ECS505U
SOFTWARE ENGINEERING

MUSTAFA BOZKURT

LECTURER IN SOFTWARE ENGINEERING

Queen Mary
University of London

WEEK 10

SOFTWARE QUALITY ASSURANCE & METRICS

SOFTWARE QUALITY ASSURANCE

OBJECTIVES

- Gain a better understanding of software quality
- Learn some of the prominent software quality models
- Gain a better understanding of quality assurance
- Understand the nature of software reliability

SOFTWARE QUALITY

What does ‘quality’ mean in software engineering?

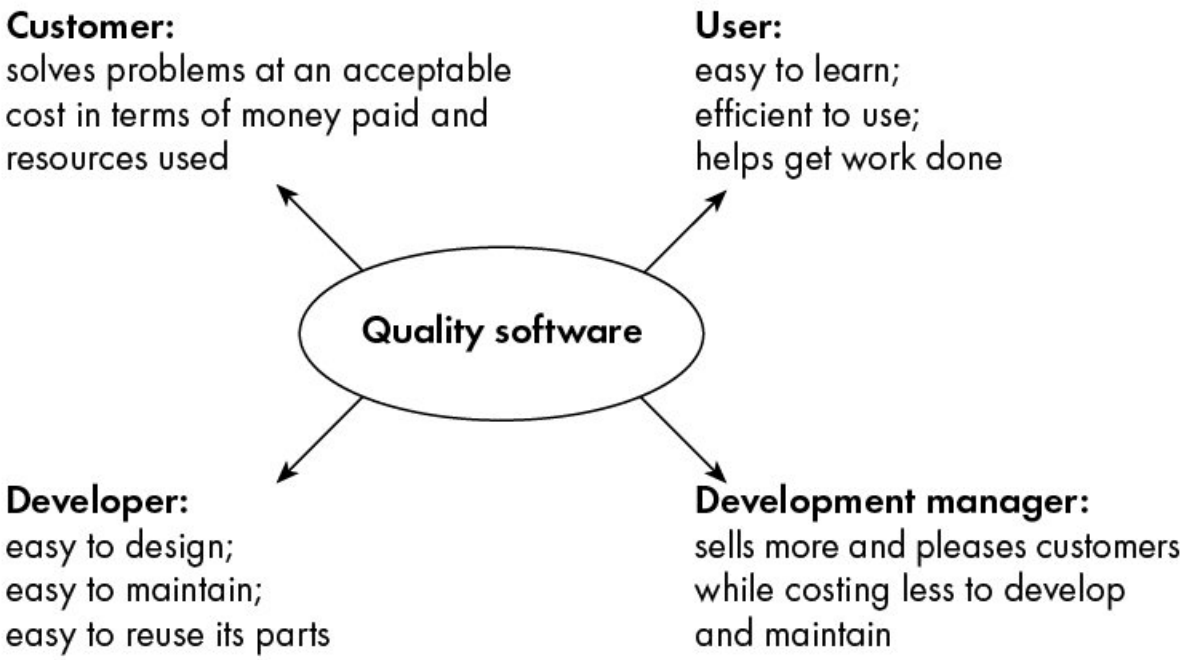
Queen Mary University of London

SOFTWARE QUALITY

Software quality is:

- The degree to which a system, component, or process meets specified requirements.
- The degree to which a system, component, or process meets customer or user needs or expectations. [IEEE]
- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. [Pressman]
- The degree of excellence of something. We measure the excellence of software via a set of attributes. [Glass]

SOFTWARE QUALITY



It has a different meaning according to different stakeholders

SOFTWARE QUALITY ATTRIBUTES

Queen Mary University of London

There are several definitions of the factors (attributes) that affect quality such as McCall's Quality Factors and ISO 9126 Quality Factors.

SOFTWARE QUALITY ATTRIBUTES

Queen Mary University of London

ISO 9126 Quality Factors:

Functionality: The degree to which the software satisfies stated needs as indicated by the following sub-attributes: suitability, accuracy, interoperability, compliance, and security.

Reliability: The amount of time that the software is available for use as indicated by the following sub-attributes: maturity, fault tolerance, recoverability.

Usability: The degree to which the software is easy to use as indicated bythe following sub-attributes: understandability, learnability, operability.

Efficiency: The degree to which the software makes optimal use of system resources as indicated by the following sub-attributes: time behavior, resource behavior.

Maintainability: The ease with which repair may be made to the software as indicated by the following sub-attributes: analyzability, changeability, stability, testability.

Portability: The ease with which the software can be transposed from one environment to another as indicated by the following sub-attributes: adaptability, installability, conformance, replaceability.

SOFTWARE QUALITY ATTRIBUTES

Queen Mary University of London

Reliability

Functionality

Usability

Efficiency

Product in operation (user perspective)

Maintainability

Reusability

Portability

Testability

Product revision (developer perspective)

SOFTWARE QUALITY ATTRIBUTES

Queen Mary University of London

Often, software engineers improve one quality at the expense of another.

- Efficiency vs maintainability
- Reliability vs efficiency
- Usability vs efficiency & maintainability

SOFTWARE QUALITY ELEMENTS

Queen Mary University of London

Standards: The IEEE, ISO, and other standards organizations have produced a broad array of SE standards and related documents. Standards may be adopted voluntarily by an organization or imposed by the customer or other stakeholders.

Inspections/Reviews: Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors.

Testing: Covered!

Configuration Management: Software configuration management (SCM is the task of tracking and controlling changes in the software. SCM practices include revision control, establishment of baselines and few other activities.

Risk Management: QA ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

and few others…

SOFTWARE QUALITY ELEMENTS

Queen Mary University of London

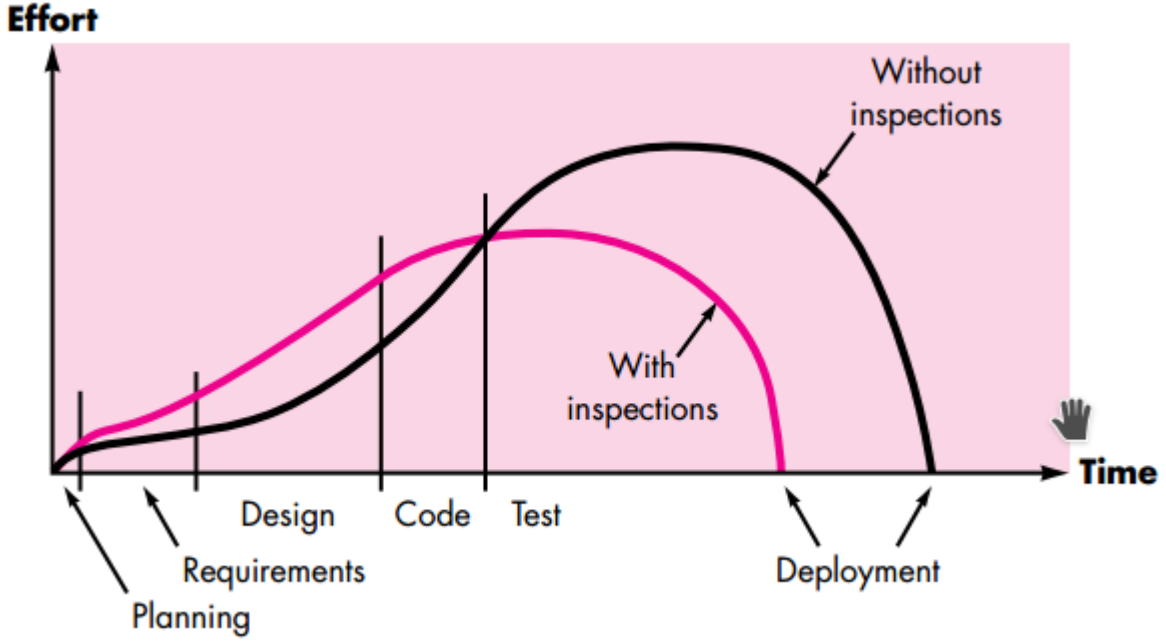
Configuration Management: Software configuration management (SCM is the task of tracking and controlling changes in the software. SCM practices include revision control, establishment of baselines and few other activities.

