Quality Management

Andy Carpenter

School of Computer Science

(Andy.Carpenter@manchester.ac.uk)

Quality management and agile development

Quality management and agile development

- Quality management in agile development is informal rather than document-based.
- It relies on establishing a quality culture, where all team members feel responsible for software quality and take actions to ensure that quality is maintained.
- The agile community is fundamentally opposed to what it sees as the bureaucratic overheads of standards-based approaches and quality processes as embodied in ISO 9001.

- *Check before check-in*
  - Programmers are responsible for organizing their own code reviews with other team members before the code is checked in to the build system.
- *Never break the build*
  - Team members should not check in code that causes the system to fail. Developers have to test their code changes against the whole system and be confident that these work as expected.
- *Fix problems when you see them*
  - If a programmer discovers problems or obscurities in code developed by someone else, they can fix these directly rather than referring them back to the original developer.

Reviews and agile methods

- The review process in agile software development is usually informal.
- In Scrum,, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In Extreme Programming, pair programming ensures that code is constantly being examined and reviewed by another team member.

Pair programming

- This is an approach where 2 people are responsible for code development and work together to achieve this.
- Code developed by an individual is therefore constantly being examined and reviewed by another team member.
- Pair programming leads to a deep knowledge of a program, as both programmers have to understand the program in detail to continue development.
- This depth of knowledge is difficult to achieve in inspection processes and pair programming can find bugs that would not be discovered in formal inspections.

Pair programming weaknesses

- *Mutual misunderstandings*
  - Both members of a pair may make the same mistake in understanding the system requirements. Discussions may reinforce these errors.
- *Pair reputation*
  - Pairs may be reluctant to look for errors because they do not want to slow down the progress of the project.
- *Working relationships*
  - The pair's ability to discover defects is likely to be compromised by their close working relationship that often leads to reluctance to criticize work partners.

Agile QM and large systems

- When a large system is being developed for an external customer, agile approaches to quality management with minimal documentation may be impractical.
  - If the customer is a large company, it may have its own quality management processes and may expect the software development company to report on progress in a way that is compatible with them.
  - Where there are several geographically distributed teams involved in development, perhaps from different companies, then informal communications may be impractical.
  - For long-lifetime systems, the team involved in development will change. Without documentation, new team members may find it impossible to understand development.

Software Measurement

Progress

Time taken

Resources used

Complexity

Test coverage

Design time

Development time

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.

Size Metric

Lines of Code (LOC)

What is a line of code ?

Non-Commented Lines of Code (NCLOC)

Source Lines of Code (SLOC)

McCabe's Cyclomatic Metric (1)

