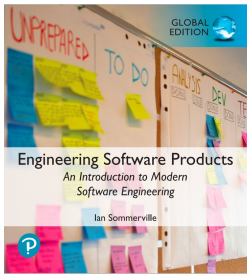



- Tools should be easy to use as managers are unlikely to have experience with analysis.
- Tools should run quickly and produce concise outputs rather than large volumes of information.
- Tools should make many measurements using as many parameters as possible. It is impossible to predict in advance what insights might emerge.
- Tools should be interactive and allow managers and developers to explore the analyses.



CRICOS PROVIDER #00120C

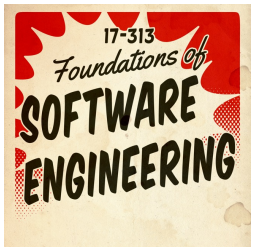
# Status of software analytics




- Software analytics is still immature and it is too early to say what effect it will have.
- Not only are there general problems of ‘big data’ processing, our knowledge depends on collected data from large companies.
  - This is primarily from software products and it is unclear if the tools and techniques that are appropriate for products can also be used with custom software.
- Small companies are unlikely to invest in the data collection systems that are required for automated analysis so may not be able to use software analytics.



CRICOS PROVIDER #00120C




# Software Engineering: Principles, practices (technical and non-technical) for confidently building high-quality software.

What does this mean?
How do we know?
→ *Measurement* and metrics are **key** concerns.



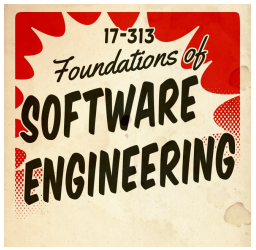
CRICOS PROVIDER #00120C

# CASE STUDY:
# THE MAINTAINABILITY INDEX

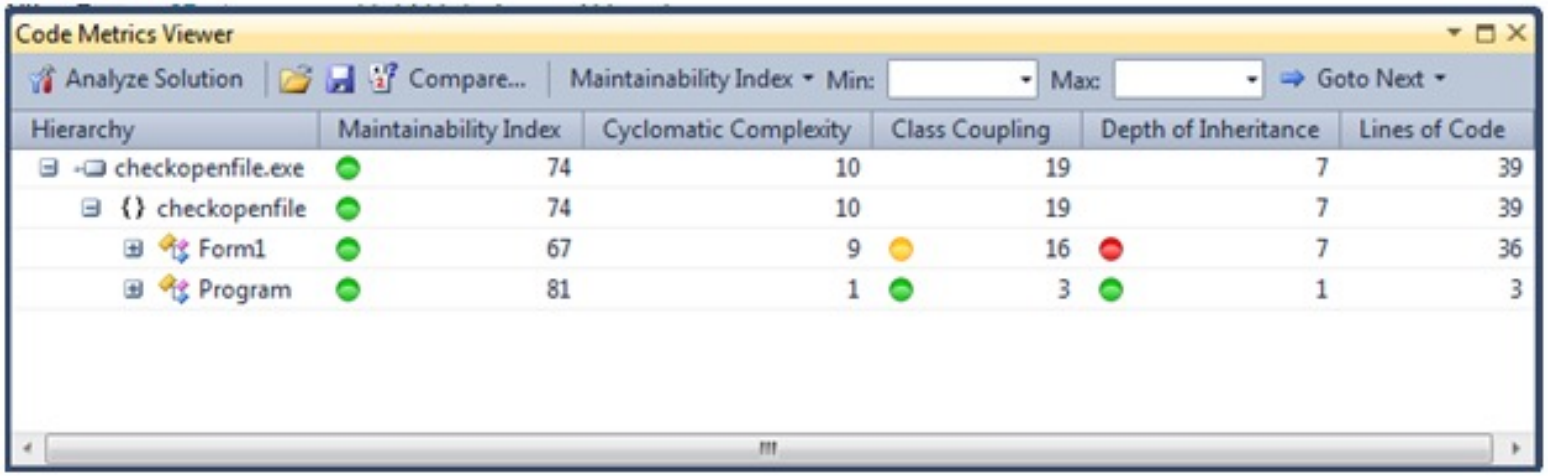


CRICOS PROVIDER #00120C

# Visual Studio (since 2007)




"Maintainability Index calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A **high value means better maintainability**. Color coded ratings can be used to quickly identify trouble spots in your code. A **green rating is between 20 and 100** and indicates that the code has good maintainability. A **yellow rating is between 10 and 19** and indicates that the code is moderately maintainable. A **red rating is a rating between 0 and 9** and indicates low maintainability."

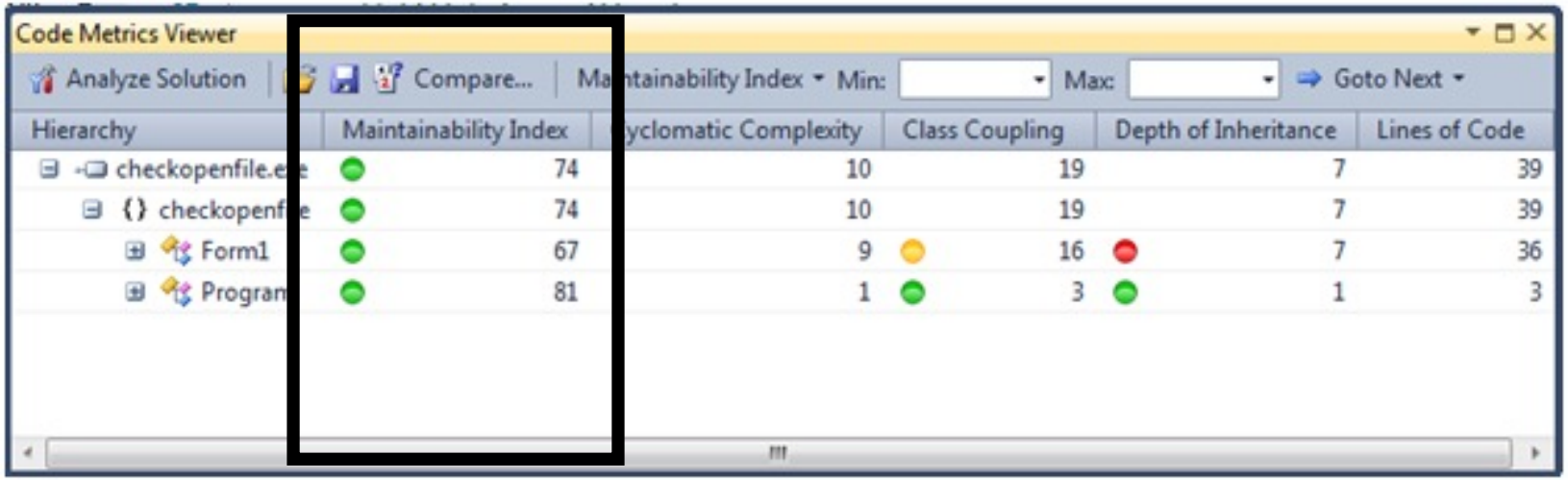



| Hierarchy | Maintainability Index | Cyclomatic Complexity | Class Coupling | Depth of Inheritance | Lines of Code |
|---|---|---|---|---|---|
| checkopenfile.exe | 74 | 10 | 19 | 7 | 39 |
| checkopenfile | 74 | 10 | 19 | 7 | 39 |
| Form1 | 67 | 9 | 16 | 7 | 36 |
| Program | 81 | 1 | 3 | 1 | 3 |



CRICOS PROVIDER #00120C

# Visual Studio (since 2007)


- Index between 0 and 100 representing the relative ease of maintaining the code.
- Higher is better. Color coded by number:
  - Green: between 20 and 100
  - Yellow: between 10 and 19
  - Red: between 0 and 9.



CRICOS PROVIDER #00120C

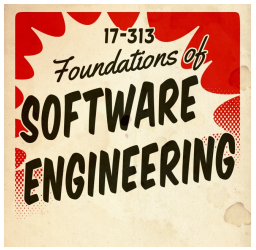
# Design rationale (from MSDN blog)


- "We noticed that as code tended toward 0 it was clearly hard to maintain code and the difference between code at 0 and some negative value was not useful."
- "The desire was that if the index showed red then we would be saying with a high degree of confidence that there was an issue with the code."
- http://blogs.msdn.com/b/codeanalysis/archive/2007/11/20/maintainability-index-range-and-meaning.aspx



CRICOS PROVIDER #00120C

# The Index




Maintainability Index =

MAX(0,(171 –

5.2 * log(Halstead Volume) –

0.23 * (Cyclomatic Complexity) –

16.2 * log(Lines of Code)

)*100 / 171)

## Calculation [ edit ]

For a given problem, let:

- $\eta_1$ = the number of distinct operators
- $\eta_2$ = the number of distinct operands
- $N_1$ = the total number of operators
- $N_2$ = the total number of operands

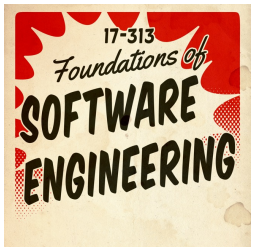
From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$



CRICOS PROVIDER #00120C

# Origins




- 1992 Paper at the International Conference on Software Maintenance by Paul Oman and Jack Hagemeister

$$171 - 5.2 ln(\text{HV}) - 0.23\text{CC} - 16.2 ln(\text{LOC}) + 50.0 sin\sqrt{2.46 * \text{COM}}$$

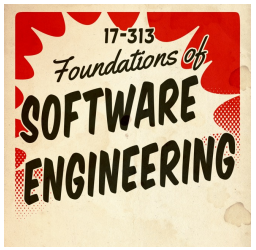
COM = percentage of comments

- Developers rated a number of HP systems in C and Pascal
- Statistical regression analysis to find key factors among 40 metrics



CRICOS PROVIDER #00120C

# The Index




Maintainability Index =

MAX(0,(171 –

5.2 * log(Halstead Volume) –

0.23 * (Cyclomatic Complexity) –

16.2 * log(Lines of Code)

)*100 / 171)



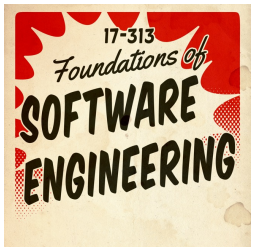
CRICOS PROVIDER #00120C

# Mini Break in Monday Lecture



CRICOS PROVIDER #00120C

# Thoughts?




- Metric seems attractive
- Easy to compute
- Often seems to match intuition
- Parameters seem almost arbitrar
  code (few developers, unclear sta
- All metrics related to size: just me
- Original 1992 C/Pascal programs
  Java/JS/C# code

ARIE VAN DEURSEN

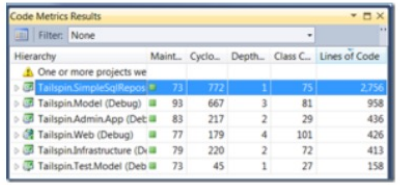
Software engineering in theory and practice

HOME ABOUT

29 AUG 2014

*Think Twice Before Using the "Maintainability Index"*

posted in Research by Arie van Deursen




This is a quick note about the "Maintainability Index", a metric aimed at assessing software maintainability, as I recently run into developers and researchers who are (still) using it.

The Maintainability Index was introduced at the International Conference on Software Maintenance in 1992. To date, it is included in Visual Studio (since 2007), in the recent (2012) JSComplexity and Radon metrics reporters for Javascript and Python, and in older metric tool suites such as verifysoft.

At first sight, this sounds like a great success of knowledge transfer from academic research to industry practice. Upon closer inspection, the Maintainability Index turns out to be problematic.

The Original Index



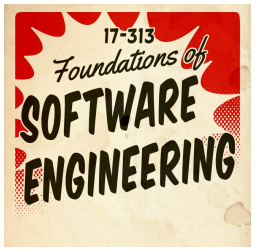
CRICOS PROVIDER #00120C

# CASE STUDY: AUTONOMOUS VEHICLE SAFETY



CRICOS PROVIDER #00120C

# How can we judge AV software quality (e.g. safety)?







CRICOS PROVIDER #00120C

# Test coverage


- Amount of code executed during testing.
- Statement coverage, line coverage, branch coverage, etc.
- E.g. 75% branch coverage → 3/4 if-else outcomes have been executed

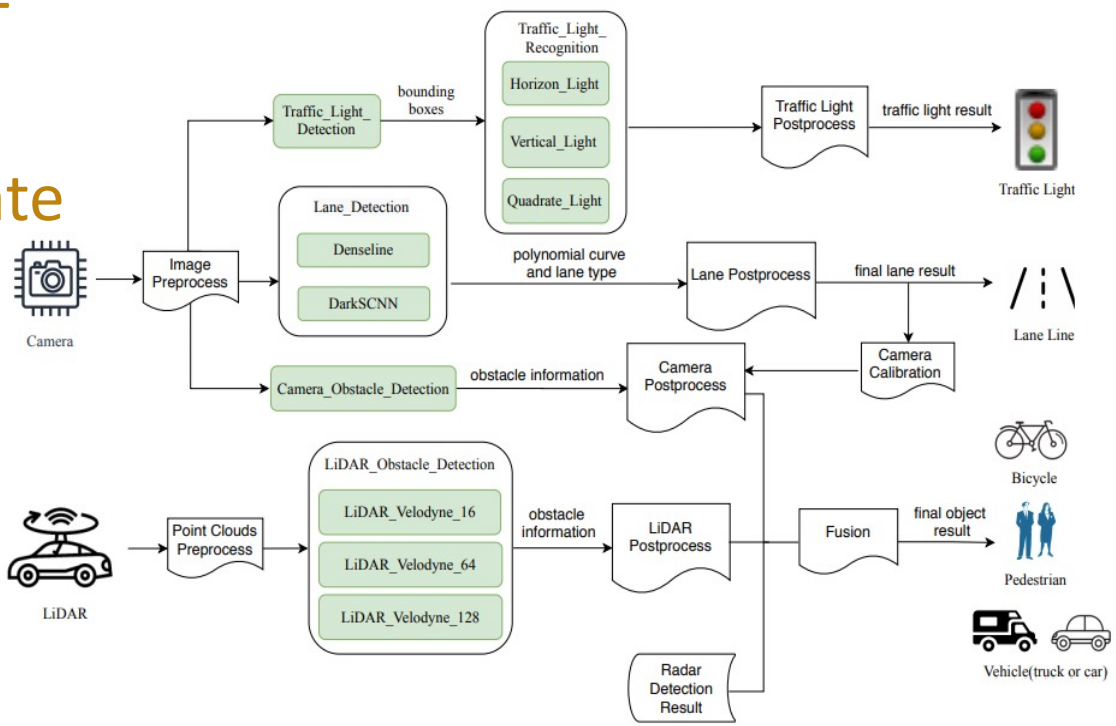
```
                :                
             1698 : const TrajectoryPoint& StGraphData::init_point() const { return init_point_; }
                :                
             2264 : const SpeedLimit& StGraphData::speed_limit() const { return speed_limit_; }
                :                
           212736 : double StGraphData::cruise_speed() const {
[ - + ]:   212736 :   return cruise_speed_ > 0.0 ? cruise_speed_ : FLAGS_default_cruise_speed;
                : }
                :                
             1698 : double StGraphData::path_length() const { return path_data_length_; }
                :                
             1698 : double StGraphData::total_time_by_conf() const { return total_time_by_conf_; }
                :                
             1698 : planning_internal::STGraphDebug* StGraphData::mutable_st_graph_debug() {
             1698 :   return st_graph_debug_;
                : }
                :                
              566 : bool StGraphData::SetSTDrivableBoundary(
                :     const std::vector<std::tuple<double, double, double>>& s_boundary,
                :     const std::vector<std::tuple<double, double, double>>& v_obs_info) {
[ + - ]:      566 :   if (s_boundary.size() != v_obs_info.size()) {
                :     return false;
                :   }
[ + + ]:    40752 :   for (size_t i = 0; i < s_boundary.size(); ++i) {
              :    80372 :     auto st_bound_instance = st_drivable_boundary_.add_st_boundary();
              :   160744 :     st_bound_instance->set_t(std::get<0>(s_boundary[i]));
              :   120558 :     st_bound_instance->set_s_lower(std::get<1>(s_boundary[i]));
              :   120558 :     st_bound_instance->set_s_upper(std::get<2>(s_boundary[i]));
[ - + ]:    40186 :     if (std::get<1>(v_obs_info[i]) > -kObsSpeedIgnoreThreshold) {
              :        0 :       st_bound_instance->set_v_obs_lower(std::get<1>(v_obs_info[i]));
              :          :     }
[ + + ]:    40186 :     if (std::get<2>(v_obs_info[i]) < kObsSpeedIgnoreThreshold) {
              :    50254 :       st_bound_instance->set_v_obs_upper(std::get<2>(v_obs_info[i]));
              :          :     }
              :          :   }
```



CRICOS PROVIDER #00120C

# Model Accuracy

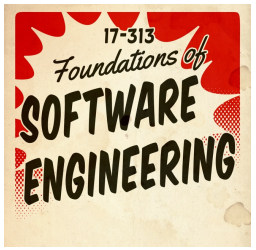

- Train machine-learning models on labelled data (sensor data + ground truth).
- Compute accuracy on a separate labelled test set.
- E.g. 90% accuracy implies that object recognition is right for 90% of the test inputs.