A baseline is a software configuration management concept that helps you to control change without seriously impeding justifiable change. The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

TESTING VS INSPECTION

- Physical execution **vs** mental execution using documentation
- Obvious **vs** subtle consequences
- Inspection can easily detect maintainability and efficiency defects.



Fagan, M., “Advances in Software Inspections,” IEEE Trans. Software Engineering, vol. 12, no. 6

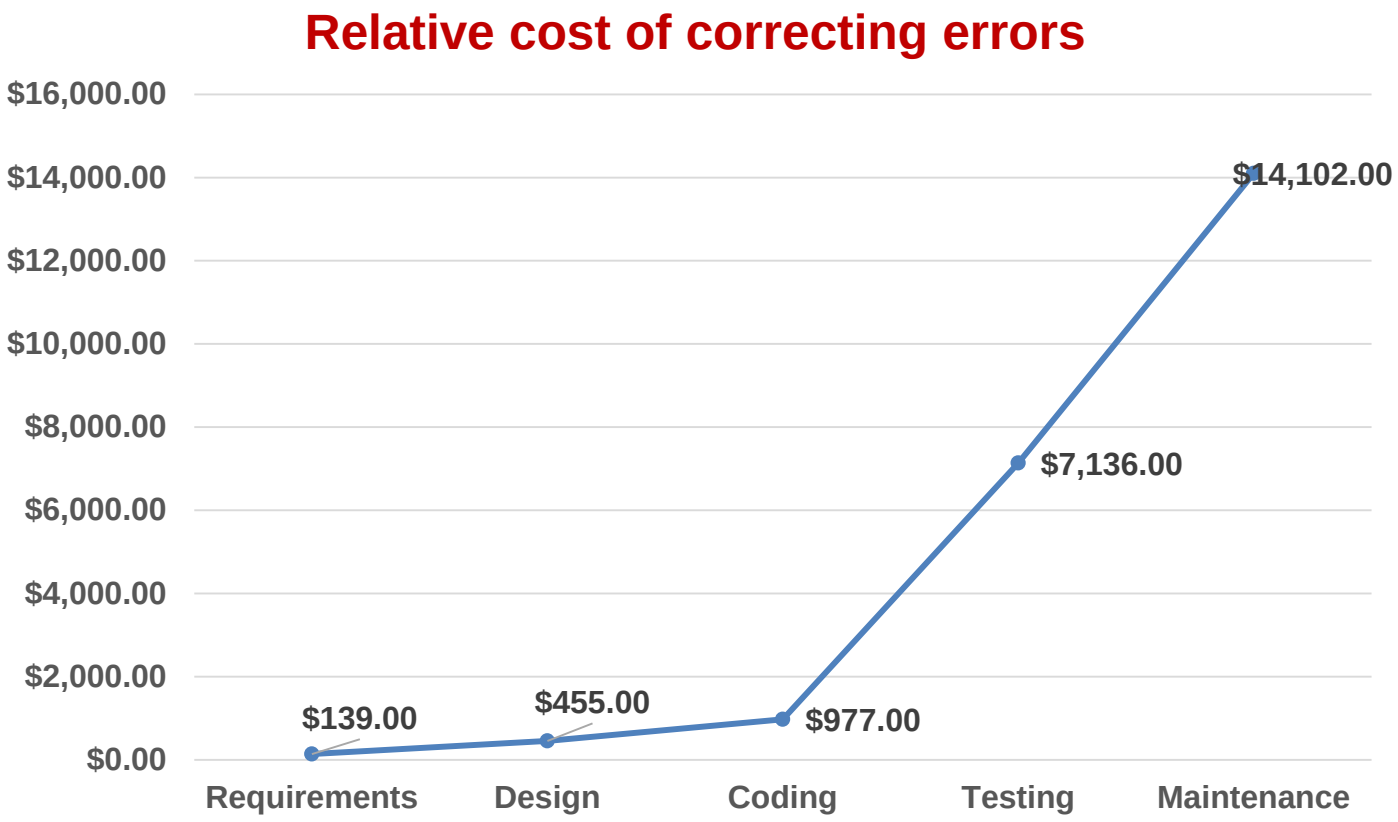
COST OF QUALITY

Queen Mary University of London

COST OF QUALITY

Category	Definition	Typical Costs for Software
Internal failures	Quality failures detected prior to product shipment	Defect management, rework, retesting
External failures	Quality failures detected after product shipment	Technical support, complaint investigation, defect notification
Appraisal	Discovering the condition of the product	Testing and associated activities, product quality audits
Prevention	Efforts to ensure product quality	SQA administration, inspections, process improvements, metrics collection and analysis

COST OF QUALITY



[Boehm 2001]

SOFTWARE QUALITY DILEMMA

Queen Mary University of London

Bertrand Meyer once said:

"*If you produce a software system that has terrible quality, you lose because no one will want to buy it. If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway. Either you missed the market window, or you simply exhausted all your resources. So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete*."

SOFTWARE QUALITY MODELS

Queen Mary University of London

Two main approaches:

Standard Models:

