FACULDADE ENGENHARIA
Departamento de Informática

Getting Started

(adapted from lecture notes of the CSCI 3060U - Software Quality Assurance unit, J.S. Bradbury, J.R. Cordy, 2018)

What is Software?

"computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system"

-IEEE

1. Computer program (the "code")
2. Procedures
3. Documentation
4. Data

FACULDADE ENGENHARIA

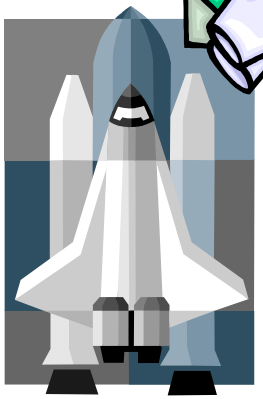
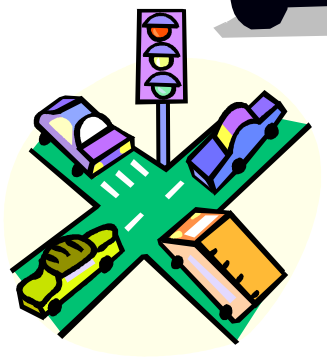
Departamento de Informática

Software is a Skin that Surrounds Our Civilization







What is Software Engineering?

"the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"

-IEEE

The Term "Bug"

The Term "Bug"

- Bug is used informally

- Sometimes speakers mean fault, sometimes error, sometimes failure … often the speaker doesn't know what it means!

Software Faults, Errors & Failures

Software Fault

A static defect in the software

Software Failure

External, incorrect behavior with respect to the requirements or other description of the expected behavior

Software Error

An incorrect internal state that is the manifestation of some fault

Software Faults, Errors & Failures

A practical example:

- A patient gives a doctor a list of symptoms - **Failures**
- The doctor tries to diagnose the root cause, the ailment (disease) – **Fault**
- The doctor may look for anomalous internal conditions (high blood pressure, irregular heartbeat, bacteria in the blood stream) - **Errors**

Software Faults, Errors & Failures

A concrete example:

```
public static int numZero (int[] x) {
  // Effects: if x == null throw NullPointerException
  //    else return the number of occurrences of 0 in x
  int count = 0;
  for (int i = 1; i < x.length; i++)
  {
    if (x[i] == 0)
    {
      count++;
    }
  }
  return count;
}
```

The **fault** in this program is that it starts looking for zeroes at index 1 instead of index 0, for example [0, 7, 2] results in a **failure**, and also in a **error** (evaluates to 0 instead to 1).


WARNING
CHALLENGES
AHEAD

Testing in the 21st Century

- More safety critical, real-time software
- Embedded software is ubiquitous
- Enterprise applications means bigger programs, more users
- Security is now all about software faults – Secure software is reliable software
- The web offers a new deployment platform
  - Very competitive and very available to more users
  - Web apps are distributed


CAUTION
EPIC
FAILURE

Epic Failures

- NASA's Mars lander (1999): crashed due to a units integration fault
- Ariane 5 explosion (1996): an exception-handling fault forced self destruct on maiden flight in 1996 (64-bit to 16-bit conversion: about 370 million $ lost)
- Northeast Blackout (2003): 508 generating units and 256 power plants shut down affected arounf 50 million people in 8 US states and Canada. The alarm system in the energy management system failed due to a software error.

What Causes Software Errors?

1. Faulty definition of requirements
   - Erroneous requirement definitions
   - Absence of important requirements
   - Incomplete requirements
   - Unnecessary requirements included
2. Client-developer communication failures
   - Misunderstanding of client requirements presented in writing, orally, …
   - Misunderstanding of client responses to design problems

What Causes Software Errors?

3. Deliberate deviations from software requirements
   - Reuse of existing software components from previous projects without complete analysis
   - Functionality omitted due to budge or time constraints
   - “Improvements” to software that are not in requirements

4. Logical design errors
   - Errors in interpreting the requirements into a design (e.g. errors in definitions of boundary conditions, algorithms, reactions to illegal operations,...)