CRICOS PROVIDER #00120C

# Failure Rate




- Frequency of crashes/fatalities
- Per 1000 rides, per million miles, per month (in the news)




CRICOS PROVIDER #00120C

# Mileage

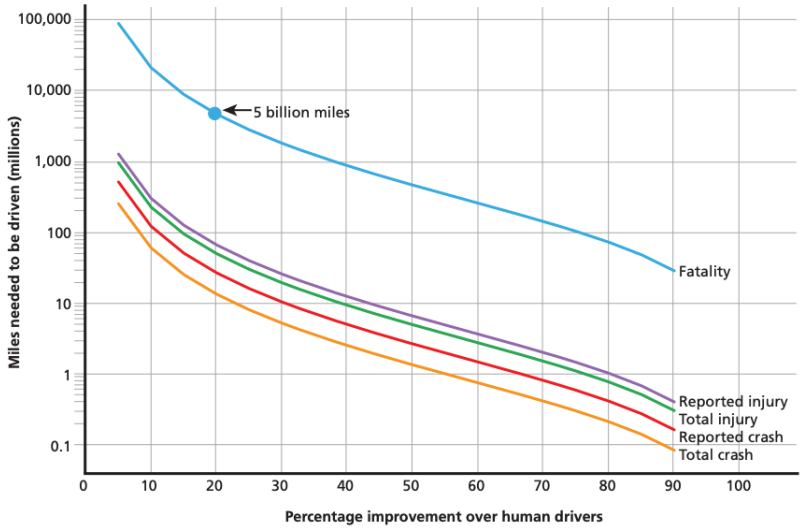

RAND CORPORATION

## Driving to Safety

**How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?**

*Nidhi Kalra, Susan M. Paddock*

**Figure 3. Miles Needed to Demonstrate with 95% Confidence that the Autonomous Vehicle Failure Rate Is Lower than the Human Driver Failure Rate**



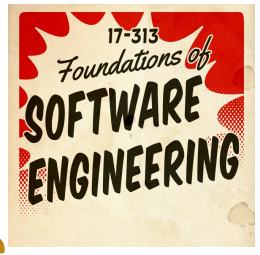


Source: waymo.com/safety (September 2021)



CRICOS PROVIDER #00120C

# Activity



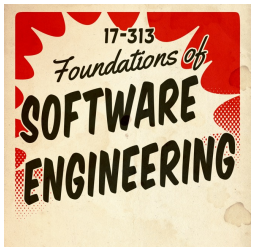

Think of “pros” and “cons” for using various quality metrics to judge AV software.

- Test coverage
- Model accuracy
- Failure rate
- Mileage
- Size of codebase
- Age of codebase
- Time of most recent change
- Frequency of code releases
- Number of contributors
- Amount of code documentation



CRICOS PROVIDER #00120C

# STOP sign or 45 speed limit?







"Robust Physical-World Attacks on Deep Learning Models" by Kevin Eykholt et al. CVPR'18



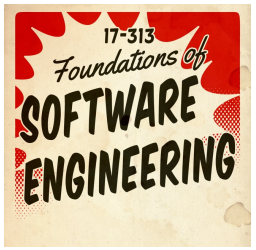
CRICOS PROVIDER #00120C

# MEASUREMENT FOR DECISION MAKING IN SOFTWARE DEVELOPMENT



CRICOS PROVIDER #00120C

# What is Measurement?




- Measurement is the empirical, objective assignment of numbers, according to a rule derived from a model or theory, to attributes of objects or events with the intent of describing them. – Craner, Bond, “Software Engineering Metrics: What Do They Measure and How Do We Know?”
- A quantitatively expressed reduction of uncertainty based on one or more observations. – Hubbard, “How to Measure Anything …”



CRICOS PROVIDER #00120C

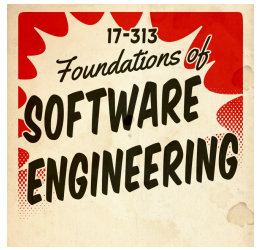
# Software Quality Metrics

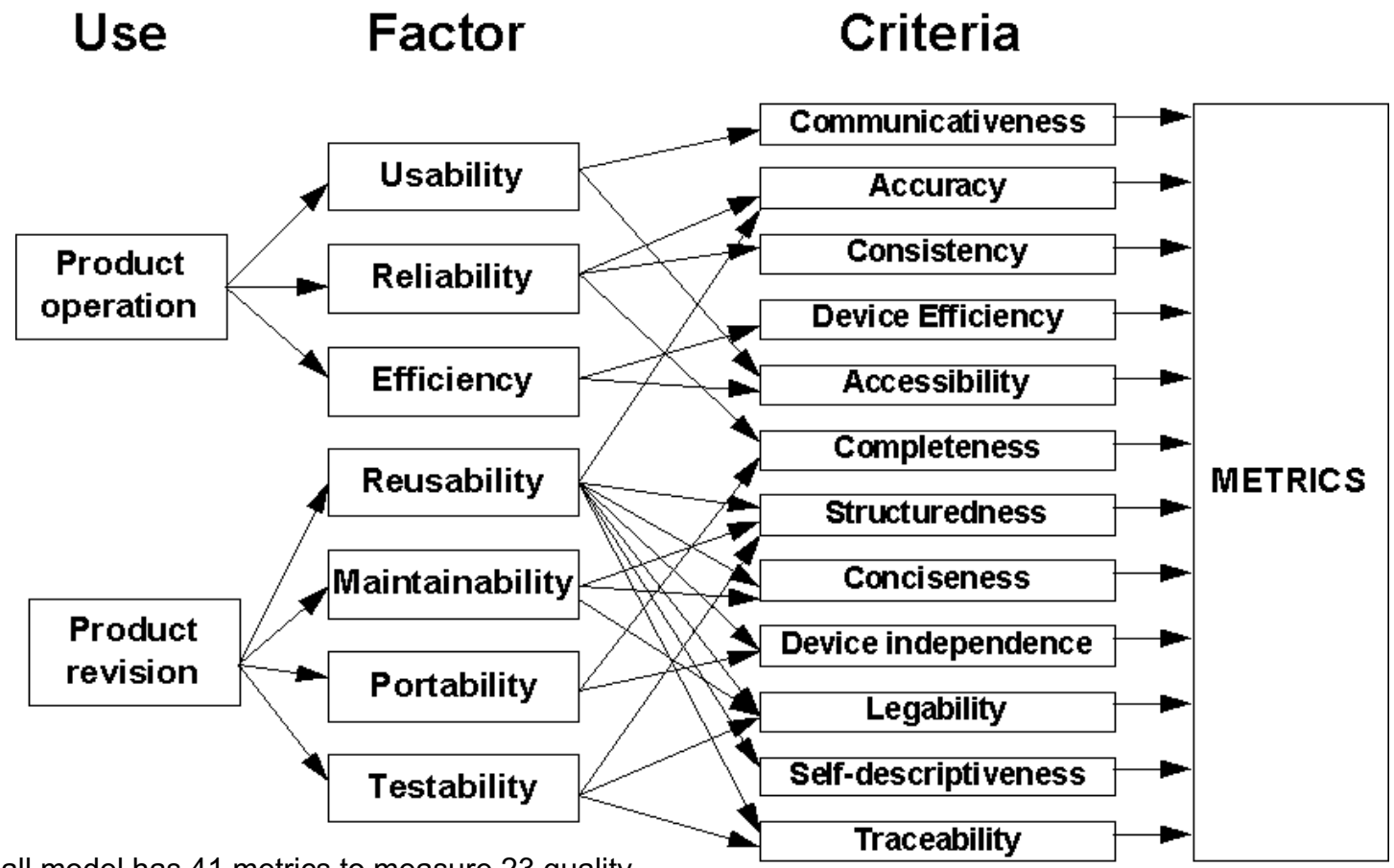

- IEEE 1061 definition: “A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given attribute that affects its quality.”
- Metrics have been proposed for many quality attributes; may define own metrics



CRICOS PROVIDER #00120C

# External attributes: Measuring Quality



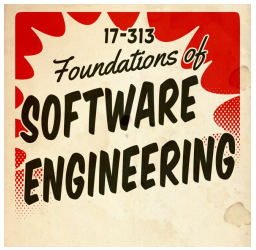



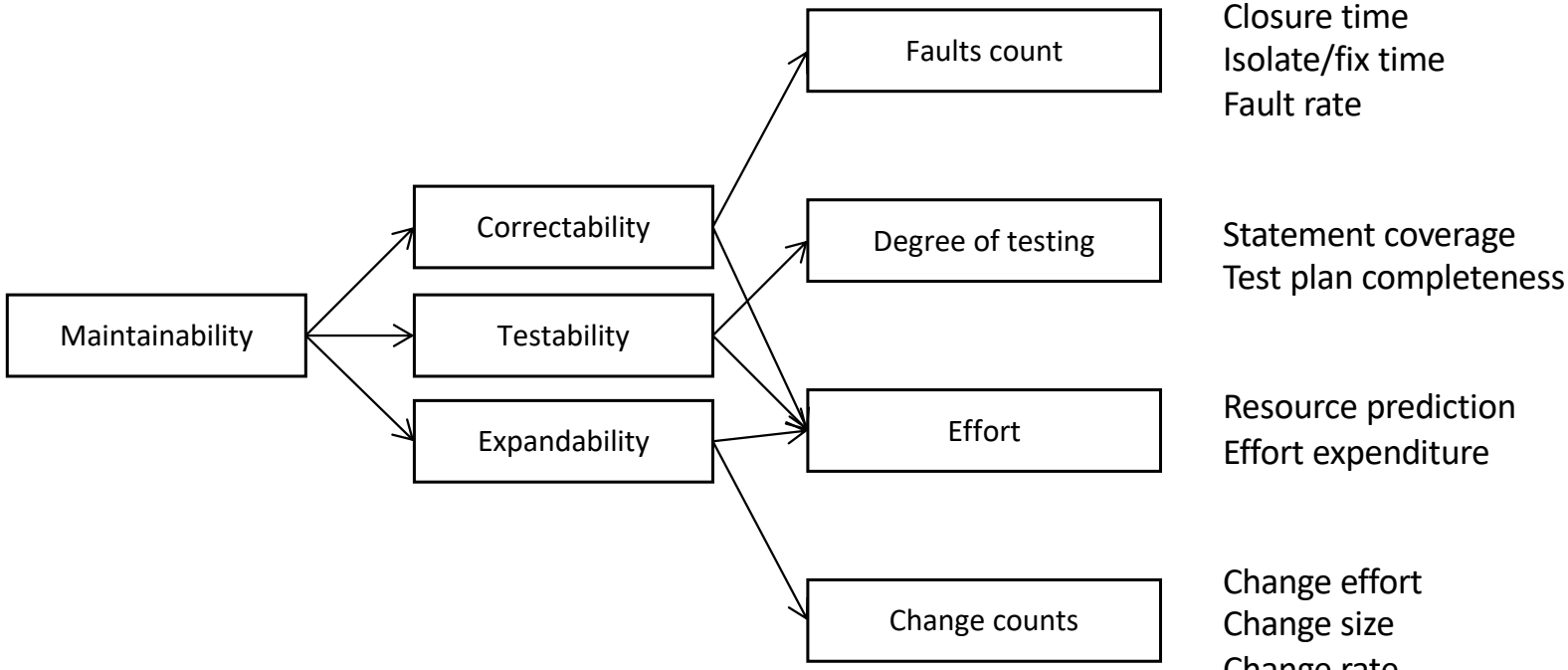

McCall model has 41 metrics to measure 23 quality criteria from 11 factors



CRICOS PROVIDER #00120C

# Decomposition of Metrics





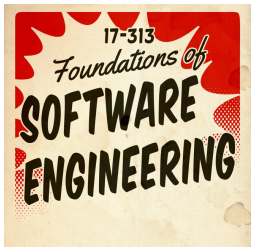

CRICOS PROVIDER #00120C

# EXAMPLES:
# CODE COMPLEXITY



CRICOS PROVIDER #00120C

# Lines of Code




- Easy to measure

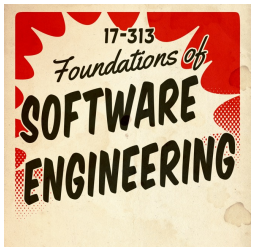
```
> wc –l file1 file2…
```

| LOC | projects |
|---|---|
| 450 | Expression Evaluator |
| 2,000 | Sudoku |
| 100,000 | Apache Maven |
| 500,000 | Git |
| 3,000,000 | MySQL |
| 15,000,000 | gcc |
| 50,000.000 | Windows 10 |
| 2,000,000,000 | Google (MonoRepo) |



CRICOS PROVIDER #00120C

# Normalising Lines of Code




- Ignore comments and empty lines
- Ignore lines < 2 characters
- Pretty print source code first
- Count statements (logical lines of code)
- See also: cloc

```
for (i = 0; i < 100; i += 1) printf("hello"); /* How many lines of code is this? */
```

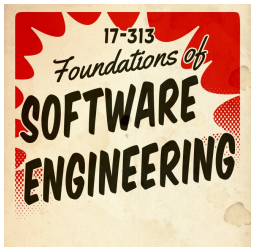
```
/* How many lines of code is this? */

for (
        i = 0;
        i < 100;
        i += 1
    ) {
        printf("hello");
}
```



CRICOS PROVIDER #00120C

# Normalisation per Language




| Language | Statement factor (productivity) | Line factor |
|---|---|---|
| C | 1 | 1 |
| C++ | 2.5 | 1 |
| Fortran | 2 | 0.8 |
| Java | 2.5 | 1.5 |
| Perl | 6 | 6 |
| Smalltalk | 6 | 6.25 |
| Python | 6 | 6.5 |

Source: "Code Complete: A Practical Handbook of Software Construction", S. McConnell, Microsoft Press (2004) and http://www.codinghorror.com/blog/2005/08/are-all-programming-languages-the-same.html u.a.