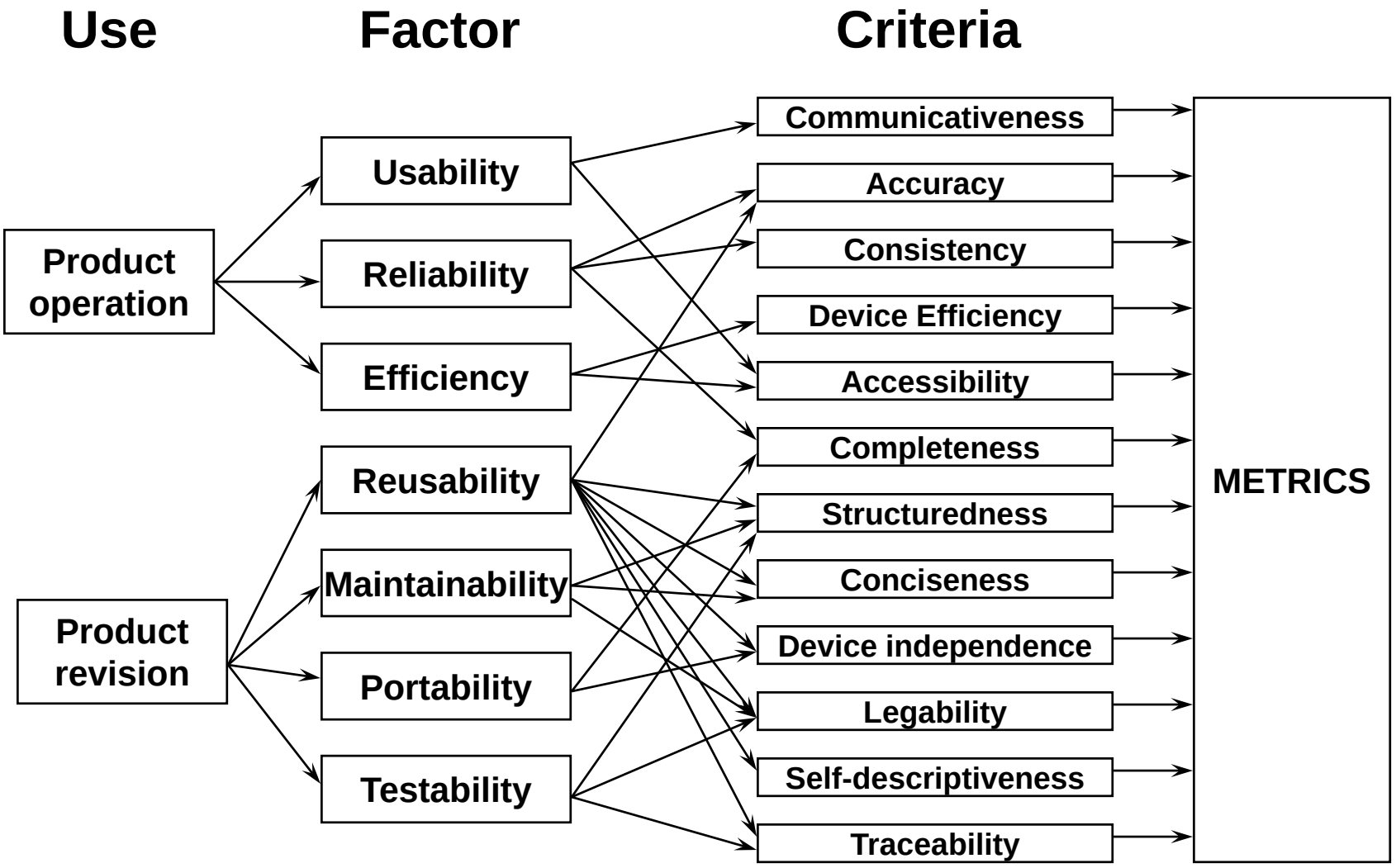
- McCall
- ISO/IEC 9126

Application or company specific quality models

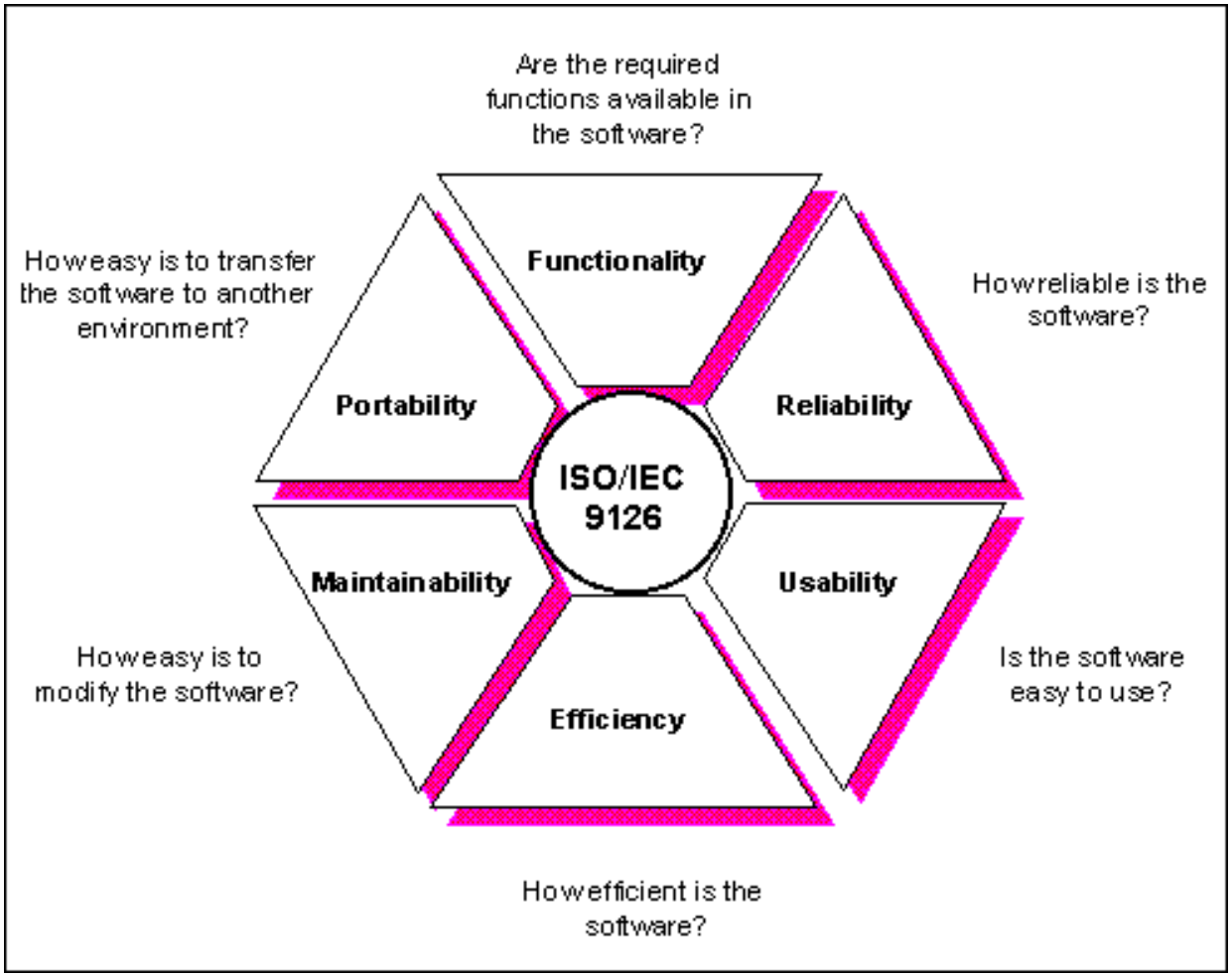
- FURPS
- GQM Approach

Queen Mary University of London

SOFTWARE QUALITY MODELS



ISO 9126


Are the required functions available in the software?
Functionality
How reliable is the software?
Reliability
Is the software easy to use?
Usability
How efficient is the software?
Efficiency
How easy is to modify the software?
Maintainability
How easy is to transfer the software to another environment?
Portability
ISO/IEC 9126

RELIABILITY

Probability the system operates without failure in a given period of time under a given operational environment

WHAT IS A SOFTWARE FAILURE?