5. Coding errors
   - Errors in interpreting the design document, errors related to incorrect use of programming language constructs, etc.

What Causes Software Errors?

6. Non-compliance with documentation and coding instructions
   - Errors resulting from other team members coordinating with non-complying member's code
   - Errors resulting from individuals trying to understand/maintain/test non-complying member's code

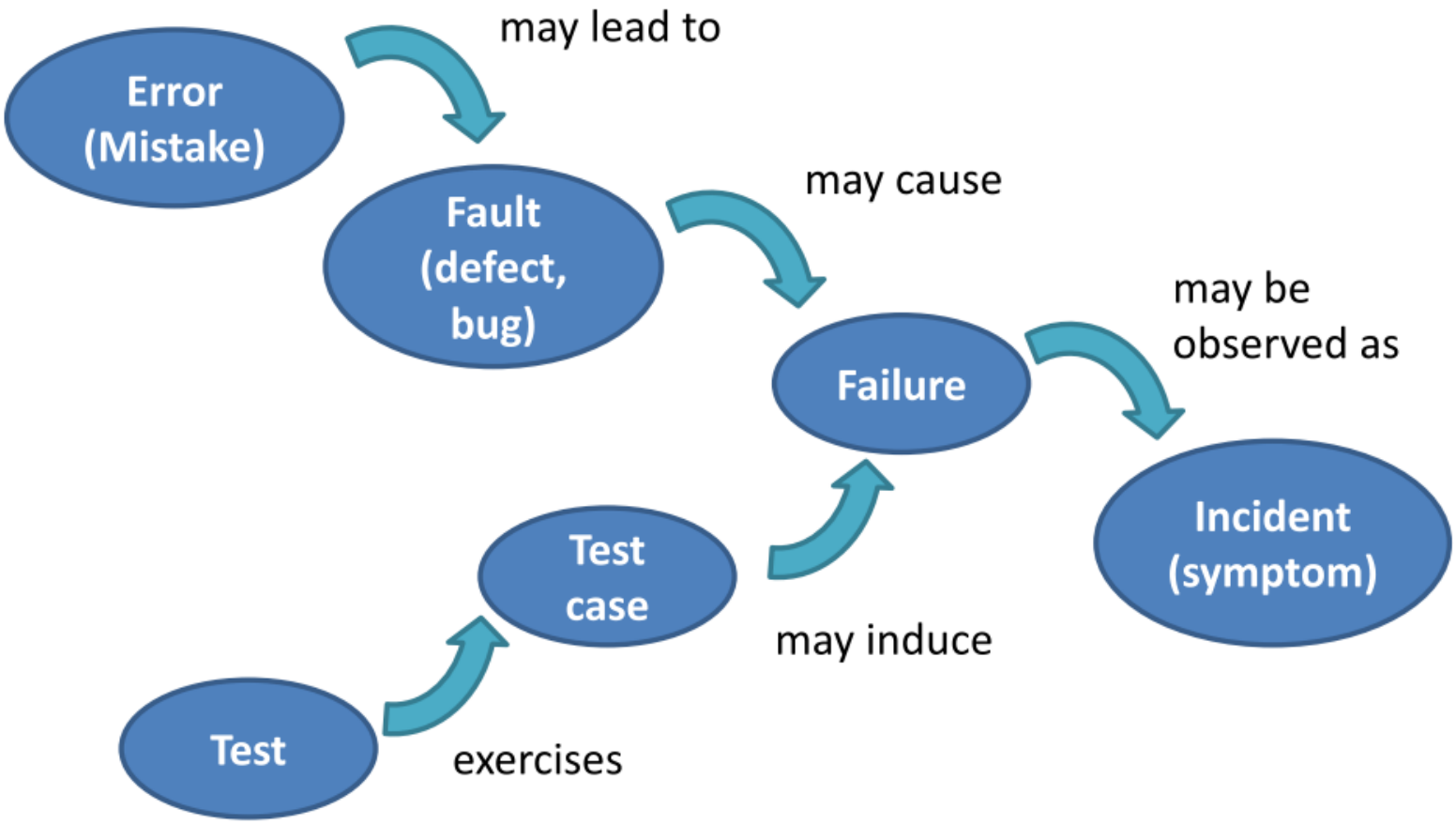
7. Shortcomings of the testing process
   - Incomplete test plan
   - Failure to report all errors/faults resulting from testing
   - Incorrect reporting of errors/faults
   - Incomplete correction of detected errors

