



# Software Metrics

Alex Boughton

# Executive Summary

- What are software metrics?
- Why are software metrics used in industry, and how?
- Limitations on applying software metrics
  - A framework to help refine and understand which metrics to use, based on IEEE software metrics standards.
- Types of software metrics
  - When to use them
  - Limitations to consider when using metrics to evaluate software systems
  - In depth look at DSQI and Software package metrics
    - Example applying BSQI to a real project
- An in depth look at research that has been done on software metrics
  - Fault prediction models
  - Human judgment and software metrics
  - Software metrics for different types of defects
  - Churn and dependency for fault prediction
  - How do we know what the metrics are measuring

# What are software metrics?

- Tools for anyone involved in software engineering to understand varying aspects of the code base, and the project progress.
- They are different from just testing for errors because they can provide a wider variety of information about the following aspects of software systems:
  - Quality of the software, different metrics look at different aspects of quality, but this aspect deals with the code.
  - Schedule of the software project on the whole. I.e. some metrics look at functionality and some look at documents produced.
  - Cost of the software project. Includes maintenance, research and typical costs associated with a project.
  - Size/Complexity of the software system. This can be either based on the code or at the macro-level of the project and it's dependency on other projects.

# Motivations for using metrics in software engineering

- In regards to software project cost and underestimation, it is a problem that has not diminished in the last 70 years.[2]
- The Standish Chaos Report (2004) found only 29% of project met their criteria for project success: projects that were on budget, on schedule, and with the expected functionality. [2]
- The Standish Chaos Report also estimated that the annual cost of cancelled projects was \$55 billion. [2]
- Help predict defects in code and can be used to determine code quality

# General Uses of Metrics

- Software metrics are used to obtain objective reproducible measurements that can be useful for quality assurance, performance, debugging, management, and estimating costs.
- Finding defects in code (post release and prior to release), predicting defective code, predicting project success, and predicting project risk
- There is still some debate around which metrics matter and what they mean, the utility of metrics is limited to quantifying one of the following goals: Schedule of a software project, Size/complexity of development involved, cost of project, quality of software

# Limitations to the general use of software metrics in project planning and assessment

- There is “a strong tendency for professionals to display over-optimism and over-confidence” [2]
- Arguments that simplistic measurements may cause more harm than good, ie data that is shown because its easy to gather and display [5]
- There are arguments about the effects that software metrics have on the developers’ productivity and well being

# Kaner and Bond's Evaluation Framework for metrics

- In a paper that is reviewed later in the presentation, the researchers develop a general framework for evaluating a metric. They lay out 10 points to check for, based on the IEEE software metric criteria.
- The framework:
  1. What is the purpose of the measure?
    - Helps you understand the stakes in making the measurement so that you know how much to invest in ensuring it's validity.
  2. What is the scope of the measure?
    - Does it impact one team or developer? Or multiple parts of a large project. The authors point out that as the scope broadens so does the number of confounding variables to the system.
  3. What attribute of the software is being measured
  4. What is the natural scale of the attribute we are trying to measure?

# Kaner and Bond's Evaluation Framework for metrics

5. What is the natural variability of the attribute?
  - ▣ What numbers do you expect to see?
6. What measuring instrument do we use to perform the measurement?
7. What is the natural scale for this metric?
8. What is the natural variability of readings from this instrument?
  - ▣ Are small differences to be expected? Or does even the slightest change mean that a huge improvement has been reached.
9. What is the relationship of the attribute to the metric value?
  - ▣ Make sure you are measuring the right thing.
10. What are the natural and foreseeable side effects of using this instrument?
  - ▣ If a metric is used will the effects of writing code that makes the metric better be what we want them to be?
  - ▣ The authors point out too that an improvement in the metric should mean an equal increase in code quality, or else there may be something the metric isn't showing you



# Types of metrics

- Requirements metrics
  - Size of requirements
  - Traceability
  - Completeness
  - Volatility
  
- Product Metrics
  - Code metrics
    - Lines of code LOC
    - Design metrics – computed from requirements or design documents before the system has been implemented
    - Object oriented metrics- help identify faults, and allow developers to see directly how to make their classes and objects more simple.
  - Test metrics
  - Communication metrics – looking at artifacts ie email, and meetings.