Queen Mary University of London

Formal view

- deviation from specified behaviour

Engineering view

- deviation from required, specified or expected behaviour

RELATIONSHIP BETWEEN FAULTS AND FAILURES

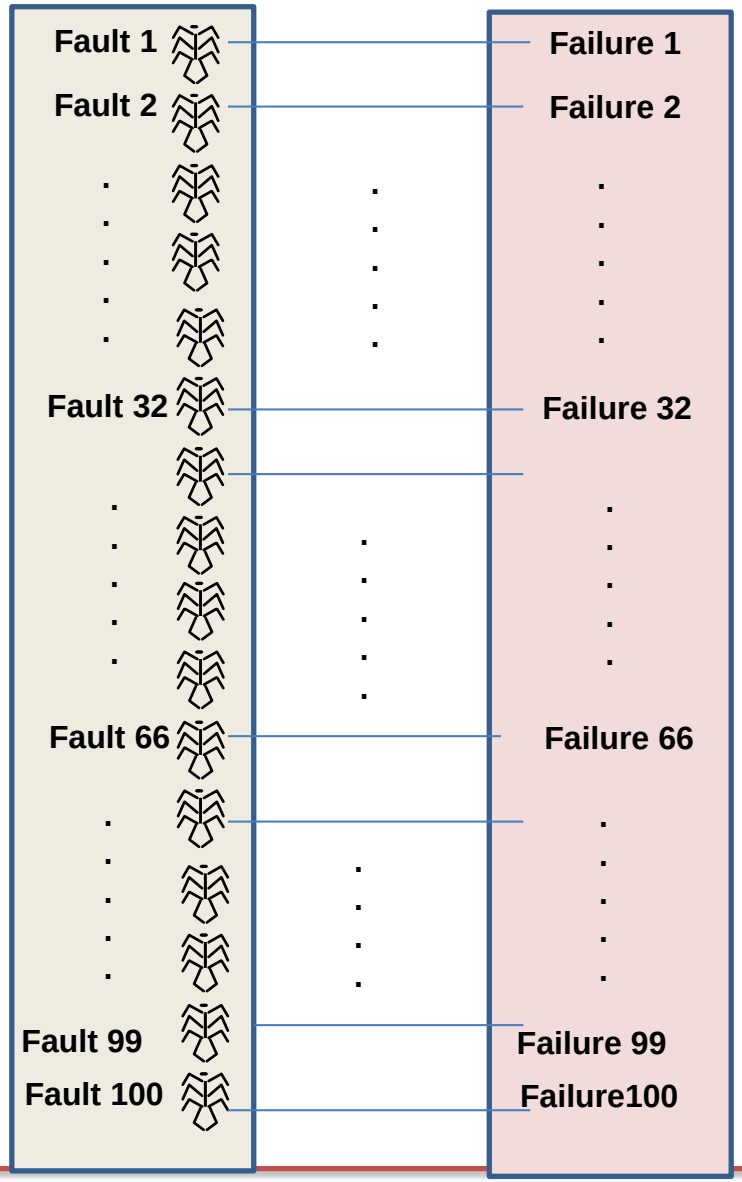
Queen Mary University of London

- Not all faults are equally important
- What matters is whether they trigger important failures
- The empirical data about the relationship between faults and failures is astonishing

Queen Mary University of London

Imagine 100 faults in the system

Assume each fault triggers one distinct failure



Failures classified by the importance (MTTF)

"Important failures" (2%)

"Unimportant failures" (33%)

"Totally irrelevant failures" (65%)

[Adams 84]

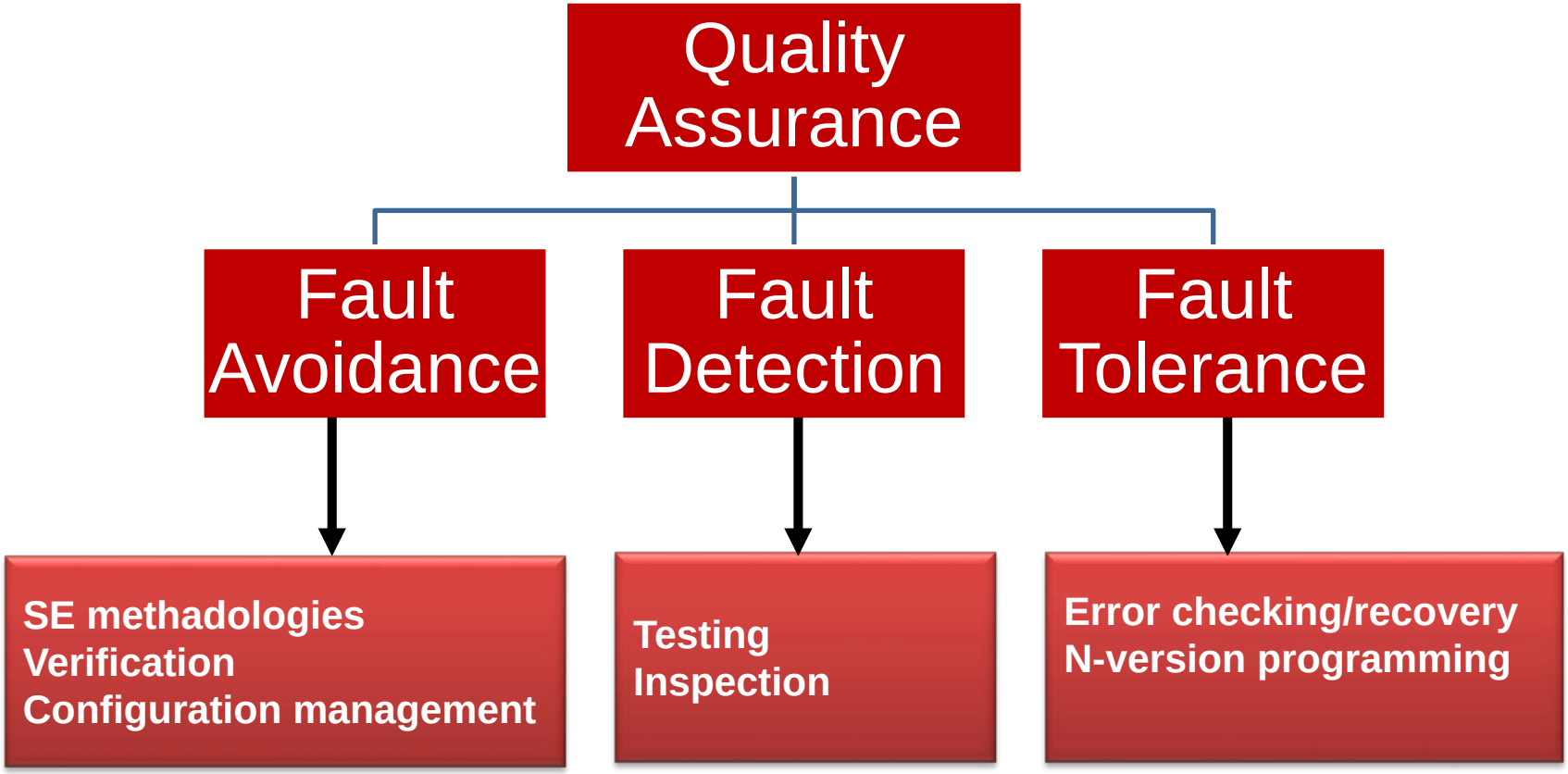
RELATIONSHIP BETWEEN FAULTS AND FAILURES

Queen Mary University of London

- Most faults are benign
- For most faults: removal will not greatly improve reliability
- Large reliability improvements comes from removing faults that lead to frequent failures
- Don’t stop looking for faults!

SOFTWARE QUALITY ASSURANCE

Queen Mary University of London



SUMMARY

- Software quality is multi-faceted
- Do not equate faults and failures
- Reviews and inspections are efficient QA procedures

SOFTWARE METRICS

Queen Mary
University of London

OBJECTIVES

- Understand why measurement is important
- Understand the basic metrics approaches used in industry
- Know how to extract and apply relevant metrics

SOFTWARE METRICS MOTIVATION

Queen Mary University of London

- Quantitative tool to manage risk in software projects
- Software metrics has always been driven by two key objectives:
  - software cost/resource estimation
  - software quality control and estimation
- Measuring ‘size’ and measuring ‘defects’ is central

SOFTWARE METRICS MOTIVATION

Queen Mary University of London

Also

- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

SOFTWARE METRICS DEFINITIONS

Queen Mary University of London

Measure: A quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process.

- A single data point (e.g. number of defects in a component)

Measurement: The act of determining a measure

Metric: A quantitative measure of the degree to which a system, component or process possesses a given attribute.

