Brunel University London

Software Development and Management
CS2002

Dr Giuseppe Destefanis
giuseppe.destefanis@brunel.ac.uk

Lecture 8

Software Quality and Metrics


A problem has been d
to your computer.
Attempt to reset the
If this is the first
restart your compute
these steps:
Check to make sure a
Fire exit
Keep clear


30%
70%
BUSINESS PLAN
time
money
quality
USER
CLOUD
USER
QUALITY
cost
quality
price
EXPENSES
INCOME

What is quality?

Different perspectives on software quality

- **Users** judge **external characteristics** (e.g. correct functionality, number of failures etc)
- **Developers** judge **internal characteristics** (e.g. types of faults)

How do we know that we are designing a good piece of software?

Software quality attributes

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

Software “fit for purpose”

- Has the software been properly tested?
- Is the performance of the software acceptable for normal use?

Software “fit for purpose”

- Is the software usable?
- Is the software well-structured and understandable?
- How programming and documentation standards been followed in the development process?

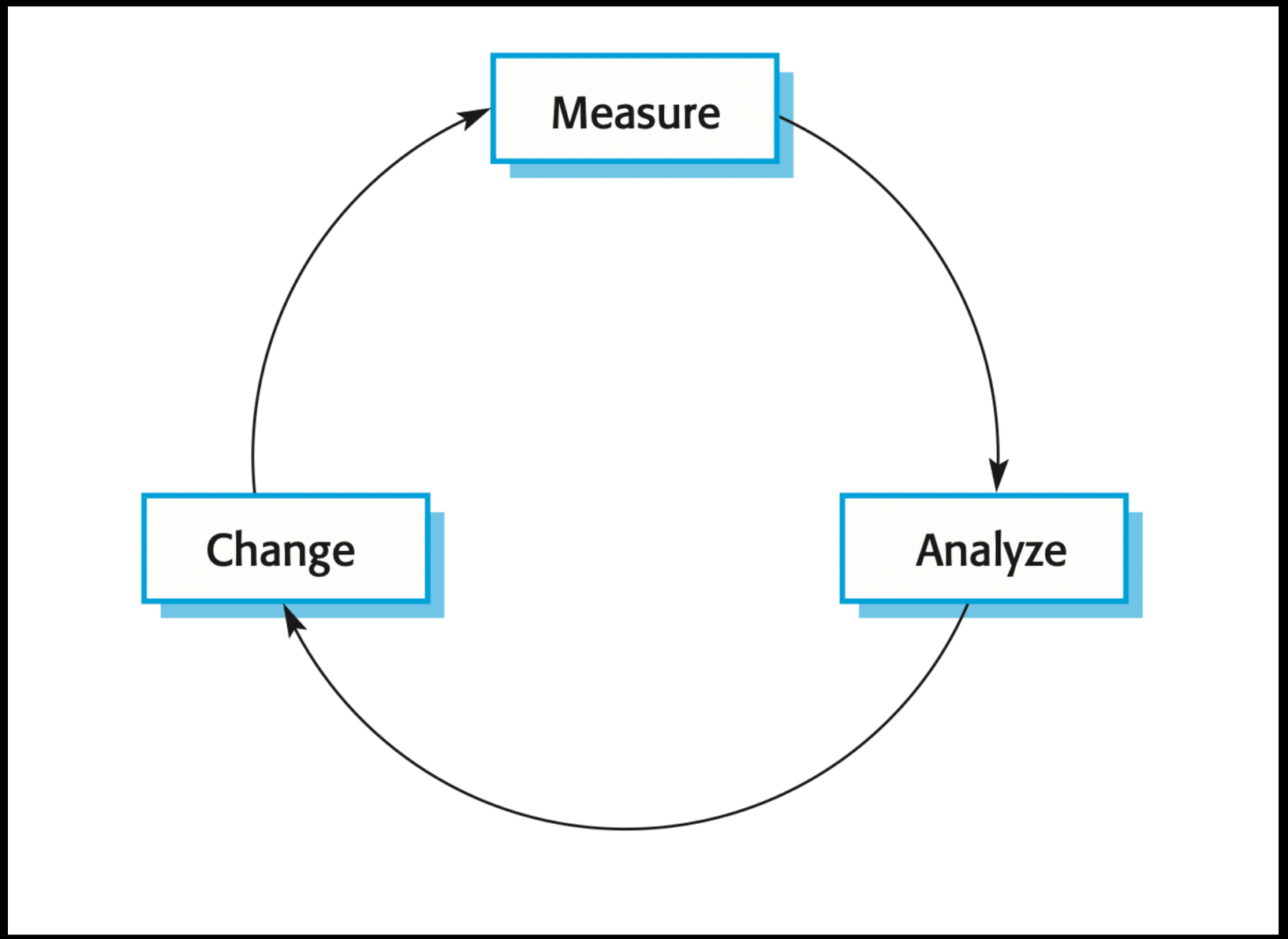
Software Process improvement

- To enhance the quality of their software, many software companies have turned to software process improvement, reducing costs or accelerating their development processes

Software Process improvement

- Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time

Software improvement cycle



Measure

A software measure is the quantitative indication of the extent, amount, dimension or size of some attributes of a product or process