# Types of metrics (cont.)

- Process metrics
  - Measure the process of software development
    - Commonly used by management to check the budget and office procedures for efficiency
    - Evaluate and track aspects of the software design process like:
      - Human resources
      - Time
      - Schedule
      - Methodology

Materials for this slide and the previous slide is adapted from [6]

# A few specific metrics in depth

- Design Structure Quality Index (DSQI)
- Robert Cecil Martin's "Software Package Metrics"

# Design Structure Quality Index

- Used for evaluating object oriented software packages
- Meant to be used with in an extreme programming framework
- Advantage is the that the metric calculation is relatively transparent, so as long as the criteria that Robert Cecil Martin decided were important to have good software, then developers can build software that follows these constraints and get better metrics on their code.
- The aspects of the software system used in calculating the metric are shown in the following screen shot of the website.  
[10]

# Design Structure Quality Index








## Design Structure Quality Index (DSQI) Calculator


[About DSQI](#) 

[Online-Help](#) 

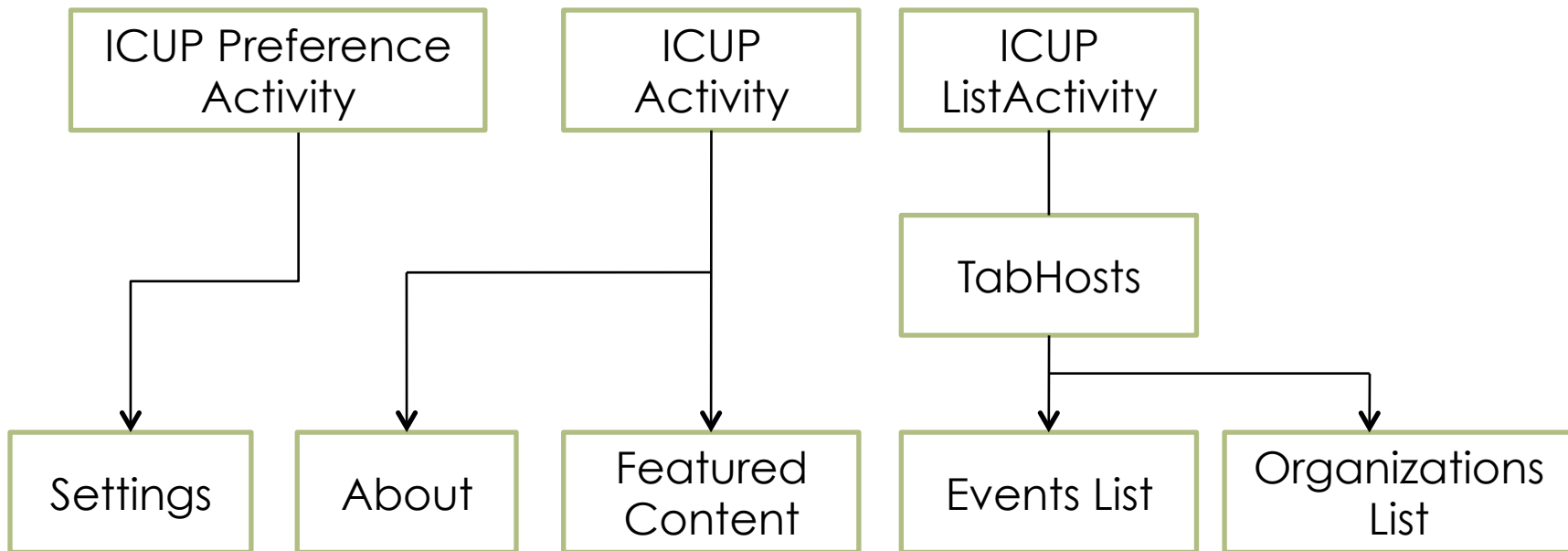
[Download](#) 

Click on the help icons  for further information.