CRICOS PROVIDER #00120C

# Halstead Volume

- Introduced by Maurice Howard Halstead in 1977
- Halstead Volume =
  number of operators/operands * log2(number of distinct operators/operands)
- Approximates size of elements and vocabulary

## Calculation [ edit ]

For a given problem, let:

- $\eta_1$ = the number of distinct operators
- $\eta_2$ = the number of distinct operands
- $N_1$ = the total number of operators
- $N_2$ = the total number of operands

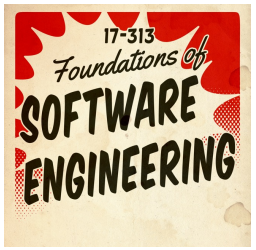
From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$



CRICOS PROVIDER #00120C

# Halstead Volume – Example (Do At Home)



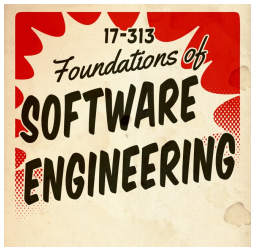

```
main() {
        int a, b, c, avg;
        scanf("%d %d %d", &a, &b, &c);
        avg = (a + b + c) / 3;
        printf("avg = %d", avg);
}
```

Operators/Operands: main, (), {}, int, a, b, c, avg, scanf, (), "…", &, a, &, b, &, c, avg, =, a, +, b, +, c, (), /, 3, printf, (), "…", avg



CRICOS PROVIDER #00120C

# Cyclomatic Complexity




- Proposed by McCabe 1976
- Based on control flow graph, measures linearly independent paths through a program
  - ~= number of decisions
  - Number of test cases needed to achieve branch coverage

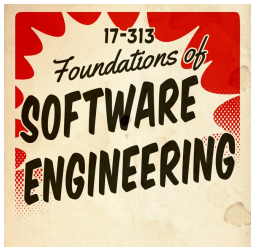
```
if (c1) {
        f1();
} else {
        f2();
}
if (c2) {
        f3();
} else {
        f4();
}
```

*"For each module, either limit cyclomatic complexity to [X] or provide a written explanation of why the limit was exceeded."*
– NIST Structured Testing methodology



CRICOS PROVIDER #00120C

# Object-Oriented Metrics




- Number of Methods per Class
- Depth of Inheritance Tree
- Number of Child Classes
- Coupling between Object Classes
- Calls to Methods in Unrelated Classes
- …



CRICOS PROVIDER #00120C

# What software qualities do we care about? (examples)


- Scalability
- Security
- Extensibility
- Documentation
- Performance
- Consistency
- Portability
- Installability
- Maintainability
- Functionality (e.g., data integrity)
- Availability
- Ease of use



CRICOS PROVIDER #00120C

# What process qualities do we care about? (examples)

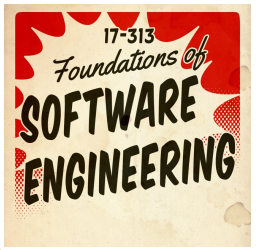

- On-time release
- Development speed
- Meeting efficiency
- Conformance to processes
- Time spent on rework
- Reliability of predictions
- Fairness in decision making

- Measure time, costs, actions, resources, and quality of work packages; compare with predictions
- Use information from issue trackers, communication networks, team structures, etc…



CRICOS PROVIDER #00120C

# Everything is measurable



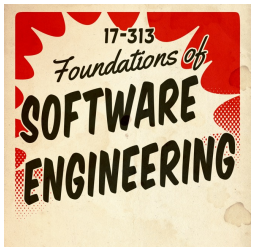

- If X is something we care about, then X, by definition, must be detectable.
  - How could we care about things like “quality,” “risk,” “security,” or “public image” if these things were totally undetectable, directly or indirectly?
  - If we have reason to care about some unknown quantity, it is because we think it corresponds to desirable or undesirable results in some way.
- If X is detectable, then it must be detectable in some amount.
  - If you can observe a thing at all, you can observe more of it or less of it
- If we can observe it in some amount, then it must be measurable.

D. Hubbard, How to Measure Anything, 2010



CRICOS PROVIDER #00120C

# Measurement for Decision Making

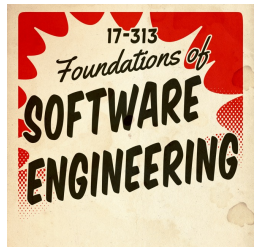


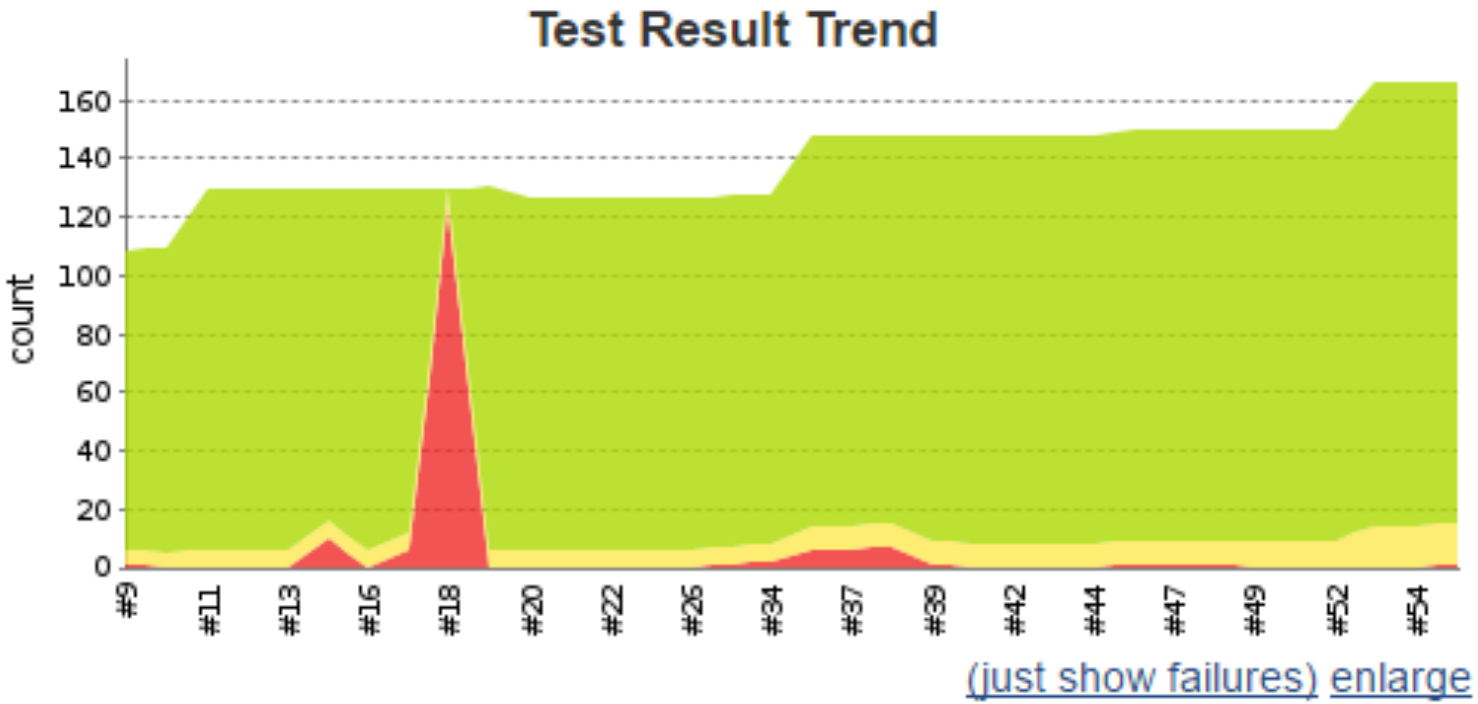

- Fund project?
- More testing?
- Fast enough? Secure enough?
- Code quality sufficient?
- Which feature to focus on?
- Developer bonus?
- Time and cost estimation? Predictions reliable?



CRICOS PROVIDER #00120C

# Trend analyses

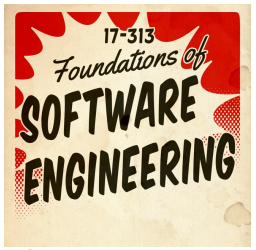


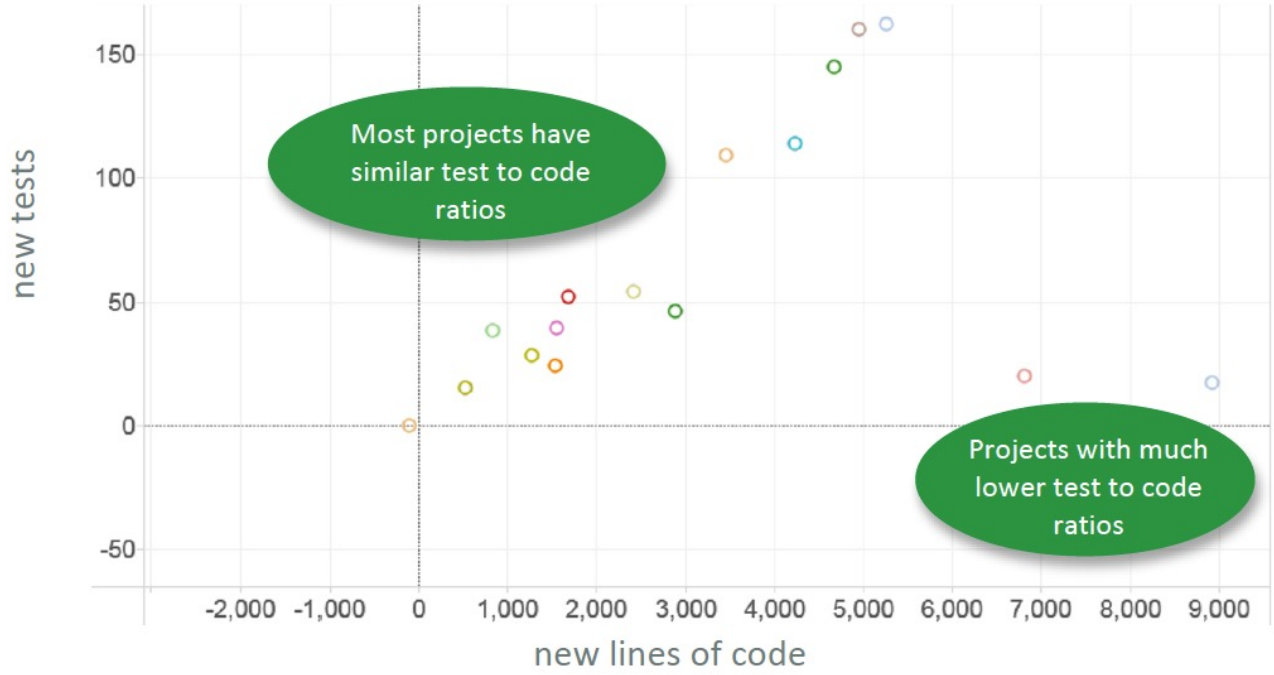



CRICOS PROVIDER #00120C

# Benchmark-Based Metrics



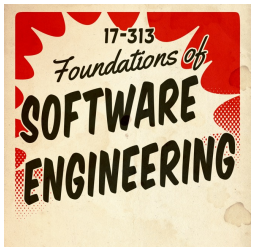

- Monitor many projects or many modules, get typical values for metrics
- Report deviations



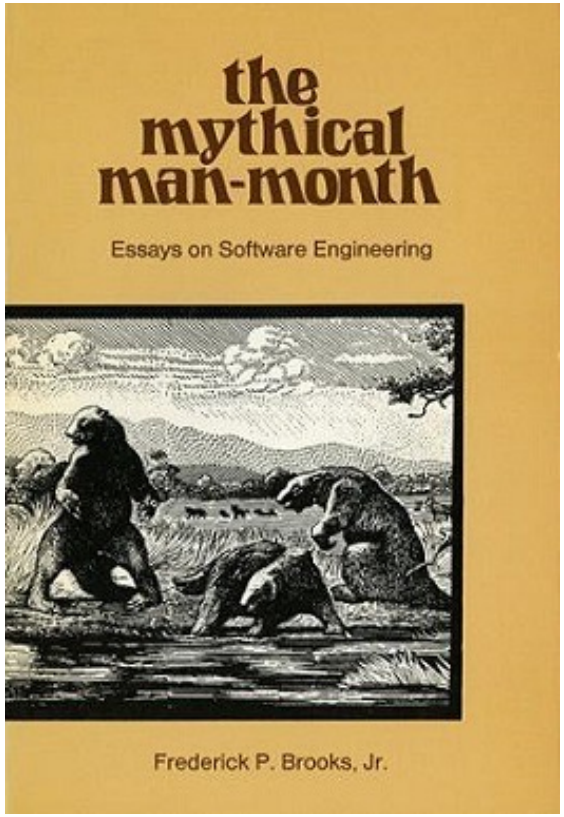



CRICOS PROVIDER #00120C

# Example: Antipattern in effort estimation



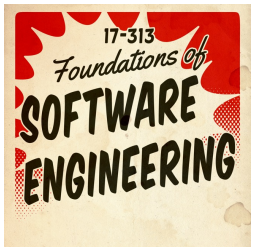

- IBM in the 60's: Would account in "person-months"
  e.g. Team of 2 working 3 months
  = 6 person-months
- LoC ~ Person-months ~ $$$
- Brooks: "*Adding manpower to a late software project makes it later.*"






CRICOS PROVIDER #00120C

# Questions to consider




- What properties do we care about, and how do we measure it?
- What is being measured? Does it (to what degree) capture the thing you care about? What are its limitations?
- How should it be incorporated into process? Check in gate? Once a month? Etc.
- What are potentially negative side effects or incentives?