- Metrics relate measures (e.g. Average number of defects found in inspections)
- Relate data points to each other

Indicator: A metric or series of metrics that provide insight into a process, project or product

SOFTWARE METRICS

Balanced scorecard

Bugs per line of code

Code coverage

Cohesion

Comment density[1]

Connascent software components

Coupling

Cyclomatic complexity (McCabe's complexity)

DSQI (design structure quality index)

Function point analysis

Halstead Complexity

Instruction path length

Maintainability index

Number of classes and interfaces

Number of lines of code

Number of lines of customer requirements

Program execution time

Program load time

Program size (binary)

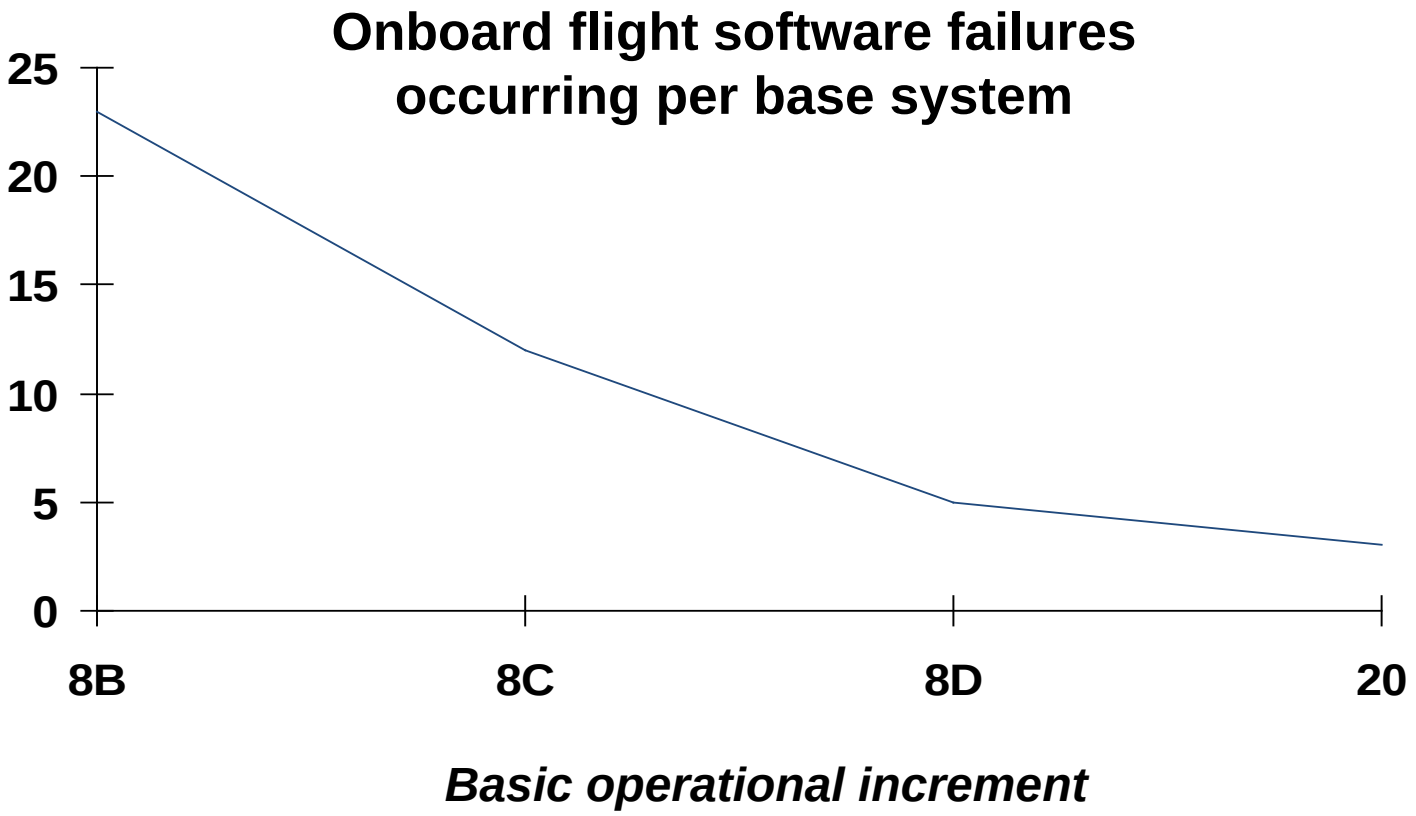
Robert Cecil Martin's software package metrics

Weighted Micro Function Points

Function Points and Automated Function Points, an Object Management Group standard[2]

CISQ automated quality characteristics measures

IBM SPACE SHUTTLE SOFTWARE METRICS PROGRAM



A BRIEF HISTORY OF SOFTWARE METRICS

Queen Mary University of London

- First book (Gilb 1976) but LOC-based measures since 1960's
- LOC used as surrogate for different notions of software size
- Drawbacks of LOC as size measure led to complexity metrics and function point metrics
- Further metrics for design level, and for different language paradigms

THE ENDURING LOC MEASURE

Queen Mary University of London

LOC: Number of Lines Of Code

- The simplest and most widely used measure of program size. Easy to compute and automate
- Used (as normalising measure) for
  - effort/cost estimation (Effort = f(LOC))
  - quality assessment/estimation (defects/LOC))
  - productivity assessment (LOC/effort)

Alternative (similar) measures

- KLOC: Thousands of Lines Of Code
- KDSI: Thousands of Delivered Source Instructions
- NCLOC: Non-Comment Lines of Code
- Number of Characters or Number of Bytes

LOC EXAMPLES

```c
# include <stdio.h>

int main() {
    printf("\nHello world\n");
}
```

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hello world!" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```

PROBLEMS WITH LOC TYPE MEASURES

Queen Mary University of London

- No standard definition
- Measures length of programs rather than size
- Wrongly used as a surrogate for:
  - effort
  - complexity
  - functionality
- Fails to take account of redundancy and reuse
- Cannot be used comparatively for different types of programming languages
- Only available at the end of the development life-cycle

METRICS FOR OBJECT-ORIENTED DESIGN

Queen Mary University of London

Size: Size is defined in terms of four views: population, volume, length, and functionality.

Complexity: Like size, there are many differing views of software complexity. Whitmire views complexity in terms of structural characteristics by examining how classes of an OO design are interrelated to one another.

Coupling: The physical connections between elements of the OO design

Cohesion: Like its counterpart in conventional software, an OO component should be designed in a manner that has all operations working together to achieve a single, well-defined purpose.

Volatility: Volatility of an OO design component measures thelikelihood that a change will occur

THE 'DEFECT DENSITY' MEASURE: AN IMPORTANT HEALTH WARNING??

Queen Mary University of London

$$\text{defect density} = \frac{\text{number of defects found}}{\text{system size (KLOC)}}$$

Defect density is used as a de-facto measure of software quality

However, the biggest problem is the assumption that somehow faults and failures are 'equally bad' and that defects are therefore in some sense homogeneous.