S1	<input type="text"/>	- The total number of modules defined in the program architecture.	
S2	<input type="text"/>	- The number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere.	
S3	<input type="text"/>	- The number of modules whose correct function depends on prior processing.	
S4	<input type="text"/>	- The number of database items.	
S5	<input type="text"/>	- The total number of unique database items.	
S6	<input type="text"/>	- The number of database segments.	
S7	<input type="text"/>	- The number of modules with a single entry and exit point.	

Was a distinct method used to develop the design?  

# Overview of Example System Structure for DSQI Example



# DSQI Calculation








## Design Structure Quality Index (DSQI) Calculator


[About DSQI](#) 

[Online-Help](#) 

[Download](#) 

Click on the help icons  for further information.

<b>S1</b>	<input type="text" value="24"/>	- The total number of modules defined in the program architecture.	
<b>S2</b>	<input type="text" value="4"/>	- The number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere.	
<b>S3</b>	<input type="text" value="4"/>	- The number of modules whose correct function depends on prior processing.	
<b>S4</b>	<input type="text" value="22"/>	- The number of database items.	
<b>S5</b>	<input type="text" value="5"/>	- The total number of unique database items.	
<b>S6</b>	<input type="text" value="80"/>	- The number of database segments.	
<b>S7</b>	<input type="text" value="0"/>	- The number of modules with a single entry and exit point.	

Was a distinct method used to develop the design?  

- S1 - Since this project was like developed using object oriented design the number of concrete classes can be used.
- S2 – Some part of the modules' functionality depends on another source, or they are producing data for another module
  - There are four classes that receive data from the modules that receive information from a web connection
- S3 – number of modules that require data to have been previously processed by another module
  - We have 4 classes that populate tabs based on information from the previously mentioned 4 modules.
- S4 – All of the database items, including data objects and attributes that define objects.

# DSQI Calculation








## Design Structure Quality Index (DSQI) Calculator


[About DSQI](#) 

[Online-Help](#) 

[Download](#) 

Click on the help icons  for further information.

<b>S1</b>	<input type="text" value="24"/>	- The total number of modules defined in the program architecture.	
<b>S2</b>	<input type="text" value="4"/>	- The number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere.	
<b>S3</b>	<input type="text" value="4"/>	- The number of modules whose correct function depends on prior processing.	
<b>S4</b>	<input type="text" value="22"/>	- The number of database items.	
<b>S5</b>	<input type="text" value="5"/>	- The total number of unique database items.	
<b>S6</b>	<input type="text" value="80"/>	- The number of database segments.	
<b>S7</b>	<input type="text" value="0"/>	- The number of modules with a single entry and exit point.	

Was a distinct method used to develop the design?  

- S5 - all database items that are not duplicated or similar to other items.
- S6 – number of database segments including different records or individual objects
- S7 – Modules that don't have multiple start and end points
- Since we used object oriented design in the creation of our app we checked this distinct method box.




# DSQI Calculation

## Design Structure Quality Index (DSQI) Calculator







[About DSQI](#) 

[On-Line Help](#) 

[Download](#) 

Enter percentages below to give weighting factors to each D-Value. 

If you want all intermediate values weighted equally check here.

<b>D1</b>	<input type="text" value="1"/>	- Program Structure.		Weight	<input type="text" value="16.7"/>	%
<b>D2</b>	<input type="text" value=".83333"/>	- Module Independence.		Weight	<input type="text" value="16.7"/>	%
<b>D3</b>	<input type="text" value=".83333"/>	- Modules not dependent on prior processing.		Weight	<input type="text" value="16.7"/>	%
<b>D4</b>	<input type="text" value=".77273"/>	- Database size.		Weight	<input type="text" value="16.7"/>	%
<b>D5</b>	<input type="text" value="-2.63636"/>	- Database compartmentalization.		Weight	<input type="text" value="16.7"/>	%
<b>D6</b>	<input type="text" value="1.00000"/>	- Module entrance and exit characteristic.		Weight	<input type="text" value="16.7"/>	%
<input type="button" value="← Back"/>					Total :	<input type="text" value="100.00"/> %

DSQI:

- This screen shows you the scores that the software system currently has for each individual design structure.
- The overall systems' SDQI metric is shown as .301 based on the priorities that we gave to each of the design structure ratings.
- The advantages of this metric are that it is easy to calculate and can help developers see directly where the design of the system may cause faults in the future.
- The number by itself is not very useful, but when used to compare a system to a past successful system it can give insight into project progress.