- It is, typically, a single data point, e.g., 1118 lines of code

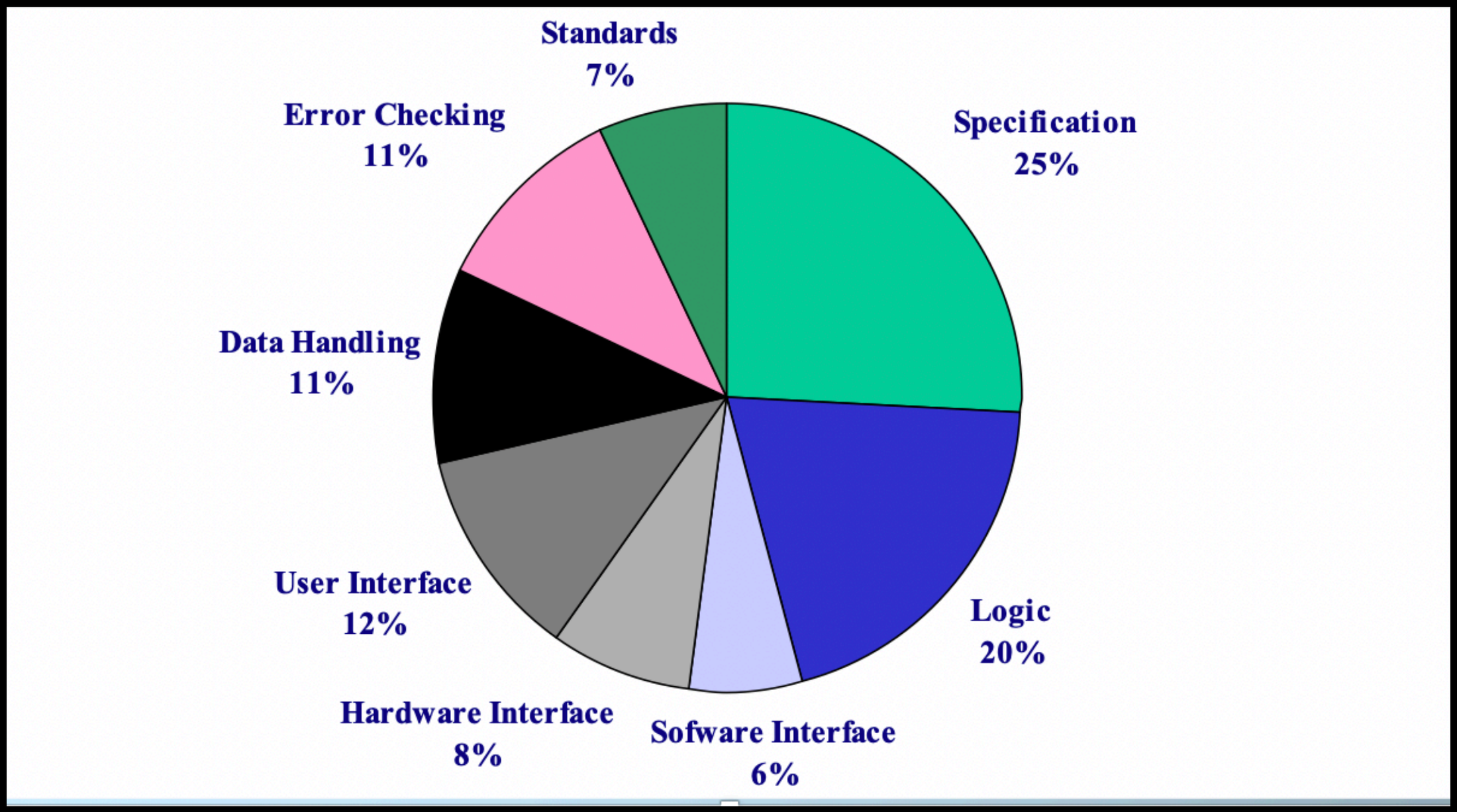
Metric

A software metric represents the degree to which a system, component or process possesses a given attribute

Typically connects a few measures, e.g. the average number of errors found per person hour

A combination of metrics can provide insight into the quality of software process, project or product

Causes and origin of defects



Software Product metrics

- Focus on Deliverable Quality
- Size
- Function
- Complexity: McCabe’s metric using the Control Flow Graph
- Structure: Henry & Kafura’s metric using the Information Flow

Size

- **Size oriented:** lines of code (**LOC**) approach
- This is easy to measure, but with important limitations
    1. Does it mean that Engineer A who wrote 1000 LOC are more productive than Engineer B who wrote 600 LOC?
    2. Does it mean that Engineer A wrote better quality code?
- In general, for a specific problem, the more lines of code in a software module, the less understandable and maintainable the module is likely to be.

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.
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.

OO metrics

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 they 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.
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. 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.

Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data
- Don’t use metrics to appraise or threaten individuals or teams!
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Metrics data that indicate a problem area should not be considered “negative.” These data are merely an indicator for process improvement
- Don’t focus on just a single metric to the exclusion of other important metrics
- There is still considerable amount of research that needs to be done to better understand the link between metrics and quality.

Reading

- Chapter 24 of Sommerville