Structure of code

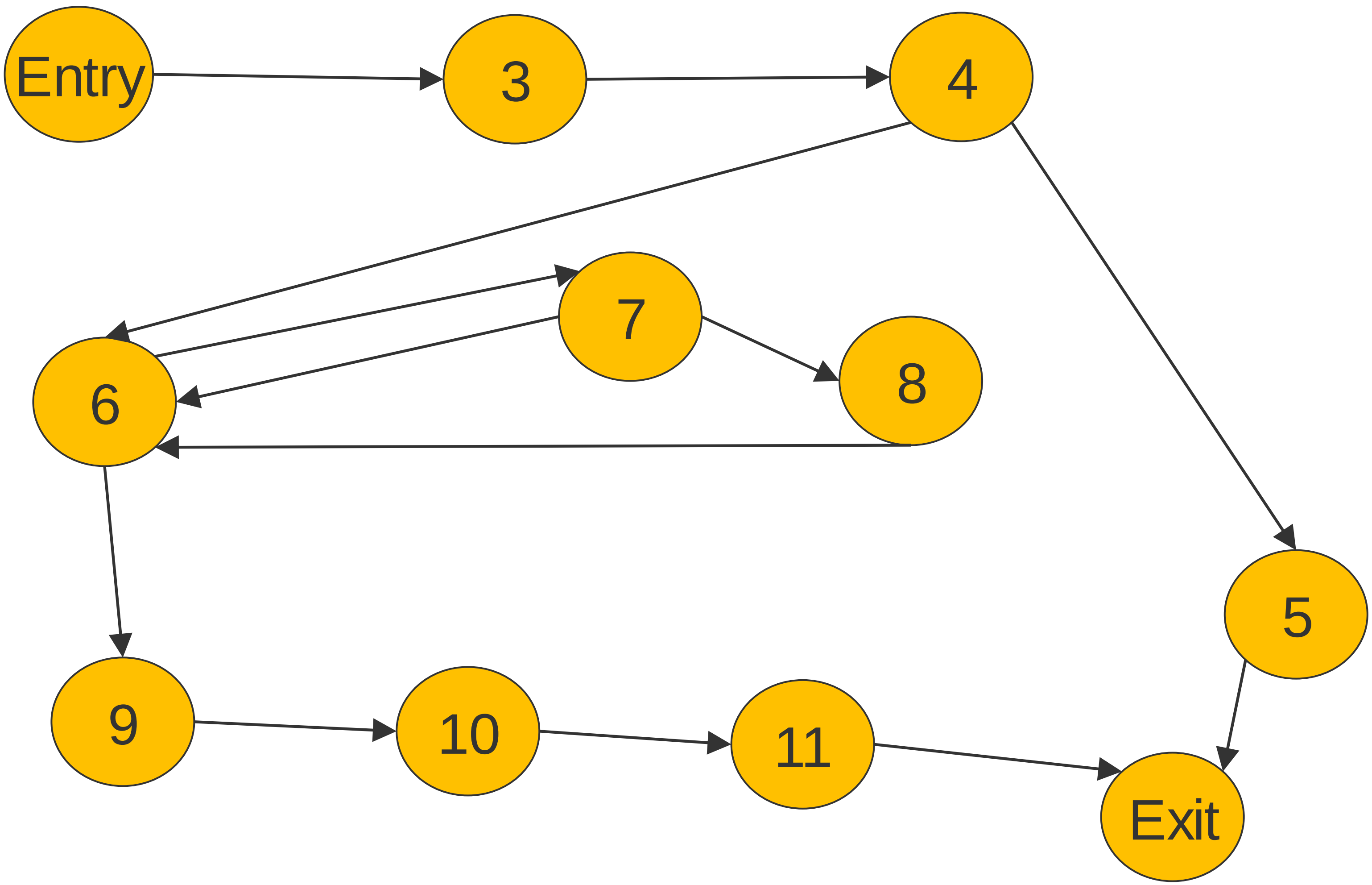
Linear paths through code

- Calculated from control-flow graph
  - representation of program
  - nodes are executable statements
  - arcs are flow of control

Example Q Sort Code

```
void iqsort(int *a, int n)
{
  int I, j;
  if (n <= 1)
    return;
  for (I = 1, j = 0; I < n; i++)
    if (a[i] < a[0])
      swap (++j, I, a);
  Swap(0, j, a);
  iqsort)(a, j)
  iqsort(a+j+1, n-j-1);
}
```

CFG of Q Sort



McCabe's Cyclomatic Metric (2)

- Cyclomatic Complexity (CC) = E – N + 2P
- Where:
  - E is number of edges
  - N is number of nodes
  - P is number of separate procedures or functions

Handles disconnected graphs

- For Q Sort: 13 - 11 + 2 x 1 = 4

MANCHESTER 1824
The University of Manchester

Halstead Complexity Metrics (1)

Set of metrics

Focus on textual source

Operators

Elements of control

if, while, …

Function declaration

Operands

Data

Variable names

Variable values

Function names

Used as basis for other metrics

Halstead Complexity Metrics (2)

- Calculate four values
  - n1 number of distinct operators
  - N1 total number of operator occurrences
  - n2 number of distinct operands
  - N2 total number of operand occurrences
- From which:
  - Program length: N = N1 + N2
  - Program vocabulary: n = n1 + n2
  - Volume = $N*(\log_2 n)$
  - Difficulty = n1/2 * N2/n2
  - Effort = Difficulty * Volume

Halstead Complexity Metrics (2)

Operator or operand?

Syntactic elements?

Parentheses

Semi-colons

Comments

Validity?

Function Point

Size without source code

Albrecht Function Point

External perspective

Interactions between system and environment

External inputs

External inquiries

Internal files

External files

External outputs

Albrecht Function Point (1)

Each interaction is graded

Average

Simple

Difficult

Graded interaction count tabulated

and weighted

Category	Simple	Average	Difficult
External input	3 x n	4 x n	6 x n
External output	4 x n	5 x n	7 x n
External inquiries	3 x n	4 x n	6 x n
Internal files	7 x n	10 x n	15 x n
External files	5 x n	7 x n	10 x n

Albrecht Function Point (2)

- Calculate Unadjusted Function point Count (UFC)
  - $\sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} \times x_{ij}$
- Where
  - x_{ij} is number of interactions of type *i* with complexity *j*
  - w_{ij} is weight of interactions of type *i* with complexity *j*
- Next calculate Technical Complexity Factor (TCF)
  - …
- Finally compute Function Point Count (FPC)
  - FPC = UFC x TCF

Modularity Metrics (1)

Coupling

Two elements related

Cohesion

Strongly interdependent

Syntactically related

Similar relationship to other classes/libraries

Changed at similar point in time

Textually similar

Share common author

Contain similar/identical code fragments

Modularity Metrics (2)