What Causes Software Errors?

8. Procedural errors
   - Incorrect procedures related to user activities that occur in the software

9. Documentation errors
   - Errors in the design documents or code comments
   - Errors in user manuals for software

Please Keep in Mind…




VERY LOW
POOR
AVERAGE
GOOD
EXCELLENT
QUALITY

Quality – What is it?

Quality ➔ Not a single idea - many aspects

Popular View

- In everyday life, usually thought of as intangible, can be felt or judged, but not weighed or measured

- *"I know it when I see it"* - implies that it cannot be controlled, managed, or quantified

- Often influenced by perception rather than fact - e.g., a Cadillac may be spoken of as a "quality" car, in spite of the fact that its reliability and repair record is no better than a Chevrolet

Quality – What is it?

Professional View

- In a profession such as software development, there is an ethical imperative to quality
- Quality is not just a marketing and perception issue, it is a moral and legal requirement – we have a professional responsibility associated with the software we create
- Professionals must be able to demonstrate, and to have confidence, that they are using "best practices"
- In practical terms, therefore, product quality must be measurable in some way
- Product quality is spoken of in terms of
  - conformance to requirements - including timeliness, cost
  - fitness for use - does it actually do the job?
  - freedom from errors and failures - is it reliable and robust?
  - customer satisfaction - are users happy with it?

So What is Software Quality?

Software Quality is: [IEEE Definition]

- The degree to which a system, component, or process meets specified requirements

[based on Philip B. Crosby's definition, 1979]

- The degree to which a system, component or process meets customer or user needs or expectations

[based on Joseph M. Juran, 1988]

So What is Software Quality?

Software Quality is: [ISTQB Definition]

- **Quality**: the degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations

- **Software Quality**: The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

So What is Software Quality?

Software Quality

- Software quality is normally spoken of in terms of several different dimensions often called quality parameters

- These can be split (roughly) into two groups

Technical Quality Parameters

- correctness, reliability, capability, performance, maintainability
- these are open to objective measures and technical solutions

User Quality Parameters

- usability, installability, documentation, availability
- these often require subjective analysis and nontechnical solutions

Technical Quality Parameters

- Correctness - lack of bugs and defects
  - measured in terms of defect rate (# bugs per line of code)
- Reliability - does not fail or crash often
  - measured in terms of failure rate (#failures per hour)
- Capability - does all that is required
  - measured in terms of requirements coverage (% of required operations implemented)
- Maintainability - is easy to change and adapt to new requirements
  - measured in terms of change logs (time and effort required to add a new feature) and impact analysis (#lines affected by a new feature)
- Performance - is fast and small enough
  - measured in terms of speed and space usage (seconds of CPU time, Mb of memory, etc.)

User Quality Parameters

- Usability - is sufficiently convenient for the intended users
  - measured in terms of user satisfaction (% of users happy with interface and ease of use)
- Installability - is convenient and fast to install
  - measured in terms of user satisfaction (#install problems reported per installation)
- Documentation - is well documented
  - measured in terms of user satisfaction (% of users happy with documentation)
- Availability - is easy to access and available when needed
  - measured in terms of user satisfaction (% of users reporting access problems)

Classification of Software Quality Factors

McCall's Factor Model

- Classifies all software requirements into 11 factors.
- The factors are grouped in 3 categories:

1. Product operation factors

   Factors that deal with requirements that directly affect the daily operation of the software.