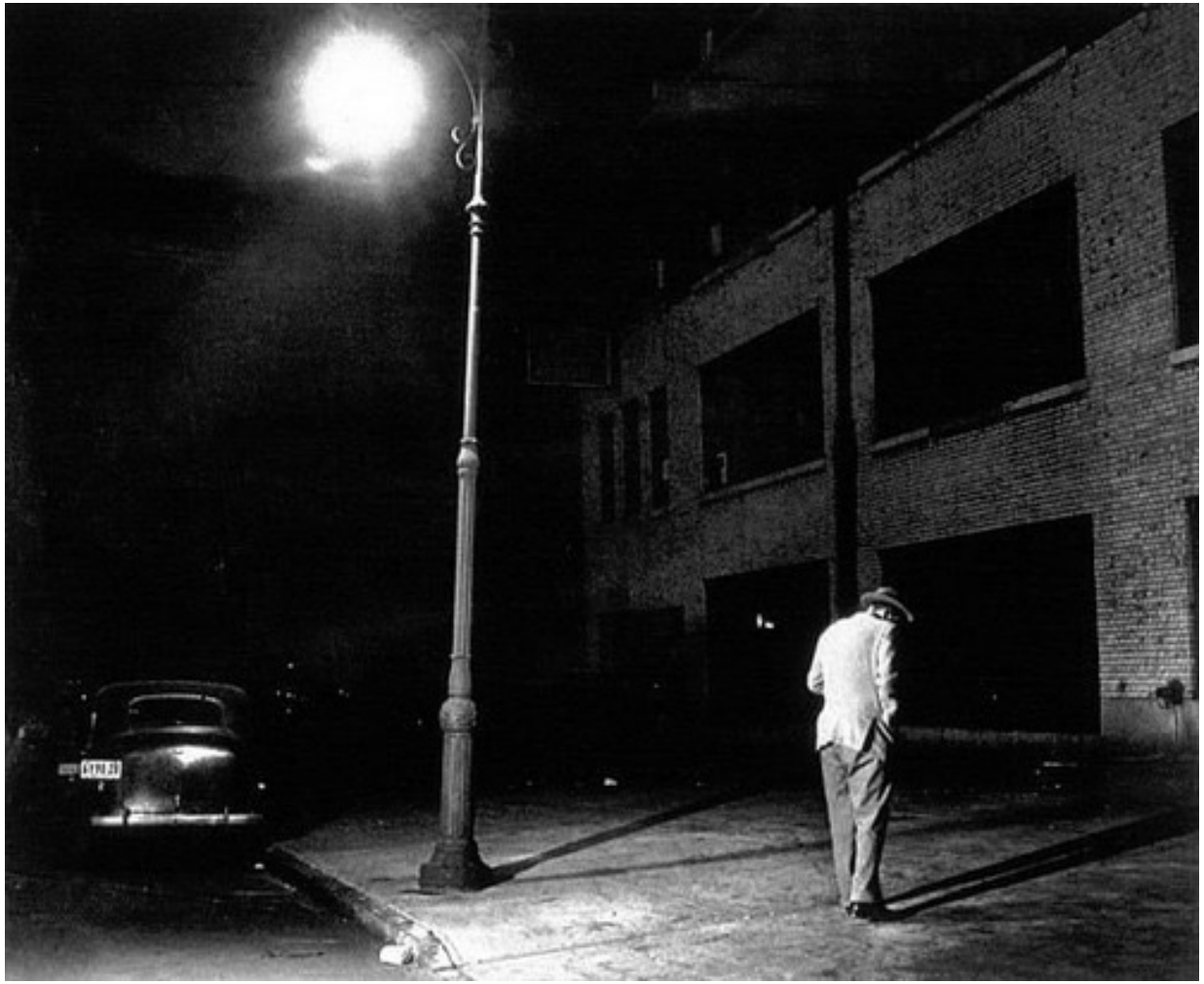


CRICOS PROVIDER #00120C

# MEASUREMENT IS DIFFICULT



CRICOS PROVIDER #00120C





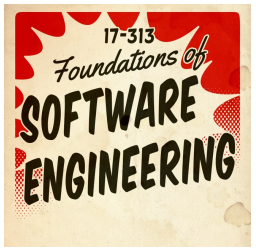
CRICOS PROVIDER #00120C

# The streetlight effect

- A known observational bias.
- People tend to look for something only where it's easiest to do so.
  - If you drop your keys at night, you'll tend to look for it under streetlights.



CRICOS PROVIDER #00120C

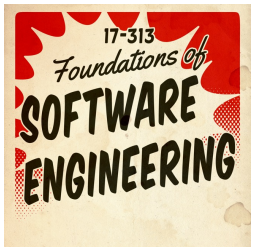






CRICOS PROVIDER #00120C

# What could possibly go wrong?




- Bad statistics: A basic misunderstanding of measurement theory and what is being measured.
- Bad decisions: The incorrect use of measurement data, leading to unintended side effects.
- Bad incentives: Disregard for the human factors, or how the cultural change of taking measurements will affect people.



CRICOS PROVIDER #00120C

# Lies, damned lies, and…


- In 1995, the UK Committee on Safety of Medicines issued the following warning: "third-generation oral contraceptive pills increased the risk of potentially life-threatening blood clots in the legs or lungs twofold -- that is, by 100 percent”



CRICOS PROVIDER #00120C

# ...statistics


- "...of every 7,000 eration oral contraceptiv umber increased to two pills..."
- "...The absolute r s the relative increase s) was indeed 100 perce

## COVID-19 Vaccines: Myth Versus Fact

**Featured Experts:**

Gabor David Kelen, M.D.

Lisa Maragakis, M.D., M.P.H.

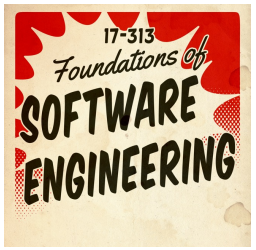
*Updated on March 10, 2022*

Now that the U.S. Food and Drug Administration has authorized vaccines for COVID-19, and their distribution has begun, Lisa Maragakis, M.D., M.P.H., senior director of infection prevention, and Gabor Kelen, M.D.,



CRICOS PROVIDER #00120C

# Measurement scales




- Scale: the type of data being measured.
- The scale dictates what sorts of analysis/arithmetic is legitimate or meaningful.
- Your options are:
  - Nominal: categories
  - Ordinal: order, but no magnitude.
  - Interval: order, magnitude, but no zero.
  - Ratio: Order, magnitude, and zero.
  - Absolute: special case of ratio.



CRICOS PROVIDER #00120C

# Nominal/categorical scale


- Entities classified with respect to a certain attribute. Categories are jointly exhaustive and mutually exclusive.
  - No implied order between categories!
- Categories can be represented by labels or numbers; however, they do not represent a magnitude, arithmetic operation have no meaning.
- Can be compared for identity or distinction, and measurements can be obtained by counting the frequencies in each category. Data can also be aggregated.

| Entity | Attribute | Categories |
|---|---|---|
| Application | Purpose | E-commerce, CRM, Finance |
| Application | Language | Java, Python, C++, C# |
| Fault | Source | assignment, checking, algorithm, function, interface, timing |



CRICOS PROVIDER #00120C

# Ordinal scale

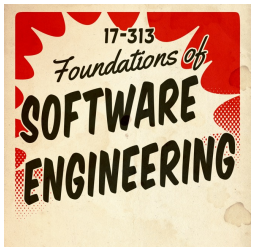

- Ordered categories: maps a measured attribute to an ordered set of values, but no information about the magnitude of the differences between elements.
- Measurements can be represented by labels or numbers, BUT: if numbers are used, *they do not represent a magnitude.*
  - Honestly, try not to do that. It eliminates temptation.
- You cannot: add, subtract, perform averages, etc (arithmetic operations are out).
- You can: compare with operators (like “less than” or “greater than”), create ranks for the purposes of rank correlations (Spearman’s coefficient, Kendall’s τ).

| Entity | Attribute | Values |
|---|---|---|
| Application | Complexity | Very Low, Low, Average, High, Very High |
| Fault | Severity | 1 – Cosmetic, 2 – Moderate, 3 – Major, 4 – Critical |



CRICOS PROVIDER #00120C

# Interval scale




- Has order (like ordinal scale) and magnitude.
  - The intervals between two consecutive integers represent equal amounts of the attribute being measured.
- Does NOT have a zero: 0 is an arbitrary point, and doesn't correspond to the absence of a quantity.
- Most arithmetic (addition, subtraction) is OK, as are mean and dispersion measurements, as are Pearson correlations. Ratios are not meaningful.
  - Ex: The temperature yesterday was 64 F, and today is 32 F. Is today twice as cold as yesterday?
- Incremental variables (quantity as of today – quantity at an earlier time) and preferences are commonly measured in interval scales.



CRICOS PROVIDER #00120C

# Ratio Scale

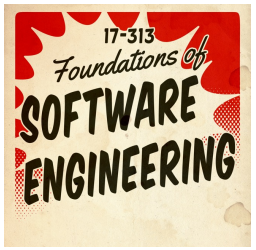

- An interval scale that has a true zero that actually represents the absence of the quantity being measured.
- All arithmetic is meaningful.
- Absolute scale is a special case, measurement simply made by counting the number of elements in the object.
  - Takes the form “number of occurrences of X in the entity.”

| Entity | Attribute | Values |
|---|---|---|
| Project | Effort | Real numbers |
| Software | Complexity | Cyclomatic complexity |



CRICOS PROVIDER #00120C

# Summary of Scales

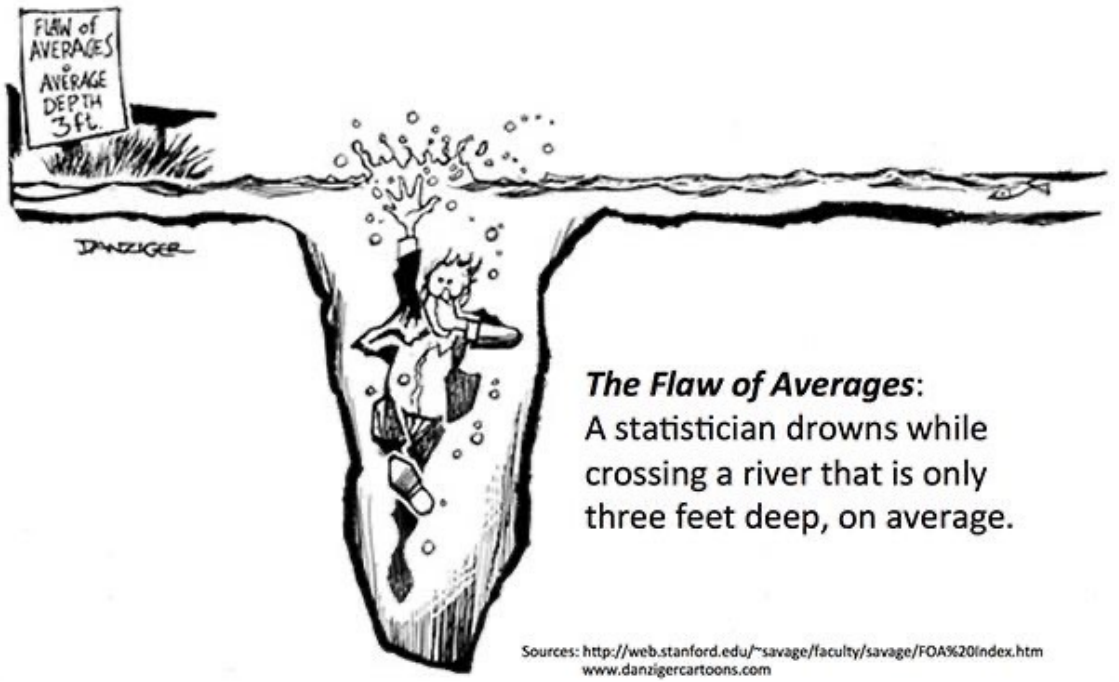



| Scale level | Examples | Operators | Possible analyses |
|---|---|---|---|
| *Quantitative scales* | | | |
| **Ratio** | size, time, cost | $*, /, \log, \sqrt{}$ | geometric mean, coefficient of variation |
| **Interval** | temperature, marks, judgement expressed on rating scales | $+, -$ | mean, variance, correlation, linear regression, analysis of variance (ANOVA), ... |
| *Qualitative scales* | | | |
| **Ordinal** | complexity classes | $<, >$ | median, rank correlation, ordinal regression |
| **Nominal** | feature availability | $=, \neq$ | frequencies, mode, contingency tables |



CRICOS PROVIDER #00120C

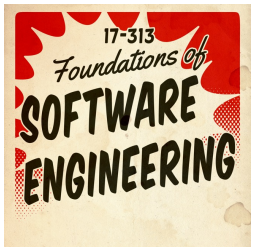
# UNDERSTAND YOUR DATA



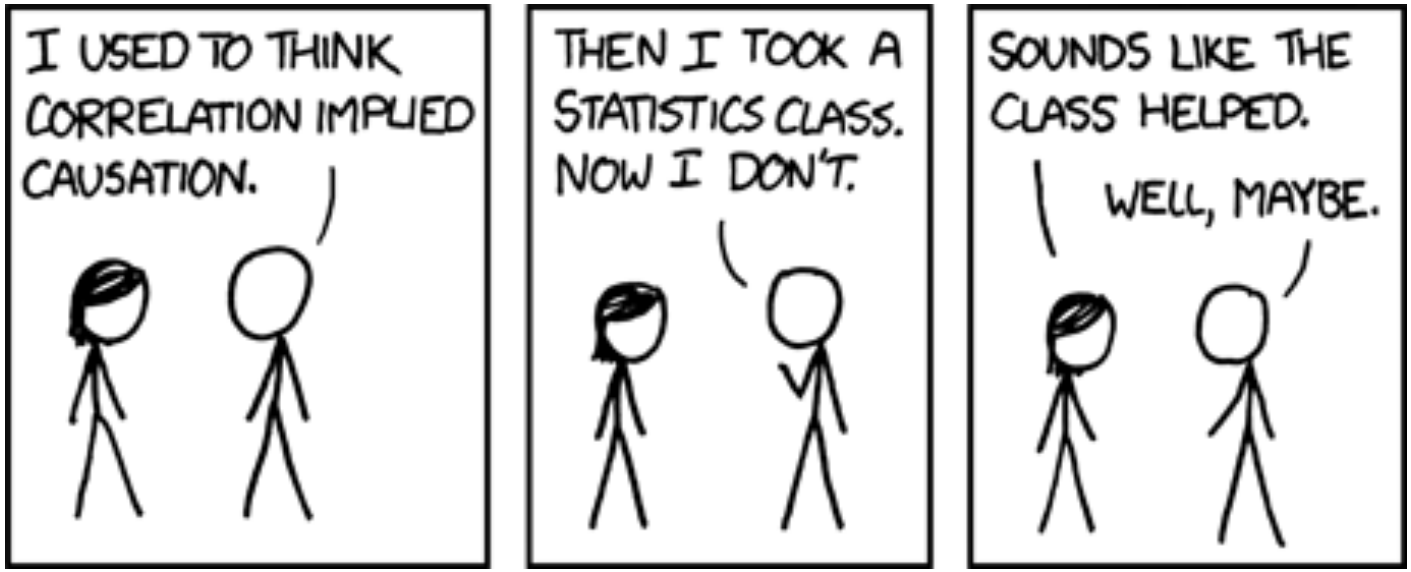



CRICOS PROVIDER #00120C

# For Causation




- Provide a theory (from domain knowledge, independent of data)
- Show correlation
- Demonstrate ability to predict new cases (replicate/validate)