- Coupling Between Objects (CBO)
  - Value for every class in system
    - Number of classes to which coupled
      - Reading from/ writing to variable
      - Function/method call
        - » Polymorphic count all possible

Modularity Metrics (3)

- Lack of Cohesion between Methods (LCOM)
  - Number of method pairs without access to shared class attributes minus number of method pairs with access to shared attributes

Much criticised

Maintainability Index

Combines

Halstead's volume (HV)

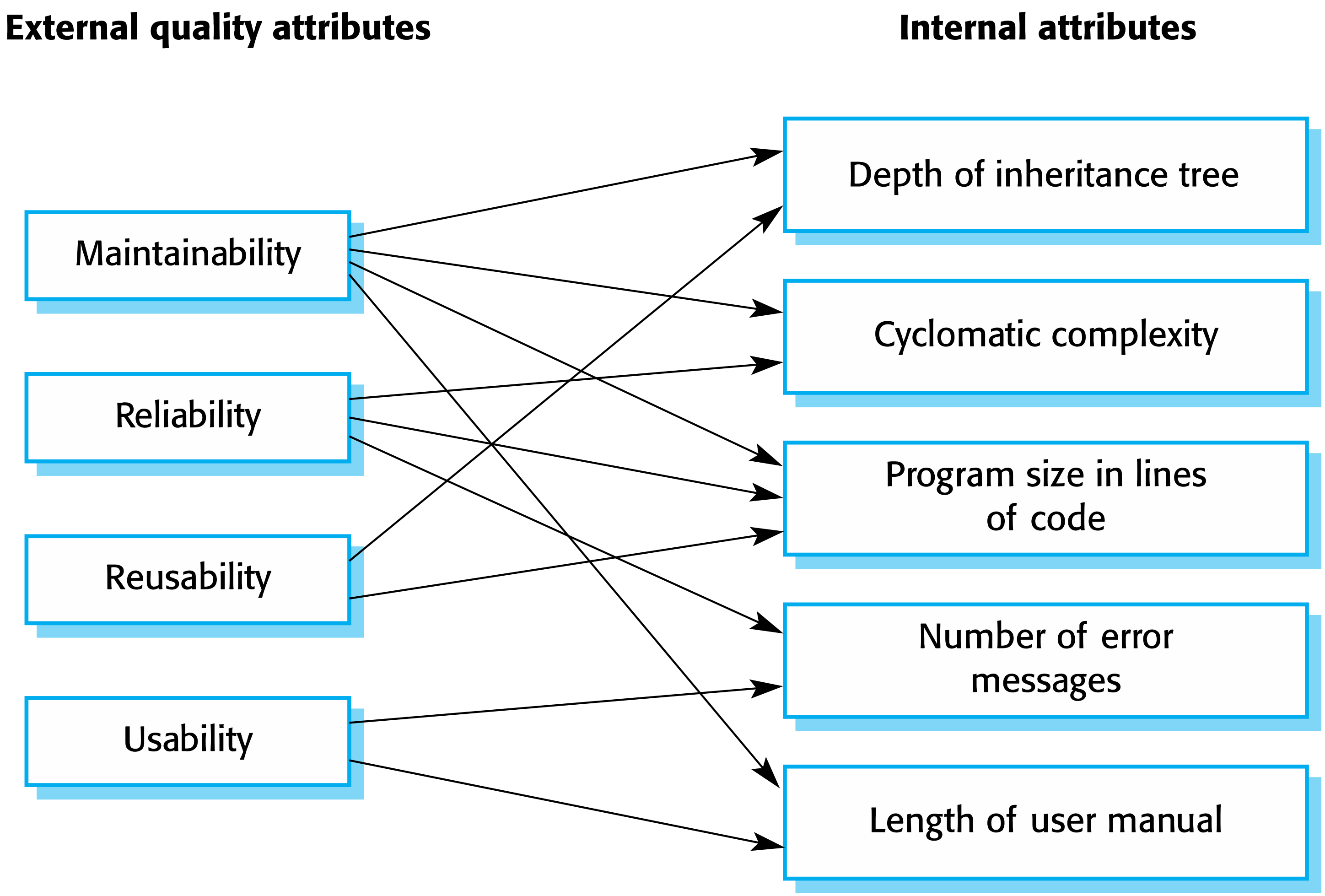
Lines of Code (LOC)

Cyclometric Complexity (CC)

Others

- 171 - 5.2ln(HV) - 0.23CC - 16.2ln(LOC) + $50\sin\sqrt{2.46 * COM}$

Relationships between internal and external software



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.

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.

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.

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.

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.

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.