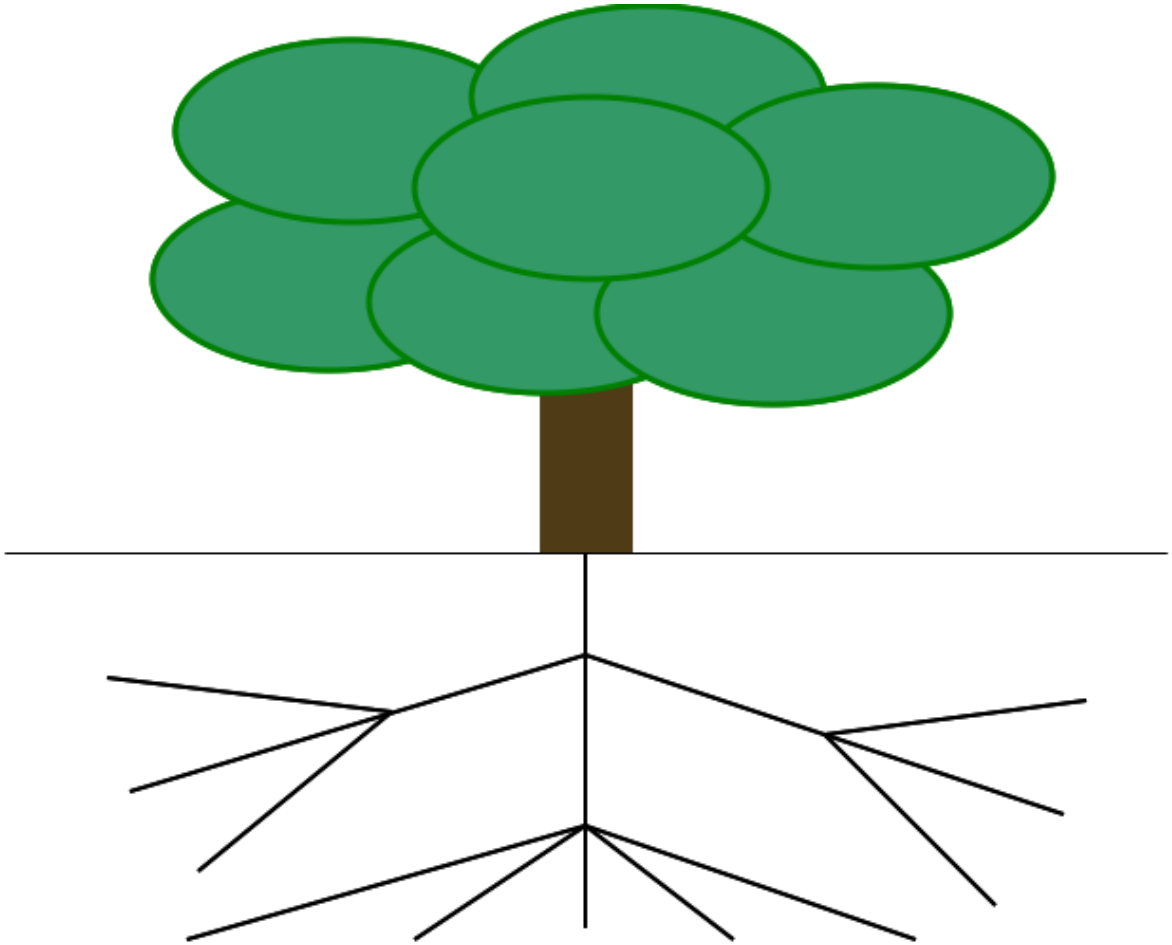
2. Product revision factors

   Factors that deal with requirements that affect software maintenance.