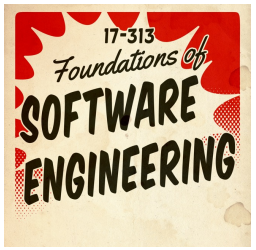


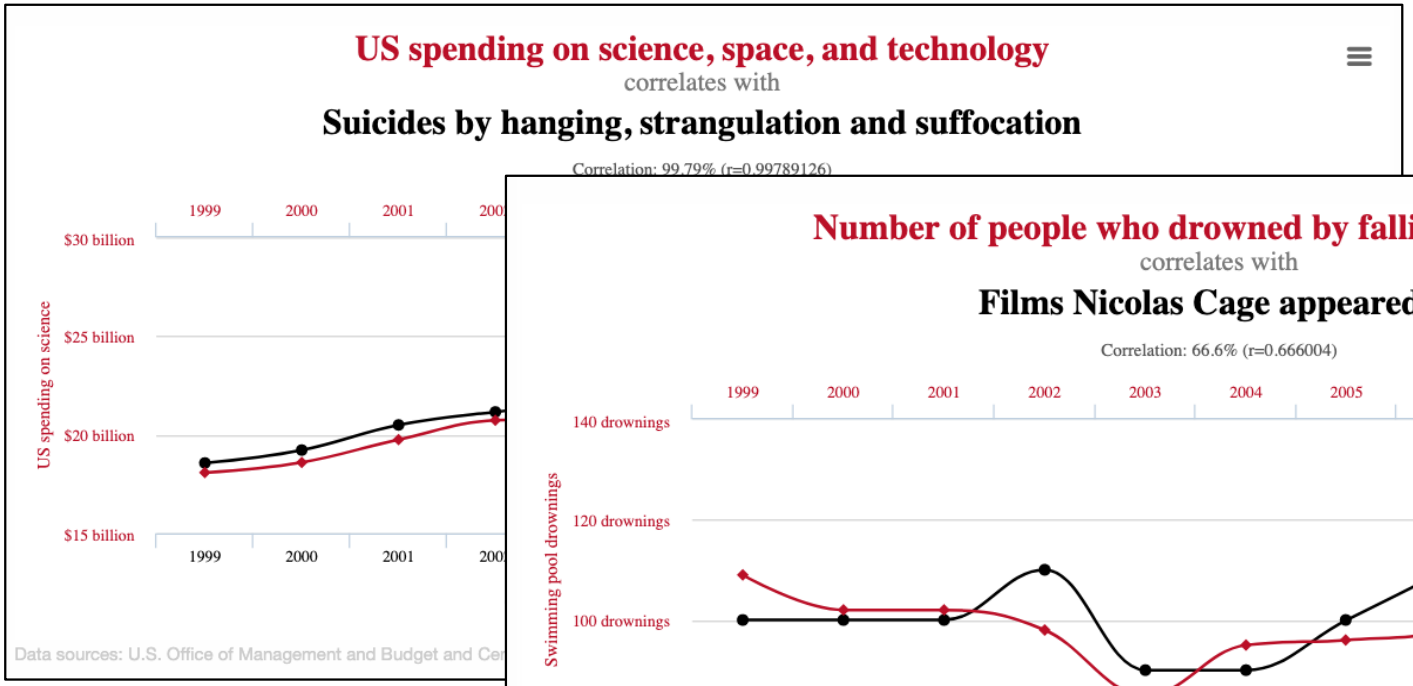

http://xkcd.com/552/

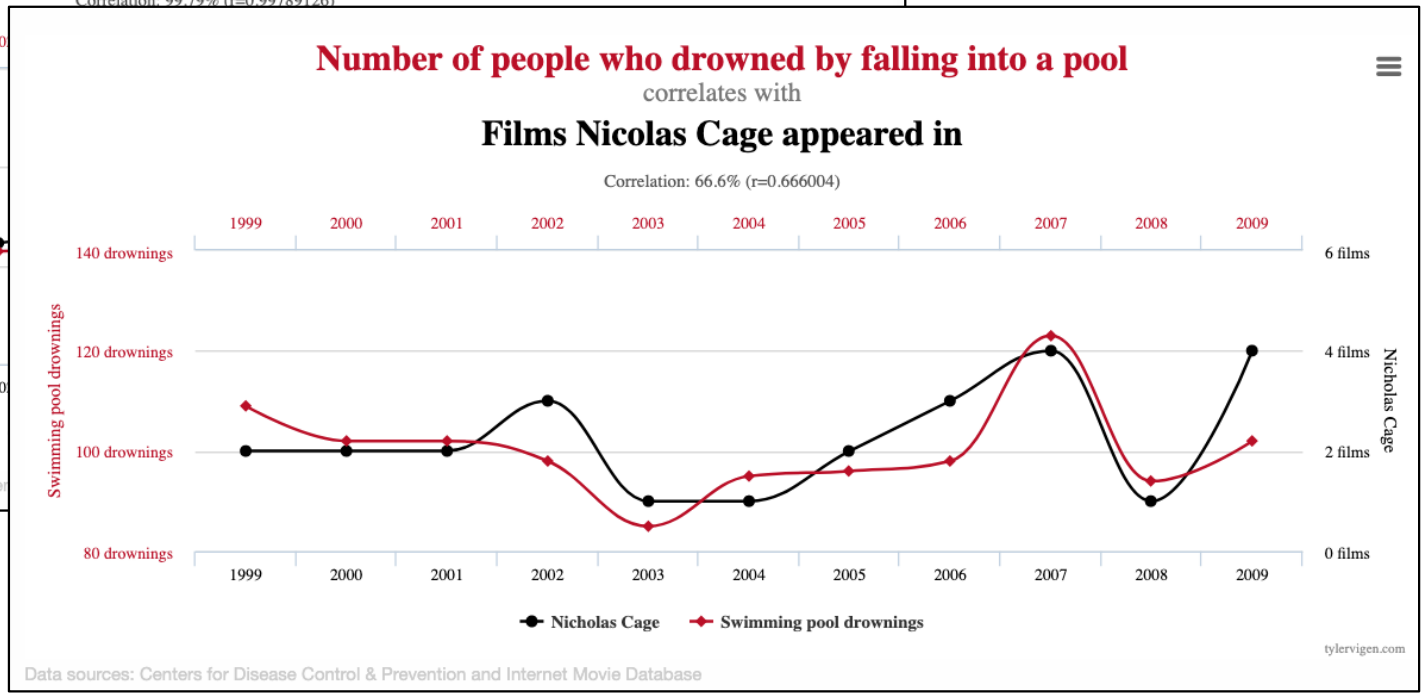


CRICOS PROVIDER #00120C

# Spurious Correlations



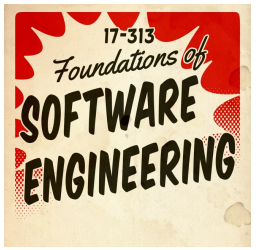



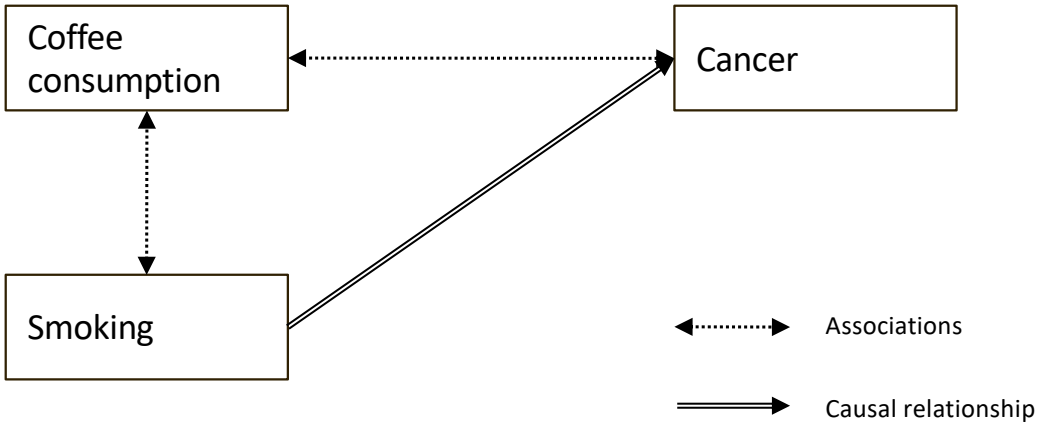






CRICOS PROVIDER #00120C

# Confounding variables




- If you look only at the coffee consumption → cancer relationship, you can get very misleading results
- Smoking is a confounder



CRICOS PROVIDER #00120C


RESEARCH-ARTICLE

# Coverage is not strongly correlated with test suite effectiveness

**Authors:** Laura Inozemtseva, Reid Holmes Authors Info & Affiliations

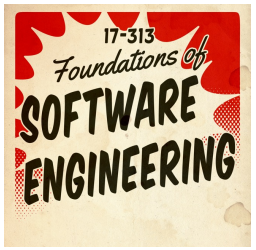
ICSE 2014: Proceedings of the 36th International Conference on Software Engineering • May 2014 • Pages 435–445 • https://doi.org/10.1145/2568225.2568271

"We found that there is a low to moderate correlation between coverage and effectiveness when the number of test cases in the suite is controlled for."



CRICOS PROVIDER #00120C

# Measurements validity

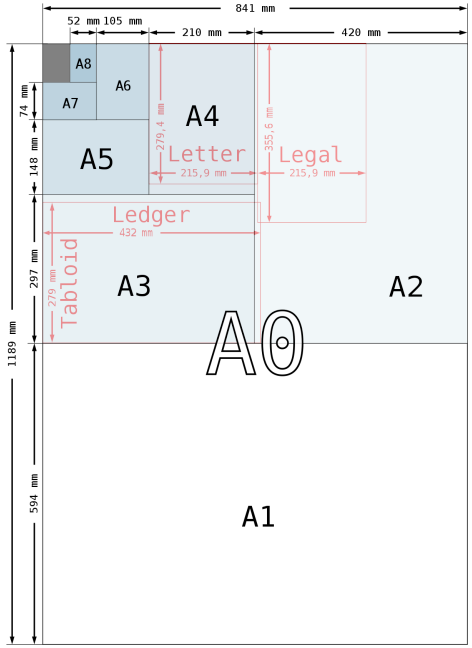



- *Construct validity* – Are we measuring what we intended to measure?
- *Internal validity* – The extent to which the measurement can be used to explain some other characteristic of the entity being measured
- *External validity* – Concerns the generalization of the findings to contexts and environments, other than the one studied



CRICOS PROVIDER #00120C

# Measurements reliability






CRICOS PROVIDER #00120C

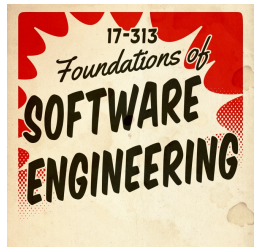
# Measurements reliability


- Extent to which a measurement yields similar results when applied multiple times
- Goal is to reduce uncertainty, increase consistency
- Example: Performance
  - Time, memory usage
  - Cache misses, I/O operations, instruction execution count, etc.
- Law of large numbers
  - Taking multiple measurements to reduce error
  - Trade-off with cost



CRICOS PROVIDER #00120C

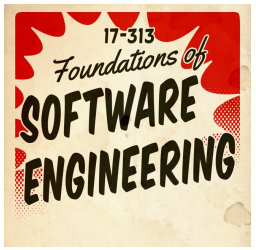
# The McNamara Fallacy






CRICOS PROVIDER #00120C

# The McNamara Fallacy



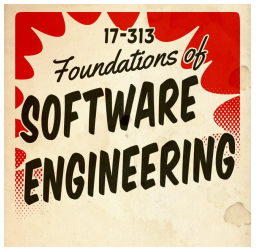

- Measure whatever can be easily measured.
- Disregard that which cannot be measured easily.
- Presume that which cannot be measured easily is not important.
- Presume that which cannot be measured easily does not exist.

https://chronotopeblog.com/2015/04/04/the-mcnamara-fallacy-and-the-problem-with-numbers-in-education/



CRICOS PROVIDER #00120C

# The McNamara Fallacy



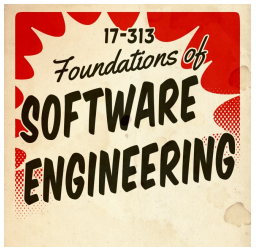

There seems to be a general misunderstanding to the effect that a mathematical model cannot be undertaken until every constant and functional relationship is known to high accuracy. This often leads to the omission of admittedly highly significant factors (most of the "intangibles" influences on decisions) because these are unmeasured or unmeasurable. To omit such variables is equivalent to saying that they have zero effect... Probably the only value known to be wrong…

J. W. Forrester, Industrial Dynamics, The MIT Press, 1961



CRICOS PROVIDER #00120C

# Defect Density




- Defect density = Known bugs / line of code
- System spoilage = time to fix post-release defects / total system development time
- Post-release vs pre-release
- What counted as defect? Severity? Relevance?
- What size metric used?
- What quality assurance mechanisms used?
- Little reference data publicly available; typically 2-10 defects/1000 lines of code



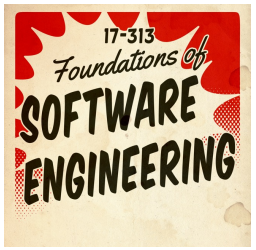
CRICOS PROVIDER #00120C

# DISCUSSION: MEASURING USABILITY



CRICOS PROVIDER #00120C

# Example: Measuring usability.




- Automated measures on code repositories
- Use or collect process data
- Instrument program (e.g., in-field crash reports)
- Surveys, interviews, controlled experiments, expert judgment
- Statistical analysis of sample



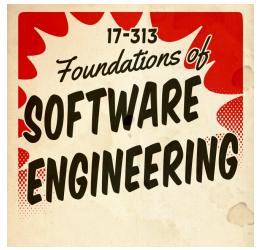
CRICOS PROVIDER #00120C

# METRICS AND INCENTIVES



CRICOS PROVIDER #00120C

# Goodhart's Law



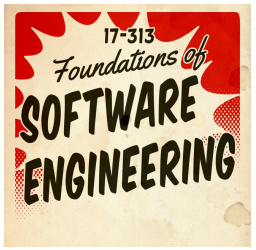

"When a measure becomes a target, it ceases to be a good measure."


http://dilbert.com/strips/comic/1995-11-13/



CRICOS PROVIDER #00120C

# Productivity Metrics

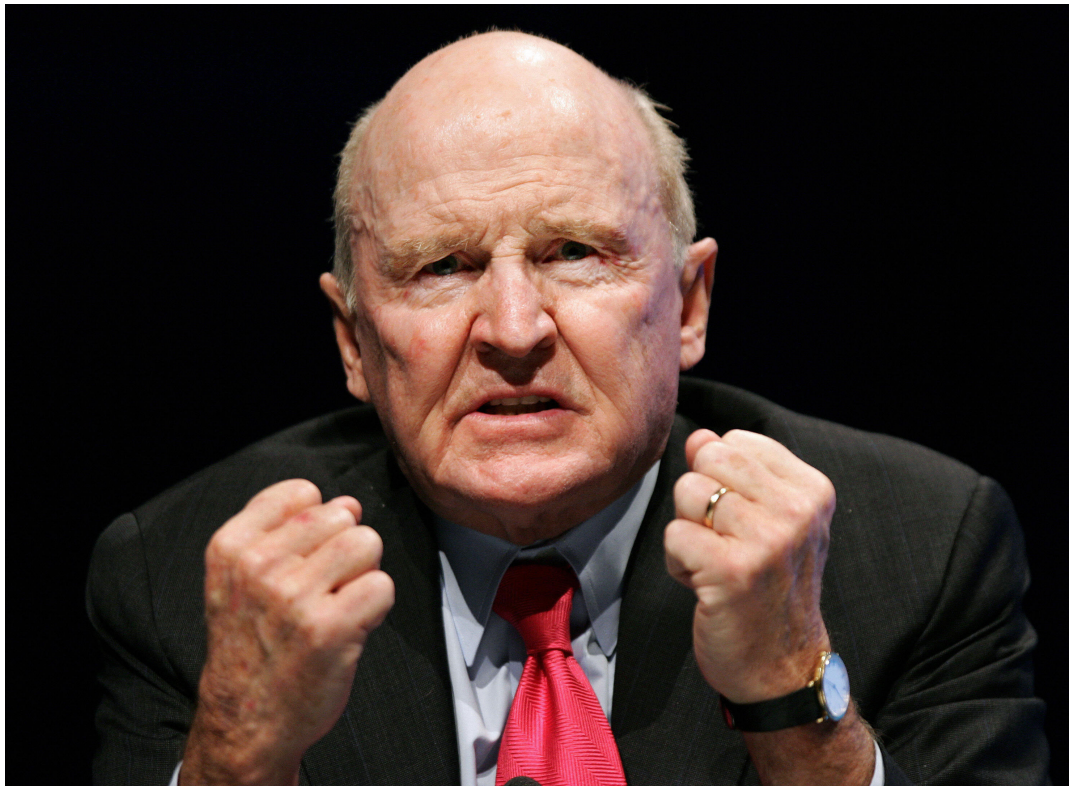



- Lines of code per day?
  - Industry average 10-50 lines/day
  - Debugging + rework ca. 50% of time
- Function/object/application points per month
- Bugs fixed?
- Milestones reached?