SOFTWARE SIZE ATTRIBUTES

Queen Mary University of London

- Length the physical size of the product
- Functionality measures the functions supplied by the product to the user
- Complexity
  - Problem complexity
  - Algorithmic complexity
  - Structural complexity
  - Cognitive complexity

THE SEARCH FOR MORE DISCRIMINATING METRICS

Queen Mary University of London

- Capture cognitive complexity
- Capture structural complexity
- Capture functionality (or functional complexity)
- Language independent
- Can be extracted at early life-cycle phases

HALSTEAD'S SOFTWARE METRICS

Queen Mary University of London

A program P is a collection of tokens, classified as either operators or operands.

n_1 = number of unique operators
n_2 = number of unique operands
N_1 = total occurrences of operators
N_2 = total occurrences of operands

Length of P is $N = N_1+N_2$ Vocabulary of P is $n = n_1+n_2$

Theory: Estimate of N is $\hat{N} = n_1 \log n_1 + n_2 \log n_2$

Theory: Effort required to generate P is

$$E = \frac{n_1 N_2 N \log n}{2n_2}$$

Bugs B= $\frac{N \, x \log n}{3000}$

Theory: Time required to program P is T=E/18 seconds

HALSTEAD'S SOFTWARE METRICS

Queen Mary University of London

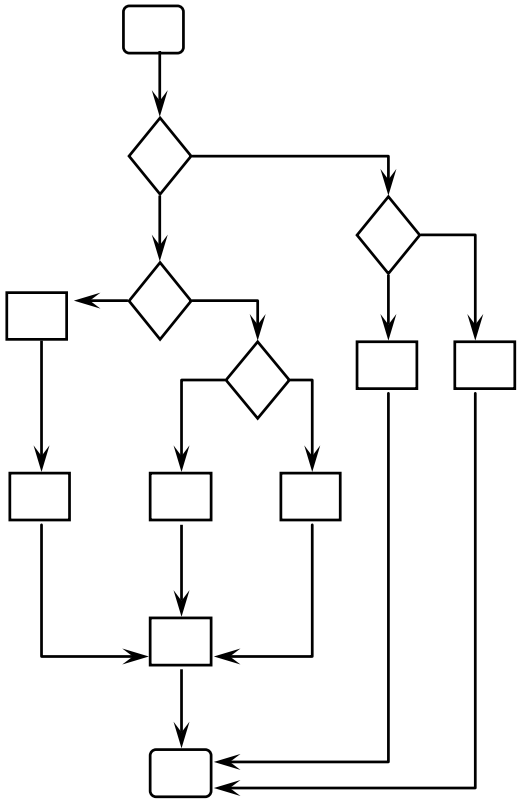
```
main()
{
    int a, b, c, avg;
    scanf("%d %d %d", &a, &b, &c);
    avg = (a + b + c) / 3;
    printf("avg = %d", avg);
}
```

$n_1 = 10$
n_2 7
$\eta = 17$
$N_1 = 16$
$N_2 = 15$
$N = 31$

The unique operators are: main, (), {}, int, scanf, &, =, +, /, printf
The unique operands are: a, b, c, avg, "%d %d %d", 3, "avg = %d"

Length $\check{N} = 10 \times \log_2 + 7 \times \log_2 7 = 52.9$
Effort $E = 10x\ 15 \times 31 \times \log_2 17/ 2 \times 7 = 1355.7$
Time $T = 1355.7 /18 = 75.3s$
Bugs $B = 126.7 / 3000 = 0.042$

MCCABE'S CYCLOMATIC COMPLEXITY METRIC *V*



If G is the control flowgraph of program P and G has e edges (arcs) and n nodes

$$v(P) = e-n+2 \times m$$

v(P) is the number of linearly independent paths in G

here e = 16 n =13 *v*(P) = 5

More simply, if d is the number of decision nodes in G then

$$v(P) = d+1$$

McCabe proposed: $v(P)<10$ for each module *P*

PROBLEMS WITH MCCABE'S CYCLOMATIC COMPLEXITY

Queen Mary University of London

- Different tools measure cyclomatic complexity differently.
- Human perception of complexity


```
String getMonthName (int month) {
        switch (month) {
                case 0: return "January";
                case 1: return "February";
                case 2: return "March";
                case 3: return "April";
                case 4: return "May";
                case 5: return "June";
                case 6: return "July";
                case 7: return "August";
                case 8: return "September";
                case 9: return "October";
                case 10: return "November";
                case 11: return "December";
                default: throw new IllegalArgumentException();
        }
}
```

ALBRECHT'S FUNCTION POINTS

Count the number of:

- External inputs
- External outputs
- External inquiries
- External files
- Internal files/interfaces

giving each a 'weighting factor'

The Unadjusted Function Count (UFC) is the sum of all these weighted scores

To get the Adjusted Function Count (FP), multiply by a Technical Complexity Factor (TCF)

$$FP = UFC \times TCF$$

ALBRECHT'S FUNCTION POINTS

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

UFC calculation table

$$UFC = \sum i \quad weight\ (i)$$

where i is the number of items per parameter

ALBRECHT'S FUNCTION POINTS

Queen Mary University of London

Rate each of these factors on a scale of 0 to 5 where 0 means the component has no influence on the system and 5 means the component is essential

F_1 Reliable backup and recovery required

F_2 Data communications

F_3 Distributed functions

F_4 High Performance (time efficiency) required

F_5 Heavily used configuration

F_6 On-line (interactive) data entry

F_7 Must be easy to use

F_8 On-line update

F_9 Complex user interface

F_{10} Complex process

F_{11} Reusability

F_{12} Ease of installation

F_{13} Multiple installation sites

F_{14} Easy to modify

$$TCF = 0.65 + (0.01 \times \sum \mathbf{F_i} \text{ (sum of all factors)})$$

(the factor varies between 0.65 and 1.35 according to Fenton)

ALBRECHT'S FUNCTION POINTS

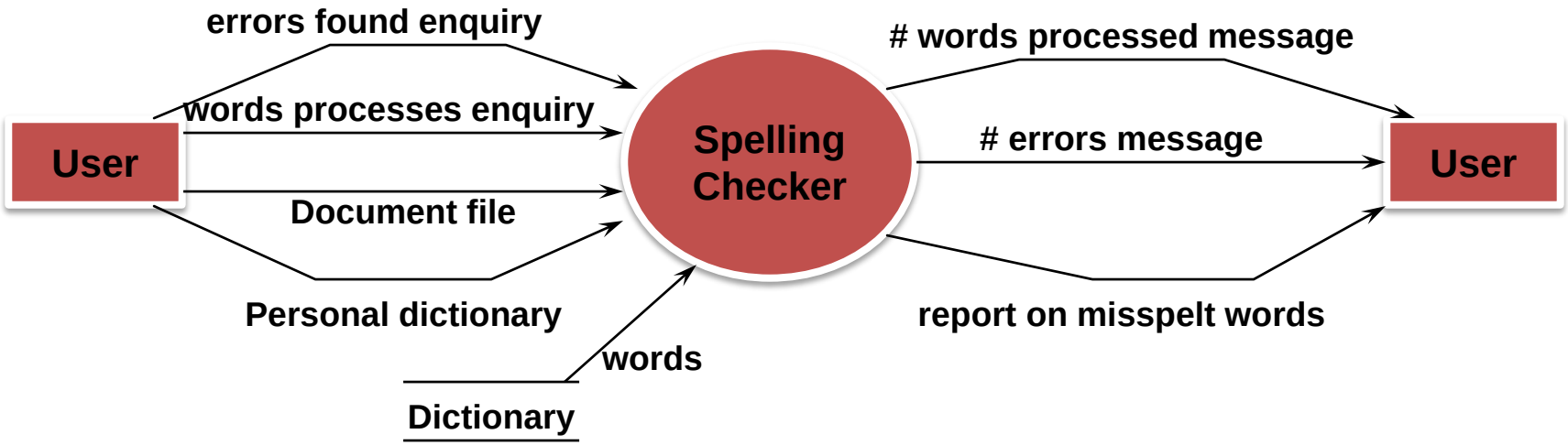
Queen Mary University of London

FP is supposed to be proportional to effort or time required.