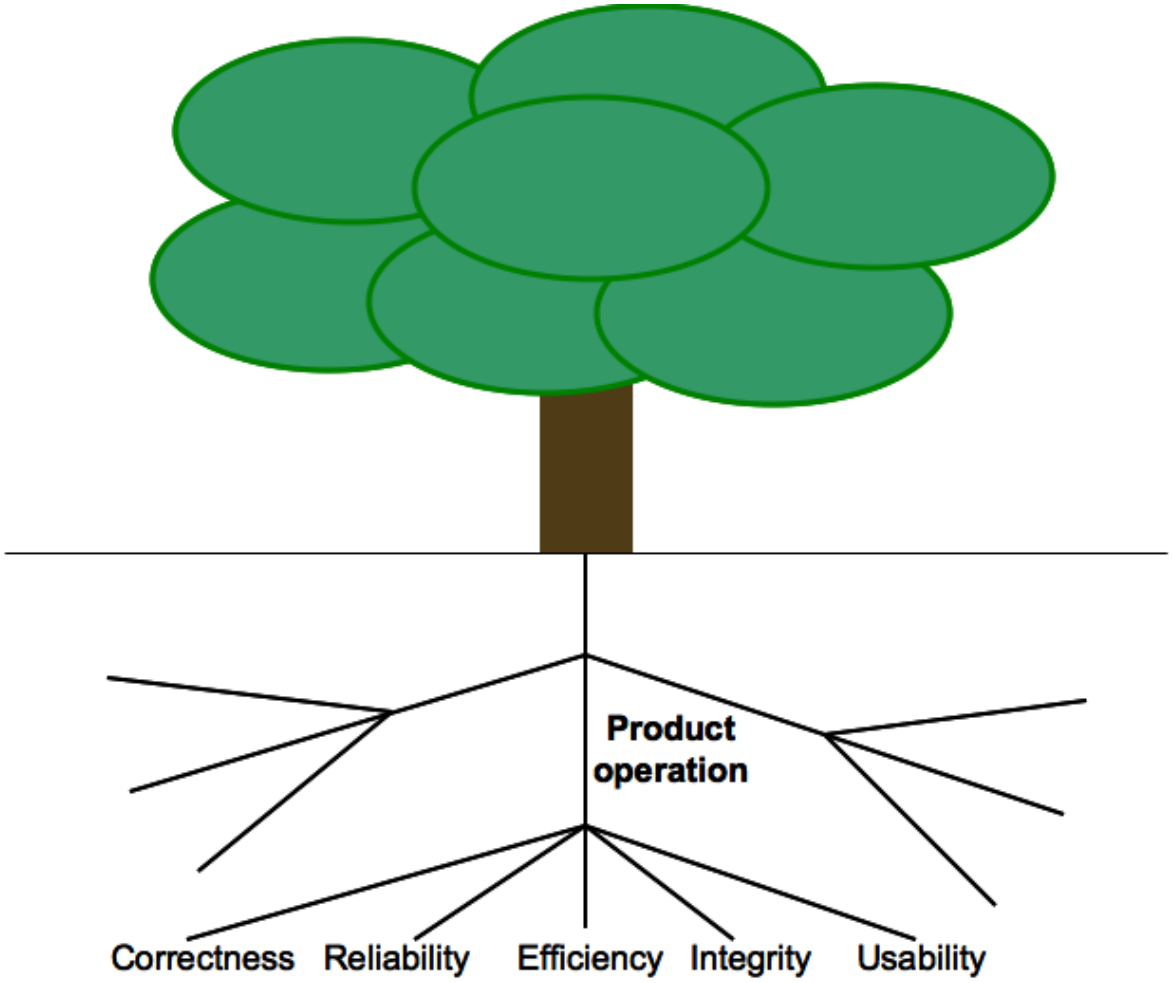
3. Product transition factors

   Factors that deal with requirements that affect the adaptation of software to other platforms, environments, interaction with other software.