CRICOS PROVIDER #00120C

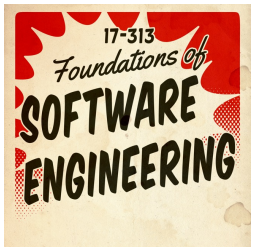
# Stack Ranking





CRICOS PROVIDER #00120C

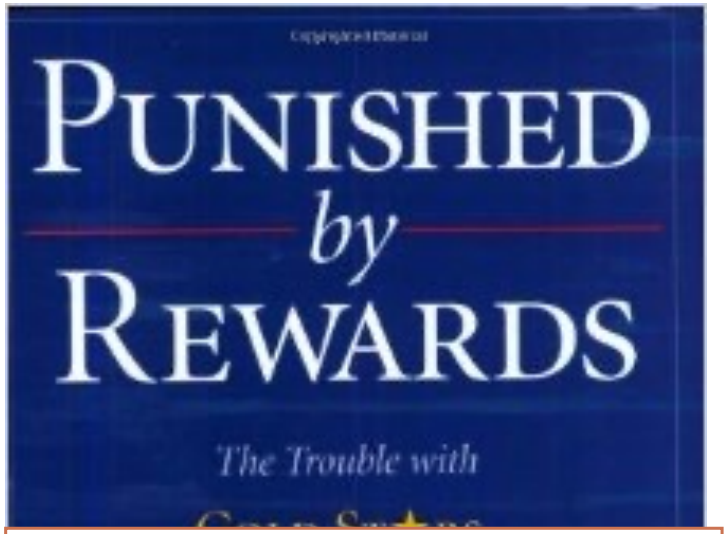
# Incentivizing Productivity



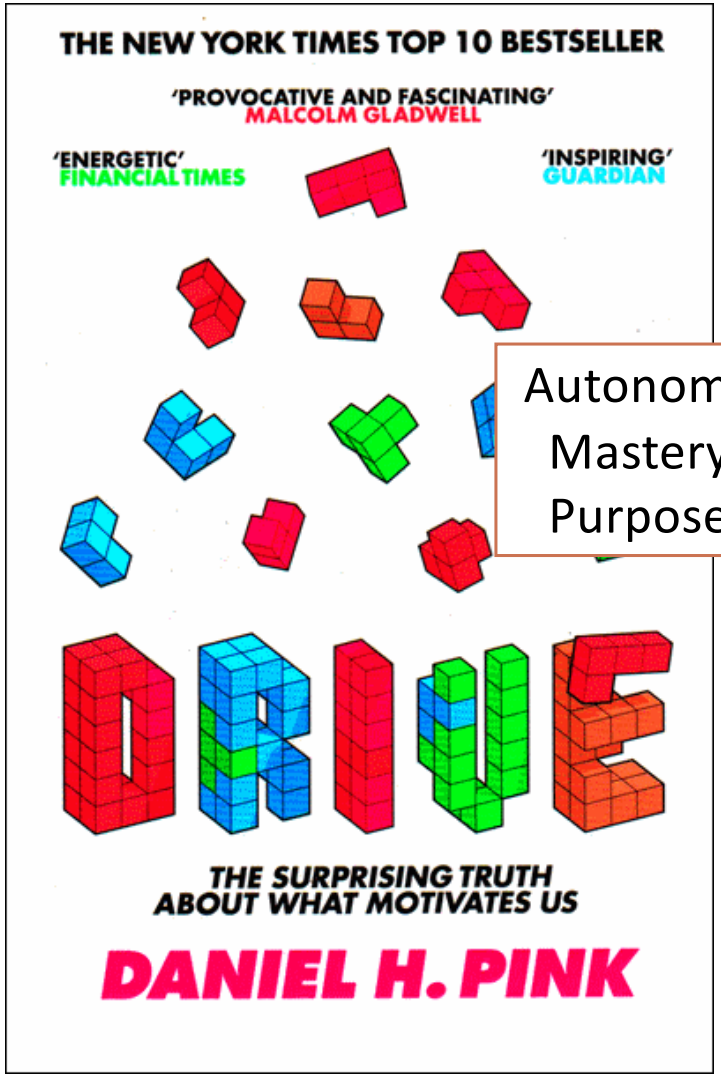

- What happens when developer bonuses are based on
  - Lines of code per day?
  - Amount of documentation written?
  - Low number of reported bugs in their code?
  - Low number of open bugs in their code?
  - High number of fixed bugs?
  - Accuracy of time estimates?



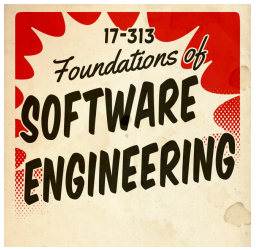
CRICOS PROVIDER #00120C




Can extinguish intrinsic motivation
Can diminish performance
Can crush creativity
Can crowd out good behavior
Can encourage cheating, shortcuts, and unethical behavior
Can become addictive
Can foster short-term thinking



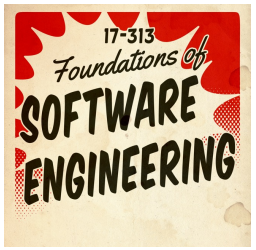

Autonomy
Mastery
Purpose






CRICOS PROVIDER #00120C

# Warning

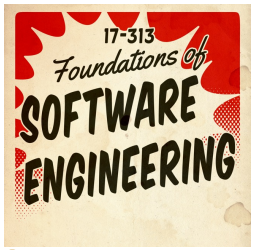



- Most software metrics are controversial
  - Usually only plausibility arguments, rarely rigorously validated
  - Cyclomatic complexity was repeatedly refuted and is still used
  - “Similar to the attempt of measuring the intelligence of a person in terms of the weight or circumference of the brain”
- Use carefully!
- Code size dominates many metrics
- Avoid claims about human factors (e.g., readability) and quality, unless validated
- Calibrate metrics in project history and other projects
- Metrics can be gamed; you get what you measure



CRICOS PROVIDER #00120C

# (Some) strategies




- Metrics tracked using tools and processes (process metrics like time, or code metrics like defects in a bug database).
- Expert assessment or human-subject experiments (controlled experiments, talk-aloud protocols).
- Mining software repositories, defect databases, especially for trend analysis or defect prediction.
  - Some success e.g., as reported by Microsoft Research
- Benchmarking (especially for performance).



CRICOS PROVIDER #00120C

# End of Monday Lecture/Start of Tuesday Lecture

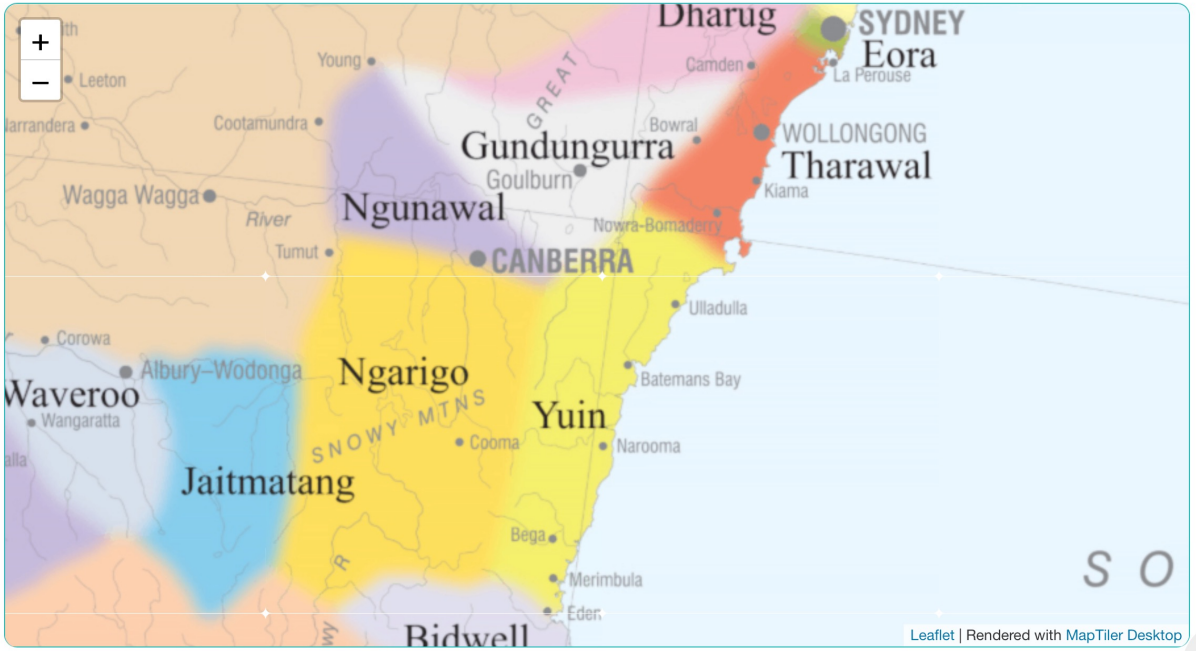


CRICOS PROVIDER #00120C

# ANU Acknowledgment of Country

AIATSIS Explore Family history Collection Research Education What's new About Shop Search

"We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present."

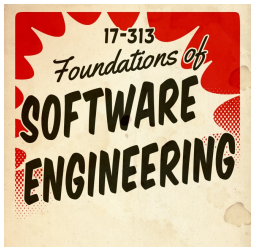



https://aiatsis.gov.au/explore/map-indigenous-australia



CRICOS PROVIDER #00120C

# Factors in a successful measurement program




- Set solid measurement objectives and plans.
- Make measurement part of the process.
- Gain a thorough understanding of measurement.
- Focus on cultural issues.
- Create a safe environment to collect and report true data.
- Cultivate a predisposition to change.
- Develop a complementary suite of measures.

Carol A. Dekkers and Patricia A. McQuaid, "The Dangers of Using Software Metrics to (Mis)Manage", 2002.



CRICOS PROVIDER #00120C

# Kaner's questions when choosing a metric

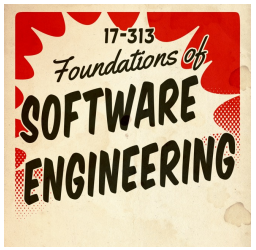

1. What is the purpose of this measure?
2. What is the scope of this measure?
3. What attribute are you trying to measure?
4. What is the attribute's natural scale?
5. What is the attribute's natural variability?
6. What instrument are you using to measure the attribute, and what reading do you take from the instrument?
7. What is the instrument's natural scale?
8. What is the reading's natural variability (normally called measurement error)?
9. What is the attribute's relationship to the instrument?
10. What are the natural and foreseeable side effects of using this instrument?

Cem Kaner and Walter P. Bond. "Software Engineering Metrics: What Do They Measure and How Do We Know?" 2004



CRICOS PROVIDER #00120C

# Further Reading on Metrics

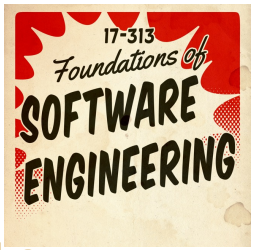



- Sommerville. Software Engineering. Edition 7/8, Sections 26.1, 27.5, and 28.3
- Hubbard. How to measure anything: Finding the value of intangibles in business. John Wiley & Sons, 2014. Chapter 3
- Kaner and Bond. Software Engineering Metrics: What Do They Measure and How Do We Know? METRICS 2004
- Fenton and Pfleeger. Software Metrics: A rigorous & practical approach. Thomson Publishing 1997



CRICOS PROVIDER #00120C

# Microsoft Survey (2014)



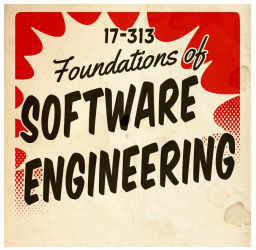

"Suppose you could work with a team of data scientists and data analysts who specialize in studying how software is developed. Please list up to five questions you would like them to answer. Why do you want to know? What would you do with the answers?"

Andrew Begel and Thomas Zimmermann. "Analyze this! 145 questions for data scientists in software engineering." *ICSE*. 2014.



CRICOS PROVIDER #00120C

# Top Questions

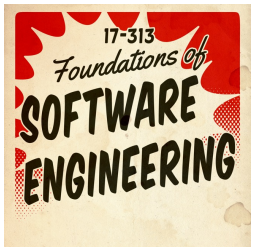



- How do users typically use my application?
- What parts of a software product are most used and/or loved by customers?
- How effective are the quality gates we run at checkin?
- How can we improve collaboration and sharing between teams?
- What are best key performance indicators (KPIs) for monitoring services?
- What is the impact of a code change or requirements change to the project and tests?



CRICOS PROVIDER #00120C

# Top Questions




- What is the impact of tools on productivity?
- How do I avoid reinventing the wheel by sharing and/or searching for code?
- What are the common patterns of execution in my application?
- How well does test coverage correspond to actual code usage by our customers?
- What kinds of mistakes do developers make in their software? Which ones are the most common?
- What are effective metrics for ship quality?



CRICOS PROVIDER #00120C

# Bottom Questions


- Which individual measures correlate with employee productivity (e.g., employee age, tenure, engineering skills, education, promotion velocity, IQ)?
- Which coding measures correlate with employee productivity (e.g., lines of code, time it take to build the software, a particular tool set, pair programming, number of hours of coding per day, language)?
- What metrics can be used to compare employees?
- How can we measure the productivity of a Microsoft employee?
- Is the number of bugs a good measure of developer effectiveness?
- Can I generate 100% test coverage?



CRICOS PROVIDER #00120C

# Context: big ole pile of code.


# ...do something to it.

Like: Fix a bug, implement a feature, write a test...



CRICOS PROVIDER #00120C

# You cannot understand the entire system.



CRICOS PROVIDER #00120C

# Goal


- *To develop and test a working model or set of working hypotheses about how (some part of) a system works.*
- Working model: an understanding of the pieces of the system (components), and the way they interact (connections).
- It is common in practice to consult documentation, experts.
- Prior knowledge/experience is also useful (see: frameworks, architectural patterns, design patterns).
- Today, we focus on individual information gathering via observation, probes, and hypothesis testing.