One Function Point = 2 days!

FUNCTION POINTS: EXAMPLE

Spell-Checker Spec: The checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number of spelling errors found at any stage during processing



A = # external inputs = 2, B =# external outputs = 3, C = # inquiries = 2,
D = # external files = 2, E = # internal interfaces = 1

Assuming average complexity in each case

UFC = 4A + 5B + 4C +10D + 7E = 58

FUNCTION POINTS: APPLICATIONS

Queen Mary University of London

Used extensively as a ‘size’ measure in preference to LOC

$$\text{Productivity} = \frac{FP}{\text{Person months effort}}$$

$$\text{Quality} = \frac{\text{Defects}}{FP}$$

$$\text{Effort prediction} = E=f(FP)$$

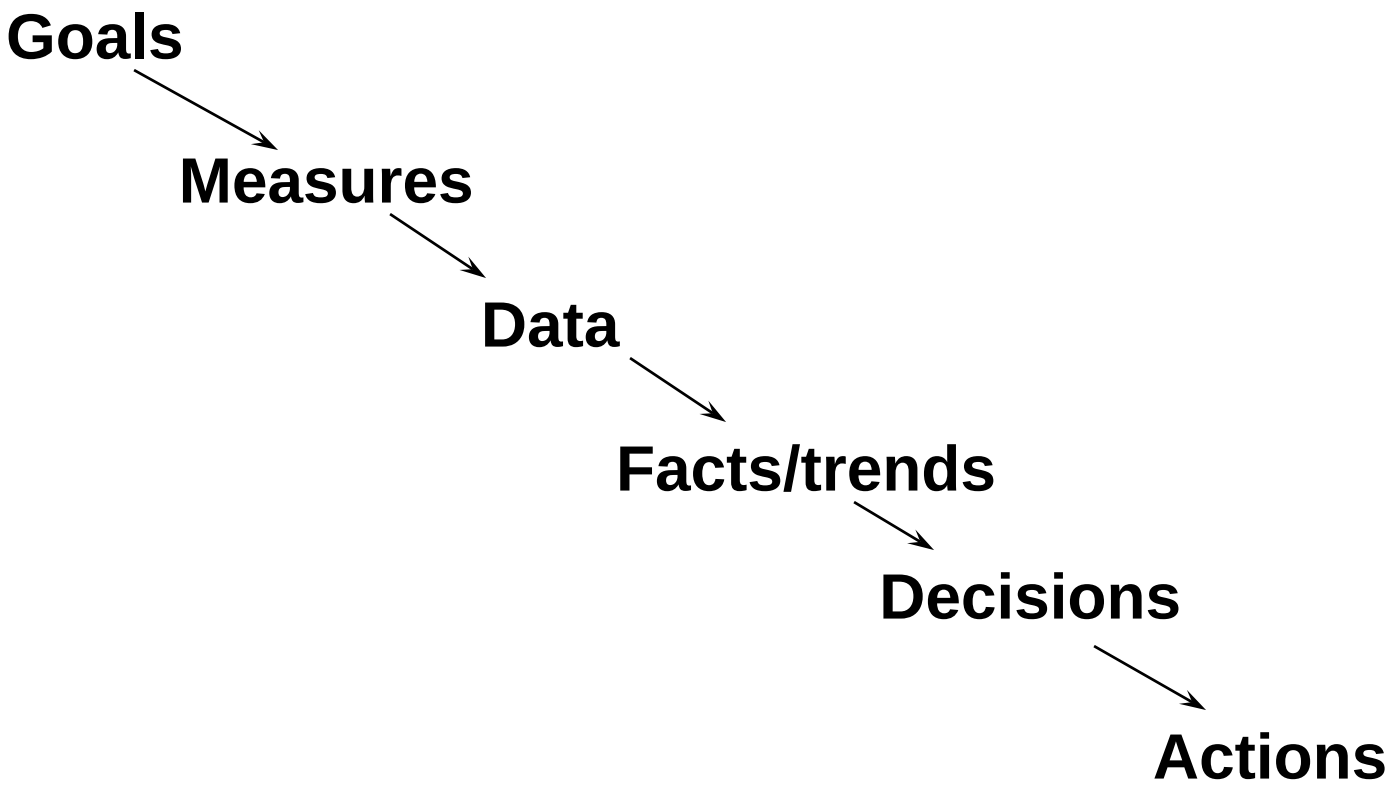
FUNCTION POINTS AND PROGRAM SIZE

Queen Mary University of London

Language	QSM SLOC/FP Data			
	Avg	Median	Low	High
C	97	99	39	333
C++	50	53	25	80
C#	54	59	29	70
Excel	209	191	131	315
HTML	34	40	14	48
J2EE	46	49	15	67
Java	53	53	14	134
JavaScript	47	53	31	63
.NET	57	60	53	60

SOFTWARE METRICS: FROM GOALS TO ACTIONS

Queen Mary University of London



GOAL QUESTION METRIC (GQM)

Queen Mary University of London

- There should be a clearly-defined need for every measurement.
- From the goals, generate questions whose answers will tell you if the goals are met.
- From the questions, suggest measurements that can help to answer the questions.

THE LEVELS IN GQM

Conceptual level (goal) :

A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment.

Operational level (question) :

A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

Quantitative level (metric) :

A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

THE METRICS PLAN

WHY metrics can address the goal

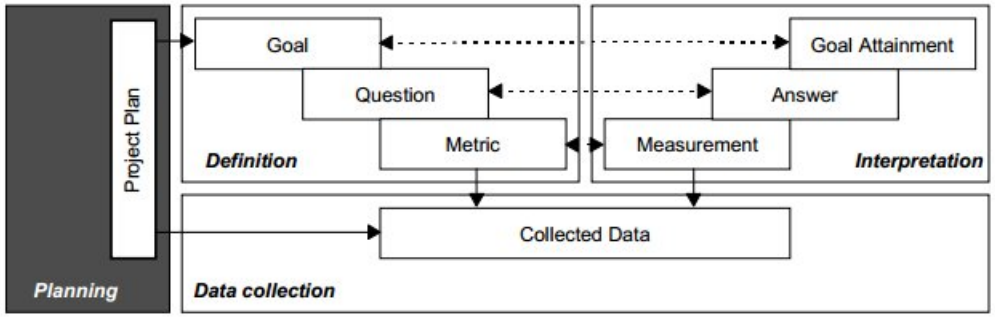
WHAT metrics will be collected, how they will be defined, and how they will be analysed

WHO will do the collecting, who will do the analysing, and who will see the results