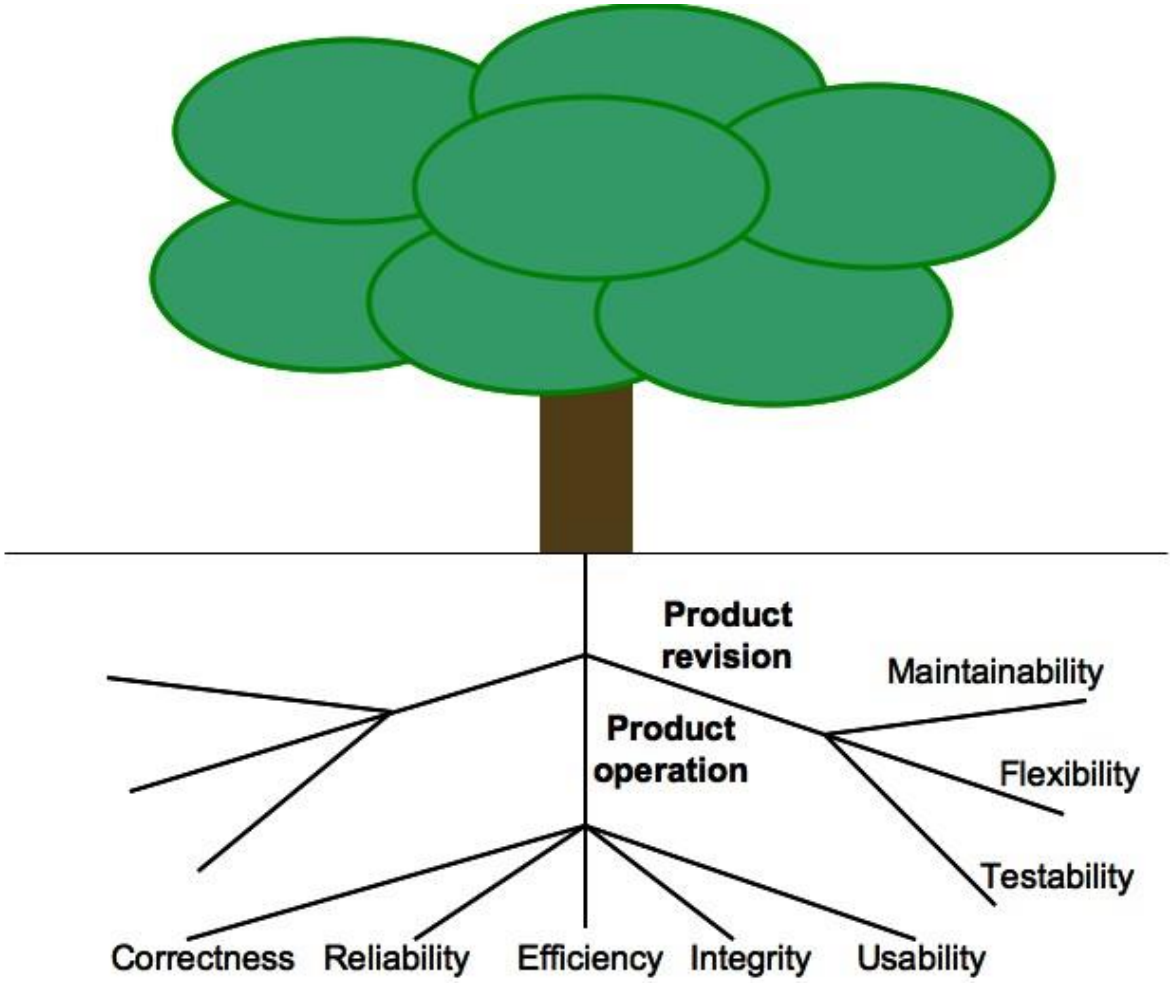
McCall's Factor Model Tree



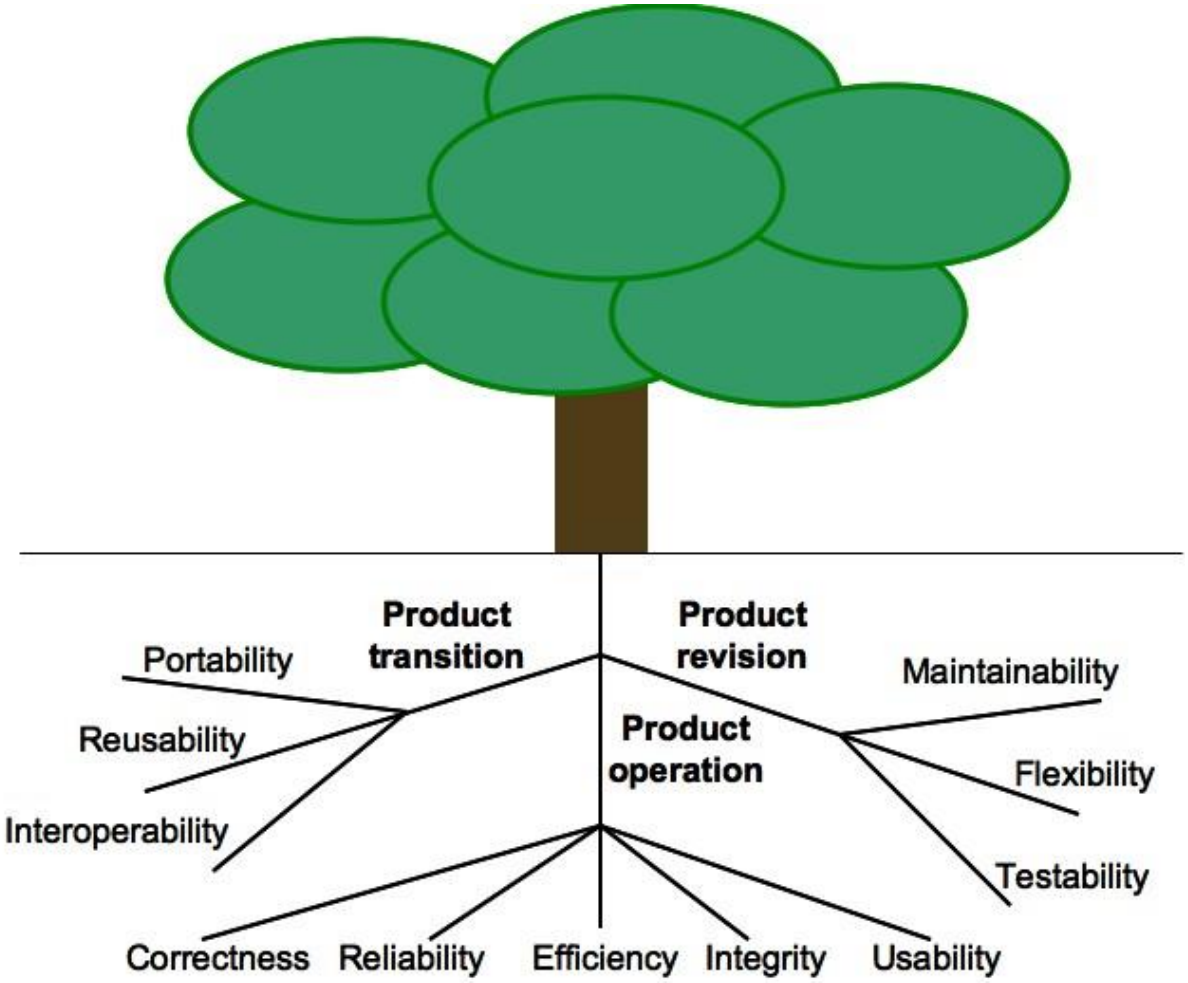
McCall's Factor Model Tree



McCall's Factor Model Tree



McCall's Factor Model Tree



Quality Assurance – What is it?