CRICOS PROVIDER #00120C

# TWO PROPERTIES OF SOFTWARE THAT ARE USUALLY ANNOYING THAT WE CAN TAKE ADVANTAGE OF



CRICOS PROVIDER #00120C

# Software constantly changes
# → Software is easy to change!

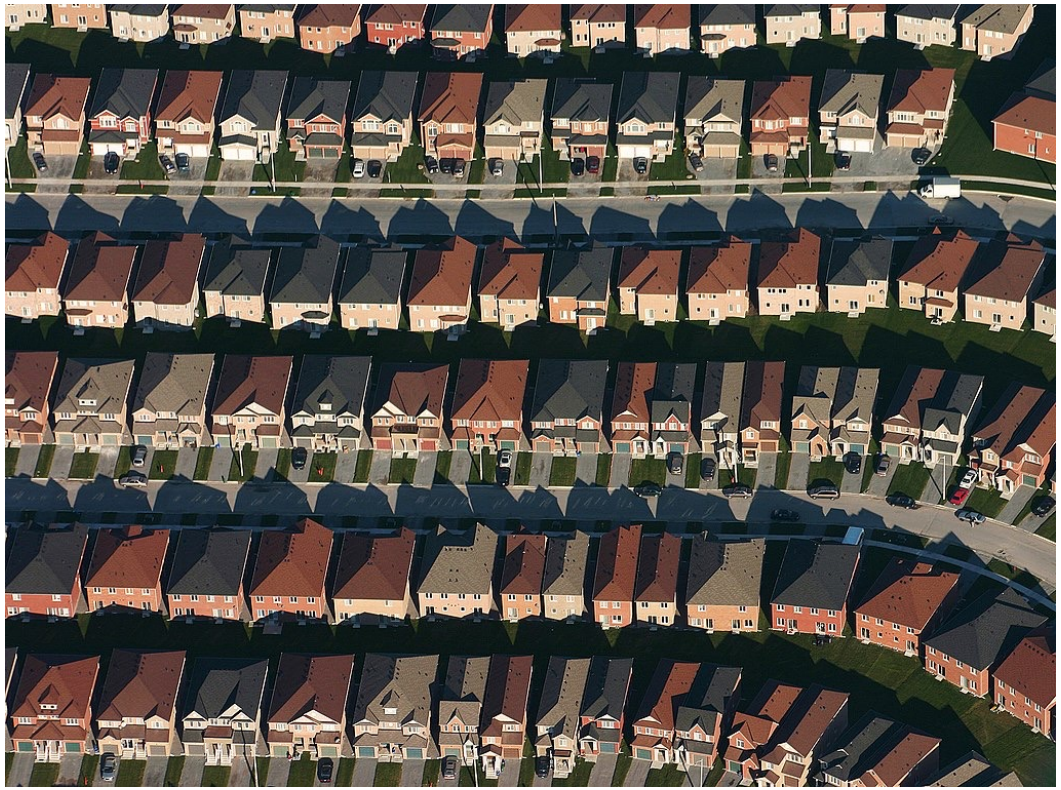

Guess so!

Is this wall load-bearing?



CRICOS PROVIDER #00120C

# Software is a big redundant mess
# → there's always something to copy as a starting point!



CRICOS PROVIDER #00120C

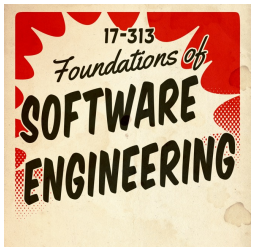
Key insight in grokking unfamiliar code/apps

# CODE MUST RUN TO DO STUFF!!



CRICOS PROVIDER #00120C

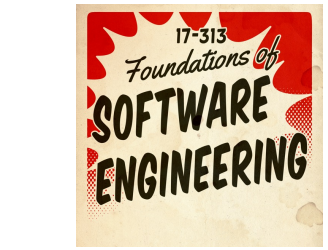
# 1. If code must run, it must have a beginning

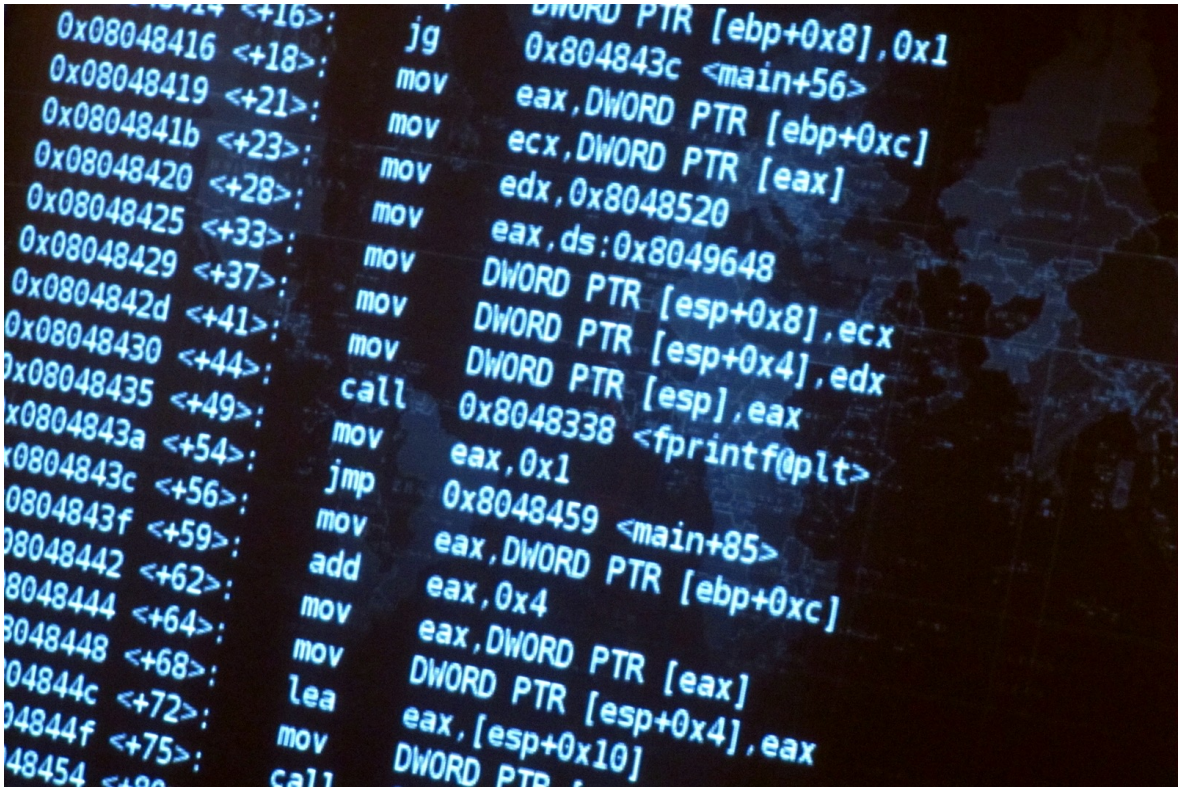





CRICOS PROVIDER #00120C

# 2. If code must run, it must exist

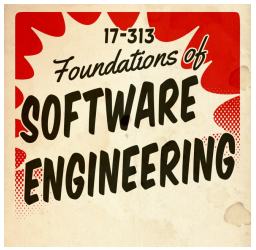





CRICOS PROVIDER #00120C

# The Beginning: Entry Points



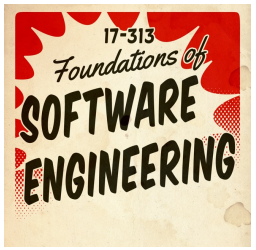

Some trigger that causes code to run.

- **Locally installed programs**: run cmd, OS launch, I/O events, etc.
- **Local applications in dev**: build + run, test, deploy (e.g. docker)
- **Web apps server-side**: Browser sends HTTP request (GET/POST)
- **Web apps client-side**: Browser runs JavaScript



CRICOS PROVIDER #00120C

# Code must exist. But where?



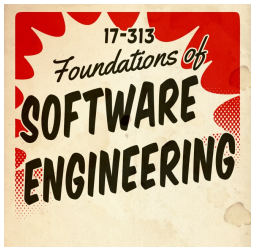

Helps to identify what's knowable and what's changeable

- **Locally installed programs**: run cmd, OS launch, I/O events, etc.
  - Binaries (machine code) on your computer
- **Local applications in dev**: build + run, test, deploy (e.g. docker)
  - Source code in repository (+ dependencies)
- **Web apps server-side**: Browser sends HTTP request (GET/POST)
  - Code runs remotely (you can only observe outputs)
- **Web apps client-side**: Browser runs JavaScript
  - Source code is downloaded and run locally (see: browser dev tools!)



CRICOS PROVIDER #00120C

# Side note on build systems

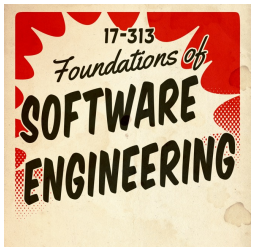



- Basically the same across languages / platforms
  - Make, maven, gradle, grunt, bazel, etc.
- **Goal**: Source code + dependencies + config → runnables
- Common themes:
  - Dependency management (repositories, versions, etc)
  - Config management (platform-specific features, file/dir names, IP addresses, port numbers, etc)
  - Runnables (start, stop?, test)
  - Almost always have ‘debug’ mode and help (‘-h’ or similar)
  - Almost always have one or more “build” directories (= not part of source repo)



CRICOS PROVIDER #00120C

# Can running code be **P**robed/**U**nderstood/**E**dited?




Transparent

Source code built locally

(P+U+E)

Translucent

Binaries running locally

| Open source | Closed source |
|---|---|
| (P+U) | (P) |

Opaque

Server-side apps running remotely

| Open source | Closed source |
|---|---|
| (U) | - |



CRICOS PROVIDER #00120C

NYTimes quiz: http://bit.ly/problemQuiz

# BUT FIRST! AN EXERCISE.



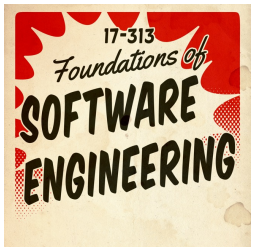
CRICOS PROVIDER #00120C

# Beware of cognitive biases.



CRICOS PROVIDER #00120C

# Beware of cognitive biases




- **anchoring**
- **confirmation bias**
- **congruence bias**: The tendency to test hypotheses exclusively through direct testing, instead of testing possible alternative hypotheses
- conservatism (belief revision)
- curse of knowledge
- default effect
- expectation bias
- overconfidence effect
- plan continuation bias
- pro innovation bias
- recency illusion

https://en.wikipedia.org/wiki/List_of_cognitive_biases



CRICOS PROVIDER #00120C

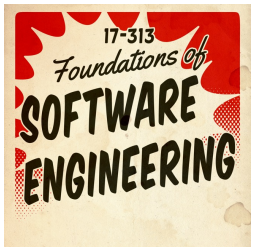
Source code built locally

# CREATING A WORKING MODEL OF UNFAMILIAR CODE



CRICOS PROVIDER #00120C

# Static (+dynamic) information gathering




- Basic needs:
  - **Code/file search and navigation**
  - Code editing (probes)
  - Execution of code, tests
  - Observation of output (observation)

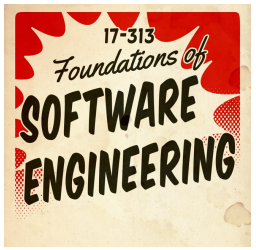
At the command line: **grep** and **find**!
(Do a web search for tutorials)

- Many choices here on tools! Depends on circumstance.
  - grep/find/etc.  Having a command on Unix tools is invaluable
  - A decent IDE
  - Debugger
  - Test frameworks + coverage reports
  - Google (or your favorite web search engine)

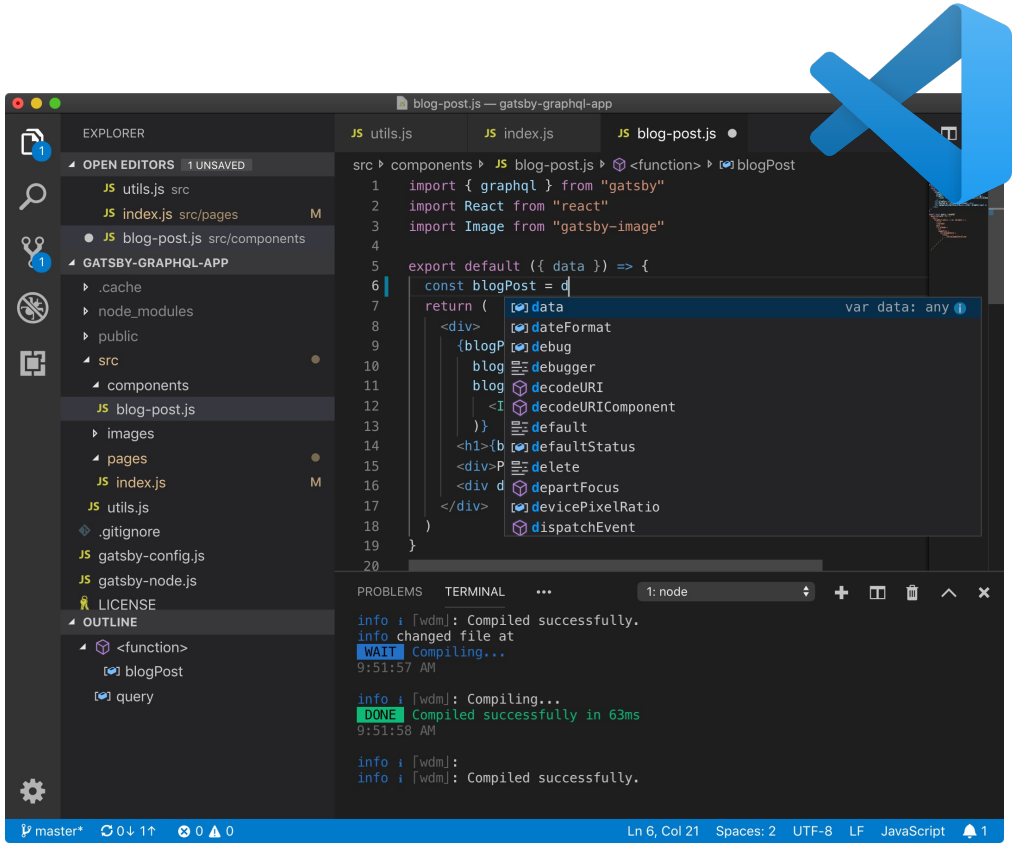


CRICOS PROVIDER #00120C

# Static Information Gathering




- Please configure and use a *legitimate* IDE.
  - No favorites? We recommend VSCode and IntelliJ IDEA.
- Why?
  - “search all files”
  - “jump to definition”
  - “download dependency source”
- Remember: real software is too complicated to keep in your head.