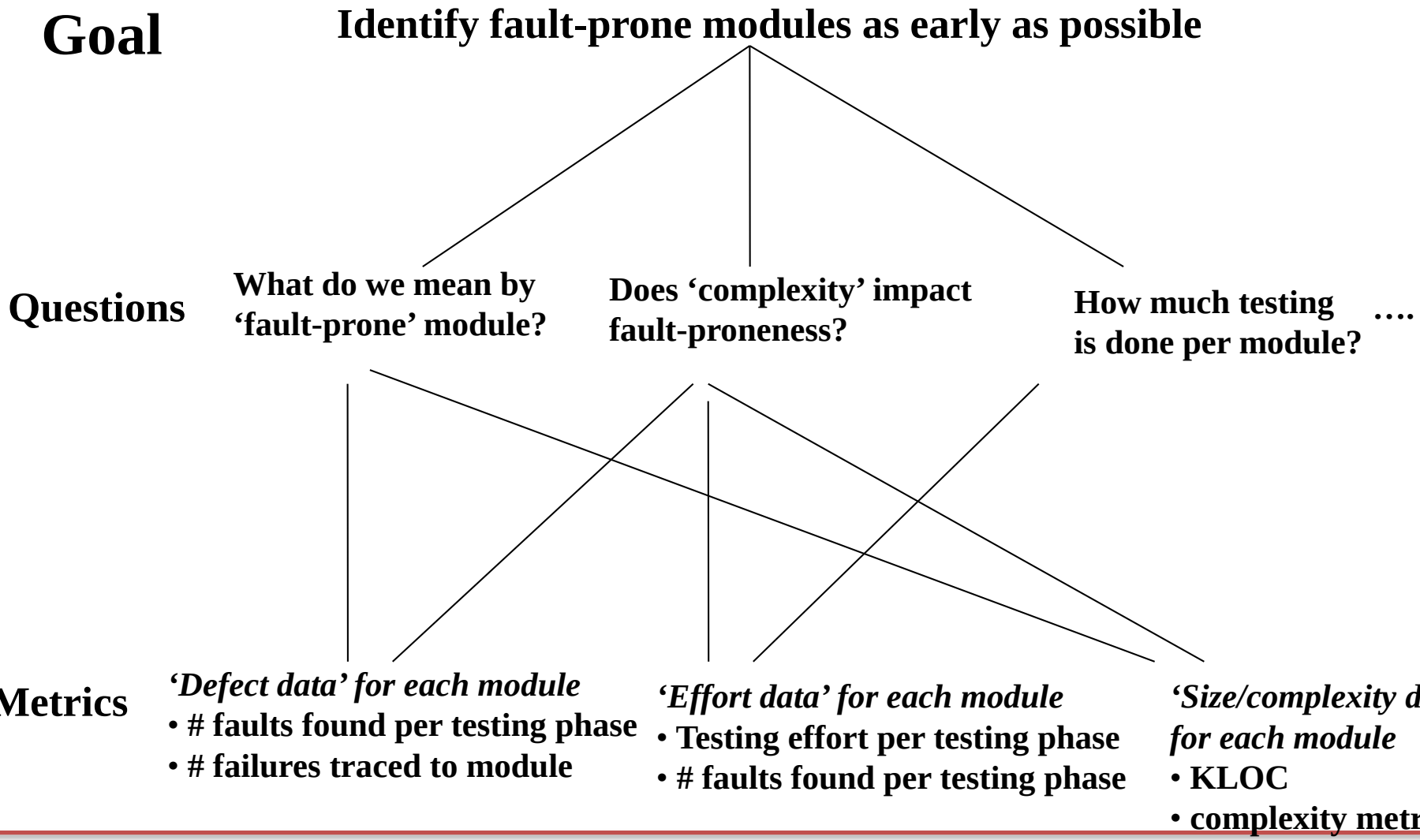
HOW it will be done - what tools, techniques and practices will be used to support metrics collection and analysis

WHEN in the process and how often the metrics will be collected and analysed

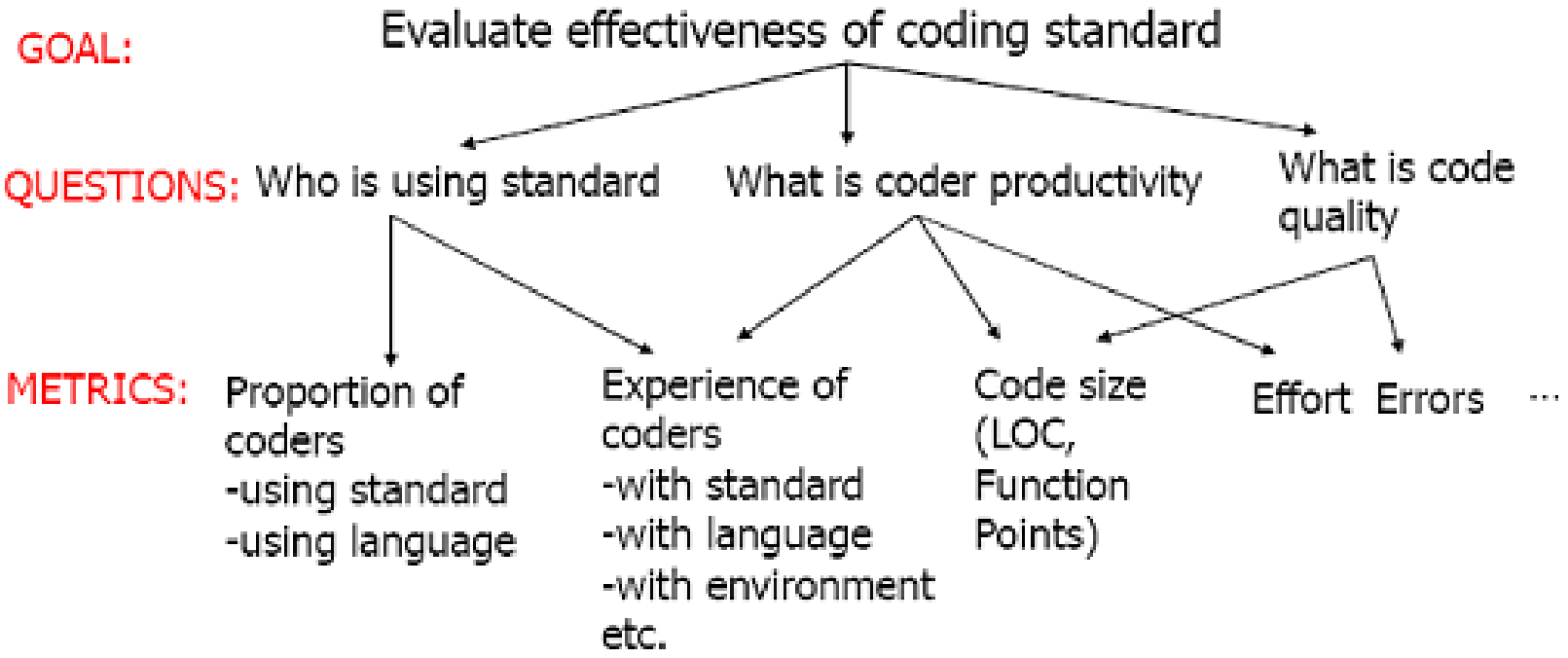
WHERE the data will be stored



GQM Example


Goal
Identify fault-prone modules as early as possible
Questions
What do we mean by 'fault-prone' module?
Does 'complexity' impact fault-proneness?
How much testing is done per module?
....
Metrics
'Defect data' for each module
• # faults found per testing phase
• # failures traced to module
'Effort data' for each module
• Testing effort per testing phase
• # faults found per testing phase
'Size/complexity data for each module
• KLOC
• complexity metrics

GQM Example


GOAL:
Evaluate effectiveness of coding standard
QUESTIONS:
Who is using standard
What is coder productivity
What is code quality
METRICS:
Proportion of coders
-using standard
-using language
Experience of coders
-with standard
-with language
-with environment
etc.
Code size (LOC, Function Points)
Effort
Errors
…

SUMMARY

- Software metrics' driven by two objectives:
  - cost/effort estimation
  - quality assessment and prediction
- All common metrics traceable to above objectives
- It is easy to extract code metrics, but they need to fit a pre-defined plan