# Robert Cecil Martin popularized “Software Package Metrics”

- Robert Cecil Martin is one of the creators of agile software development methodologies and extreme programming.
- Used for evaluating object oriented software packages
- Meant to be used with in an extreme programming framework
- Advantage is the that the metric calculation is relatively transparent.
- As long as the criteria are important, developers can build software that follows these constraints and get better metrics about their code.

# Robert Cecil Martin popularized “Software Package Metrics”

- Various aspects of software packages can be measured such as [4]
  - Number of classes and interfaces
  - Afferent Couplings: Number of packages that depend on the package being evaluated
  - Efferent Couplings: Number of outside packages the package under evaluation depends on
  - Abstractness: number of classes in the package that are abstract, ranges from 0-1 and is a percentage of abstract classes to concrete classes

# Robert Cecil Martin popularized “Software Package Metrics” (cont)

- Instability:  $I = \text{efferent couplings} / \text{total couplings}$ , ranges from 0-1. 0 means the package is completely stable to change, and 1 means the package is completely unstable to change.
- Distance from main sequence: this metric shows how the package balances between abstractness and stability. A package will do well in this metric by being either mostly abstract or stable, or completely concrete and instable.
- Package dependency cycles: packages in the packages hierarchy that depend on each other. (Dependency cycles)

# Current Research in Software Metrics

- Fault prediction models
  - Allow organizations to predict defects in code before software has been released.
  - There is debate around aggregating the models or modeling only some of the defects more accurately.
  - Part of the debate stems from the need for more research in software decomposition and how to decompose software for better quality systems.
- Models are used to explain aspects of a software system numerically in at a statistically significant level.

# Current Research in Software Metrics (cont.)

- Human judgment and software metrics
  - Research that is trying to address the human judgment side of metrics processing.
  - Working to address the expensive problem of failing projects that I mentioned earlier, costs \$55 billion annually.
- Software metrics for different types of software defects
  - Breaking down the defects that software is measured for will give a better view of the particular type of defect you are interested in.
- Frameworks for understanding metrics and making sure that we are using them correctly
  - One framework was shown earlier, more information from those researchers is to follow

# By No Means: A Study on Aggregating Software Metrics

- SLOC is currently used by many software engineering groups based on the intuitive belief that those large systems have more faults in them than small systems [1]
- Distribution in many software metrics is typically skewed.
- The cause for this can be speculated to be many things, but [1] argues that aggregation of metrics from the micro level of classes, methods, and packages to an entire system would be helpful as the outliers would get pulled into the larger amounts of data.
  - The researchers found that the correlation of this is not strong and that the results vary based on the aggregation technique that is used.

# By No Means: A Study on Aggregating Software Metrics

- This means that metrics analysis techniques, and the usefulness of data is not fool proof.
- Developers/managers that have an understanding of the parts of a system are still needed to make sure that the metrics are not misused or misunderstood. Metrics are a powerful tool that needs to be used with care.
- As mentioned in limitations, paying too much attention to one metric or another can lead to incorrect views of software systems and from that incorrect behavior based on those views. [5][7]



# Human Judgement and Software Metrics: Vision for the Future

- Paper works to address the problem “the low impact that software metrics has on current software development”
- Failing projects are an expensive problem that costs around 55\$ billion dollars each year.
- One of the reasons that cognitive psychologists believe this is happening is from various problems with management, and developer error.
- Within the problems with management is how they interpret the metrics, which they base their decision on.

# Human Judgement and Software Metrics: Vision for the Future (cont)

- Software metrics are statistical predictions and estimations, and not just a number. The numbers have three dimensions [2] error, bias, and variance or scatter. A human typically ignores these dimensions for simplicity and with the loss of information comes over optimism and over-confidence.
- Research is being done to use metacognition experiment results in application to software metrics interpretation and use in project planning and reflection. [2]
- Researchers Carolyn Mair and Martin Shepperd want to include the perspective on how people actually employ software metrics results currently so they can understand how to make those metrics better.

# Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories

- Defect category sensitive prediction models are more successful than general models [3]
  - Each category has different aspects to them, breaking them off by defect type makes them more accurate.
  - Tosun Misirli et al propose constructing models using churn, code, and network metrics

# Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories (cont)

- There are defects found during functional tests, during systems tests, and post-release [3]
- From an academic perspective the authors of [3] think that research in software metrics should focus on metric selection within defect prediction modeling because data collection is hard, and defect predictors have the potential to have a high payoff in combating common problems with software engineering and project management.

# Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study

- Using software dependencies and churn, Nagappan and Ball build models for predicting the post release failure of system binaries. They do this using the real system, Windows Server 2003
- Churn: a measurement of the amount of code change taking place within a software unit over time. Usually taken from a repository system, not difficult to obtain.
- The idea is that when there is a lot of dependency between two pieces of code, the changes in one piece will propagate changes in the other pieces.
- The relationship between dependence and churn is directly related, the more dependencies that exist in the code, the more changes that will propagate from one update.
- The changes are likely to lead to faults.

# Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study

- Results of Nagappan and Ball's experiments show that they were right about the propagation of code changes that occurs when there are more dependencies in code.
- They also found through software metrics that failures had a statistically significant relationship between churn and software faults. They were able to show that their model for failure proneness was also statistically significant.
- The impact of this research is that production systems can be evaluated post release for fault likelihood and businesses can plan for the expense of maintaining the software past its release date.
  - This is the most expensive phase of software projects.

This information was about this paper [8]

# Software Engineering Metrics: What Do they Measure and How Do We Know?

- Kaner and Bond argue that metrics in software engineering are not yet properly used. In part this is because the validity of the the metrics that are used is not emphasized.[10]
- This could be due to many reasons, one being that software engineering is traditionally a empirical field. If the software is working the default action is typically to leave it alone.
- Also, one of the most common form of software metrics is automated and user testing. This happens to be very expensive and one of the first things to get cut from the development lifecycle. So if a company or organization is going to cut testing, it seems likely to me that they would also not use metrics for the same reasons.

# Software Engineering Metrics: What Do they Measure and How Do We Know? (cont.)

- Balanced score cards are a common technique for helping to prevent developers from “putting all their eggs in one basket” when it comes to choosing a metric to use to monitor the quality of their system.
- The conclusions that the authors draw is that software development is complex by the nature of the task, so it makes sense that we need complex metrics to help us understand the effects of our changes and the inner workings of systems as they get larger and more complex.
- This is a very similar argument to that proposed by Fred Brooks about the complexity of software. As he says there is no one silver bullet to fixing the complexity. However, I think that research is heading in a direction where developers and managers can begin to use statistics to make ascertaining quality of software less difficult.

This information was based on this paper [10]



# References

- [1] Vasilescu, B., Serebrenik, A., van den Brand, M. (2011). By No Means: A Study on Aggregating Software Metrics. *WETSoM'11* (May 24, 2011), Waikiki, Honolulu, HI USA.
- [2] Mair, C., Shepperd, M. (2011) Human Judgement and Software Metrics: Vision for the Future. *WETSoM'11* (May 24, 2011), Waikiki, Honolulu, HI USA.
- [3] Misirli, A., Caglayan, B., Miransky, A., Bener, A., Ruffolo, N. (2011). Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories. *WETSoM'11* (May 24, 2011), Waikiki, Honolulu, HI USA.
- [4] Agile Software Development: Principles, Patterns and Practices. Pearson Education. Robert Cecil Martin (2002)
- [5] <http://www.sdtimes.com/link/34157>
- [6] <http://www8.cs.umu.se/education/examina/Rapporter/JaveedAli.pdf>
- [7] <http://www.codeproject.com/Articles/28440/When-Why-and-How-Code-Analysis>
- [8] Nagappan, N., Ball, T. (2007) Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study. *IEEE First International Symposium on Empirical Software Engineering Measurement*.
- [9] <http://groups.engin.umd.umich.edu/CIS/tinytools/cis375/f00/dsqi/main.html>
- [10] Kaner, C., Bond, W. (2004) Software Engineering Metrics: What Do They Measure and How Do we Know? *10<sup>th</sup> International Software Metrics Symposium*