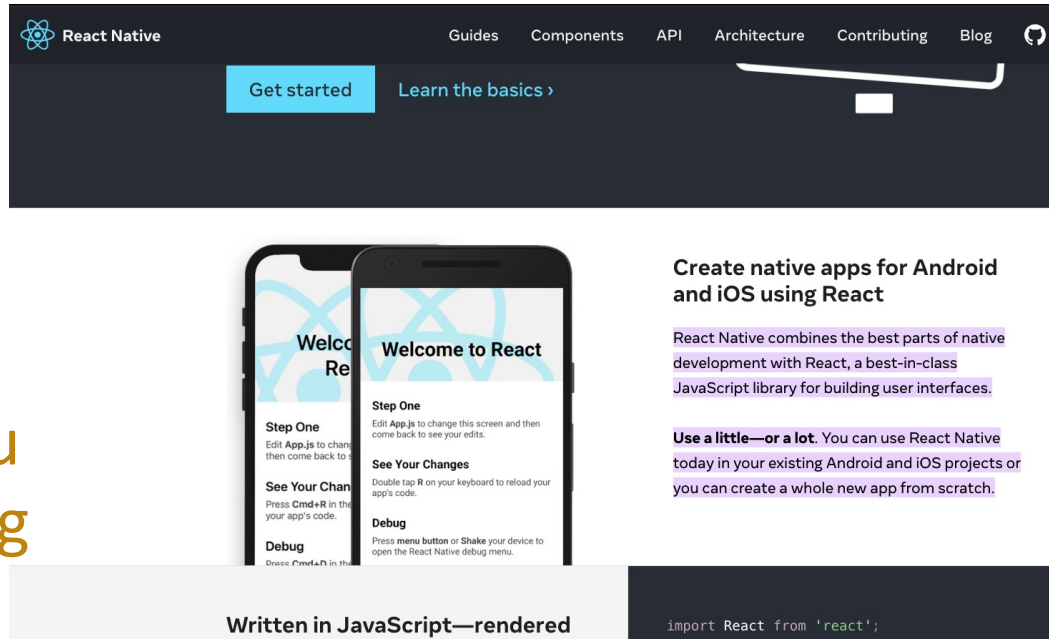





CRICOS PROVIDER #00120C

# Consider documentation/tutorials judiciously

- Great for discovering entry points!
- Can teach you about general structure, architecture.
  - Forward-reference to architectural patterns!
- As you gain experience, you will recognize more of these, and you will immediately know something about how the program works.
- For example, next time you work on a mobile app…





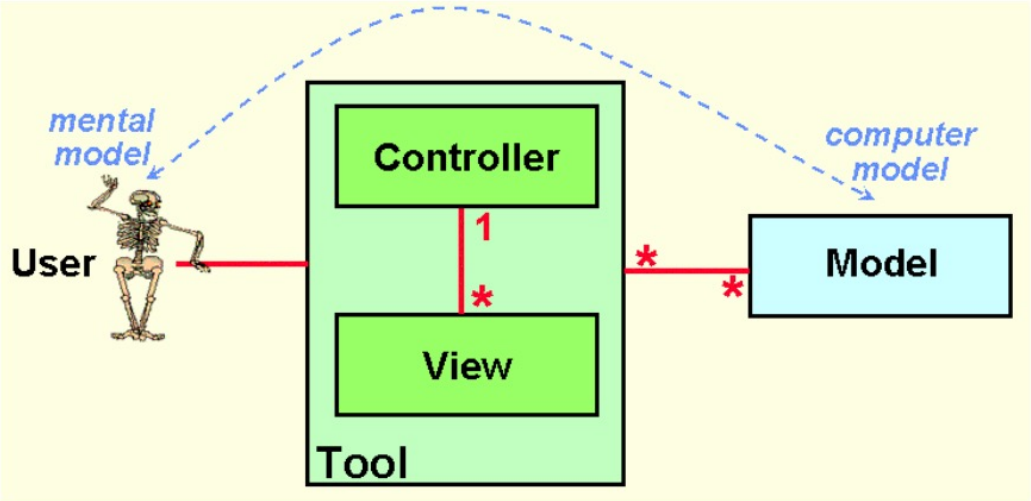

CRICOS PROVIDER #00120C

# Consider documentation/tutorials judiciously

**A bit of Model View Controller history**

Trygve Reenskaug discovered MVC at Xerox PARC in 1978.

> *The essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer [Trygve Reenskaug].*




http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

https://medium.com/swlh/elements-of-mvc-in-react-9382de427c09

The term *mental-model* stuck wit 211 represents the essence of our



CRICOS PROVIDER #00120C

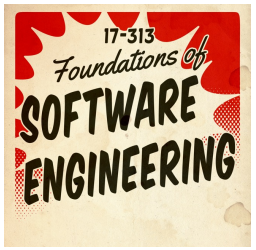
# Dynamic Information Gathering


- Key principle 1: change is a useful primitive to inform mental models about a software system.
- Key principle 2: systems almost always provide some kind of starting point.
- Put simply:
    1. Build it.
    2. Run it.
    3. Change it.
    4. Run it again.
- Can provide information both *bottom up* or *top down*, depending on the situation.



CRICOS PROVIDER #00120C

# Probes - Observe, control or “lightly” manipulate execution




- Printf(“here”)
- Turning on automatic debug info logging
- Breakpoints
- Sophisiticated debugging tools
  - Breakpoint, eval, step through / step over
  - (Some tools even support remote debugging)
- Delete debugging (equivalent of `kill -9`)



CRICOS PROVIDER #00120C

# Step 0: sanity check basic model + hypotheses.


- Confirm that you can build and run the code.
  - Ideally *both* using the tests provided, *and* by hand.
- Confirm that the code you are running *is the code you built.*
- Confirm that you can make *an externally visible change*.
- How? Where? Starting points:
  - Run an existing test, change it.
  - Write a new test.
  - Change the code, write or rerun a test that should notice the change.
- Make sure the changes persist if you want them to.
  - Distinguish between source repository and build/deploy directories.



CRICOS PROVIDER #00120C



CRICOS PROVIDER #00120C

# Notes on Measuring Engineering Productivity

- Collecting and analysing data on the *human side of things*
- As organisations grow in size *linearly*, communication costs grow *quadratically* (see The Mythical Man-Month or even Amdahl's Law in Computer Architecture ☺)
- Could try to make each individual more productive?
- How to *measure* individual productivity and identify inefficiencies without taking up too many resources?
- Google has a team of researchers dedicated to *engineering productivity*



CRICOS PROVIDER #00120C

# Notes on Measuring Engineering Productivity

- Building on *social sciences*, allows to study human side like personal motivations, incentives, and strategies for complex tasks
- What *should* we measure?
- *How* to use metrics to track improvements and productivity?
- Case Study around the process of C++ and Java language teams around *Code Readability*
- *Is the time spent on the readability process worthwhile?*



CRICOS PROVIDER #00120C

# Notes on Measuring Engineering Productivity

- Is It Even Worth Measuring?
- Triage Questions:
  1. What result are you expecting, and why?
  2. If the data supports your expected result, what action will be taken?
  3. If we get a negative result, will appropriate action be taken?
  4. Who is going to decide to take action on the result, and when would they do it?
- Reasons NOT to measure can be:
  - You can't afford to change the process/tools right now
  - Any results will soon be invalidated by other factors
  - The results will be used only as vanity metrics to support something you were going to do anyway
  - The only metrics available are not precise enough to measure the problem and can be confounded by other factors



CRICOS PROVIDER #00120C

# Notes on Measuring Engineering Productivity

- At Google they use Goals/Signals/Metrics (GSM) framework to guide metrics creation:
  - A *goal* is a desired end result. It's phrased in terms of what you want to understand at a high level and should not contain references to specific ways to measure it.
  - A signal is how you might know that you've achieved the end result. Signals are things we would *like* to measure, but they might not be measurable themselves.
  - A *metric* is a proxy for a signal. It is the thing we actually can measure. It might not be the ideal measurement, but it is something that we believe is close enough.
- GSM encourages us to select metrics based on their ability to measure the original goals



CRICOS PROVIDER #00120C

# Goals (Capturing Productivity Trade Offs)

***Quality of the code***

What is the quality of the code produced? Are the test cases good enough to prevent regressions? How good is an architecture at mitigating risk and changes?

***Attention from engineers***

How frequently do engineers reach a state of flow? How much are they distracted by notifications? Does a tool encourage engineers to context switch?

***Intellectual complexity***

How much cognitive load is required to complete a task? What is the inherent complexity of the problem being solved? Do engineers need to deal with unnecessary complexity?

***Tempo and velocity***

How quickly can engineers accomplish their tasks? How fast can they push their releases out? How many tasks do they complete in a given timeframe?

***Satisfaction***

How happy are engineers with their tools? How well does a tool meet engineers' needs? How satisfied are they with their work and their end product? Are engineers feeling burned out?



CRICOS PROVIDER #00120C

# Goals (Readability Case Study)

*Quality of the code*

Engineers write higher-quality code as a result of the readability process; they write more consistent code as a result of the readability process; and they contribute to a culture of code health as a result of the readability process.

*Attention from engineers*

We did not have any attention goal for readability. This is OK! Not all questions about engineering productivity involve trade-offs in all five areas.

*Intellectual complexity*

Engineers learn about the Google codebase and best coding practices as a result of the readability process, and they receive mentoring during the readability process.

*Tempo and velocity*

Engineers complete work tasks faster and more efficiently as a result of the readability process.

*Satisfaction*

Engineers see the benefit of the readability process and have positive feelings about participating in it.



CRICOS PROVIDER #00120C

# Signals (Readability Case Study)

*Table 7-1. Signals and goals*

| Goals | Signals |
|---|---|
| Engineers write higher-quality code as a result of the readability process. | Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability. The readability process has a positive impact on code quality. |
| Engineers learn about the Google codebase and best coding practices as a result of the readability process. | Engineers report learning from the readability process. |
| Engineers receive mentoring during the readability process. | Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process. |
| Engineers complete work tasks faster and more efficiently as a result of the readability process. | Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability. Changes written by engineers who have been granted readability are faster to review than changes written by engineers who have not been granted readability. |
| Engineers see the benefit of the readability process and have positive feelings about participating in it. | Engineers view the readability process as being worthwhile. |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| QUANTS | Goal | Signal | Metric |
|---|---|---|---|
| Quality of the code | Engineers write higher-quality code as a result of the readability process. | Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability. | Quarterly Survey: Proportion of engineers who report being satisfied with the quality of their own code |
| | | The readability process has a positive impact on code quality. | Readability Survey: Proportion of engineers reporting that readability reviews have no impact or negative impact on code quality |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| QUANTS | Goal | Signal | Metric |
|---|---|---|---|
| | | | Readability Survey: Proportion of engineers reporting that participating in the readability process has improved code quality for their team |
| | Engineers write more consistent code as a result of the readability process. | Engineers are given consistent feedback and direction in code reviews by readability reviewers as a part of the readability process. | Readability Survey: Proportion of engineers reporting inconsistency in readability reviewers' comments and readability criteria. |
| | Engineers contribute to a culture of code health as a result of the readability process. | Engineers who have been granted readability regularly comment on style and/or readability issues in code reviews. | Readability Survey: Proportion of engineers reporting that they regularly comment on style and/or readability issues in code reviews |
| Attention from engineers | n/a | n/a | n/a |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| | | | |
|---|---|---|---|
| Intellectual | Engineers learn about the Google codebase and best coding practices as a result of the readability process. | Engineers report learning from the readability process. | Readability Survey: Proportion of engineers reporting that they learned about four relevant topics |
| | | | Readability Survey: Proportion of engineers reporting that learning or gaining expertise was a strength of the readability process |
| | Engineers receive mentoring during the readability process. | Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process. | Readability Survey: Proportion of engineers reporting that working with readability reviewers was a strength of the readability process |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| | | | |
|---|---|---|---|
| Tempo/velocity | Engineers are more productive as a result of the readability process. | Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability. | Quarterly Survey: Proportion of engineers reporting that they're highly productive |
| | | Engineers report that completing the readability process positively affects their engineering velocity. | Readability Survey: Proportion of engineers reporting that *not* having readability reduces team engineering velocity |
| | | Changelists (CLs) written by engineers who have been granted readability are faster to review than CLs written by engineers who have not been granted readability. | Logs data: Median review time for CLs from authors with readability and without readability |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| QUANTS | Goal | Signal | Metric |
|---|---|---|---|
| | | CLs written by engineers who have been granted readability are easier to shepherd through code review than CLs written by engineers who have not been granted readability. | Logs data: Median shepherding time for CLs from authors with readability and without readability |
| | | CLs written by engineers who have been granted readability are faster to get through code review than CLs written by engineers who have not been granted readability. | Logs data: Median time to submit for CLs from authors with readability and without readability |
| | | The readability process does not have a negative impact on engineering velocity. | Readability Survey: Proportion of engineers reporting that the readability process negatively impacts their velocity |
| | | | Readability Survey: Proportion of engineers reporting that readability reviewers responded promptly |
| | | | Readability Survey: Proportion of engineers reporting that timeliness of reviews was a strength of the readability process |



CRICOS PROVIDER #00120C

# Metrics (Readability Case Study)

| | | | |
|---|---|---|---|
| Satisfaction | Engineers see the benefit of the readability process and have positive feelings about participating in it. | Engineers view the readability process as being an overall positive experience. | Readability Survey: Proportion of engineers reporting that their experience with the readability process was positive overall |
| | | Engineers view the readability process as being worthwhile | Readability Survey: Proportion of engineers reporting that the readability process is worthwhile |
| | | | Readability Survey: Proportion of engineers reporting that the quality of readability reviews is a strength of the process |
| | | | Readability Survey: Proportion of engineers reporting that thoroughness is a strength of the process |
| | | Engineers do not view the readability process as frustrating. | Readability Survey: Proportion of engineers reporting that the readability process is uncertain, unclear, slow, or frustrating |

Quarterly Survey: Proportion of engineers reporting that they're satisfied with their own engineering velocity



CRICOS PROVIDER #00120C

# Case Study on Readability Outcome

- Study showed that it was overall worthwhile:
  - Engineers who had achieved readability were satisfied with the process and felt they learned from it
  - Logs showed that they also had their code reviewed faster and submitted it faster, even accounting for no longer needing as many reviewers
  - Study also showed places for improvement with the process: engineers identified pain points
- The language teams improved the tooling and process based on the results



CRICOS PROVIDER #00120C

# Key Points

- Measurement is difficult but important for decision making
- Software metrics are easy to measure but hard to interpret, validity often not established
- Many metrics exist, often composed; pick or design suitable metrics if needed
- Careful in use: monitoring vs incentives
- Strategies beyond metrics



CRICOS PROVIDER #00120C

# Key Points

- Use measurements as a decision tool to reduce uncertainty
- Understand difficulty of measurement; discuss validity of measurements
- Provide examples of metrics for software qualities and process
- Understand limitations and dangers of decisions and incentives based on measurements



CRICOS PROVIDER #00120C

# Key Points

- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability etc. Software standards are important for quality assurance as they represent an identification of ‘best practice’. When developing software, standards provide a solid foundation for building good quality software.
- Reviews of the software process deliverables involve a team of people who check that quality standards are being followed. Reviews are the most widely used technique for assessing quality.



CRICOS PROVIDER #00120C

# Key Points

- In a program inspection or peer review, a small team systematically checks the code. They read the code in detail and look for possible errors and omissions. The problems detected are discussed at a code review meeting.
- Agile quality management relies on establishing a quality culture where the development team works together to improve software quality.
- Software measurement can be used to gather quantitative data about software and the software process.



CRICOS PROVIDER #00120C

# Key Points

- You may be able to use the values of the software metrics that are collected to make inferences about product and process quality.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems. These components should then be analyzed in more detail.
- Software analytics is the automated analysis of large volumes of software product and process data to discover relationships that may provide insights for project managers and developers.



CRICOS PROVIDER #00120C

# Key Points

- Understand and scope the task of taking on and understanding a new and complex piece of existing software.
- Appreciate the importance of configuring an effective IDE.
- Contrast different types of code execution environments including local, remote, application, and libraries.
- Enumerate both static and dynamic strategies for understanding and modifying a new codebase.



CRICOS PROVIDER #00120C

# Key Points

- Before measuring productivity, ask whether the result is actionable, regardless of whether the results is positive or negative
- Select meaningful metrics using the GSM framework
- Select metrics that cover all parts of productivity (QUANTS)
- Qualitative metrics are metrics too!
- Aim to create recommendations that are built into the developer workflow and incentives



CRICOS PROVIDER #00120C