Achieving Quality

- Product and software quality does not happen by accident, and is not something that can be added on after the fact
- To achieve quality, we must plan for it from the beginning, and continuously monitor it day to day
- This requires discipline
- Methods and disciplines for achieving quality results are the study of Quality Assurance or QA

Quality Assurance – What is it?

Three General Principles of QA

- Know what you are doing
- Know what you should be doing
- Know how to measure the difference

QA Principle 1: Know What You Are Doing

- In the context of software quality, this means continuously understanding what it is you are building, how you are building it and what it currently does

- This requires organization, including having a management structure, reporting policies, regular meetings and reviews, frequent test runs, and so on

- We normally address this by following a software process with regular milestones, planning, scheduling, reporting and tracking procedures

QA Principle 2: Know What You Should be Doing

- In the context of software quality, this means having explicit requirements and specifications

- These must be continuously updated and tracked as part of the software development and evolution cycle

- We normally address this by requirements and use-case analysis, explicit acceptance tests with expected results, explicit prototypes, frequent user feedback

- Particular procedures and methods for this are usually part of our software process

QA Principle 3: Know How to Measure the Difference

- In the context of software quality, this means having explicit measures comparing what we are doing to what we should be doing

- Achieved using four complementary methods:
  - Formal Methods - consists of using mathematical models or methods to verify mathematically specified properties
  - Testing - consists of creating explicit inputs or environments to exercise the software, and measuring its success
  - Inspection - consists of regular human reviews of requirements, design, architecture, schedules and code
  - Metrics - consists of instrumenting code or execution to measure a known set of simple properties related to quality

Formal Methods

- Formal methods include formal verification (proofs of correctness), abstract interpretation (simulated execution in a different semantic domain, e.g., data kind rather than value), state modelling (simulated execution using a mathematical model to keep track of state transitions), and other mathematical methods

- Traditionally, use of formal methods requires mathematically sophisticated programmers, and is necessarily a slow and careful process, and very expensive

- In the past, formal methods have been used directly in software quality assurance in a small (but important) fraction of systems
  - Primarily safety critical systems such as onboard flight control systems, …

- The vast majority (over 99%) of software quality assurance uses testing, inspection and metrics instead of formal methods

Testing

- Testing includes a wide range of methods based on the idea of running the software through a set of example inputs or situations and validating the results

- Includes methods based on requirements (acceptance testing), specification and design (functionality and interface testing), history (regression testing), code structure (path testing), and many more…

Inspection

- Inspection includes methods based on a human review of the software artifacts
- Includes methods based on requirements reviews, design reviews, scheduling and planning reviews, code walkthroughs, and so on
- Helps discover potential problems before they arise in practice

Metrics

- Software metrics includes methods based on using tools to count the use of features or structures in the code or other software artifacts, and compare them to standards

- Includes methods based on code size (number of source lines), code complexity (number of parameters, decisions, function points, modules or methods), structural complexity (number or depth of calls or transactions), design complexity, and so on

- Helps expose anomalous or undesirable properties that may reduce reliability and maintainability

Achieving Software Quality

Standards and Disciplines

- Because of the importance of quality in software production and the relatively bad track record of the past, many standards and disciplines have been developed to enforce quality practice

- Examples are Total Quality Management (TQM), the Capability Maturity Model (CMM), the Personal Software Process (PSP), the Microsoft Shared Development Process (SDP), the Rational Unified Process (RUP), ISO 9000, and **ISO 25010**

Software Quality Control

- Software Quality Control (SQC) is a set of activities for ensuring quality in software products.

- SQC is limited to the Review/Testing phases of the Software Development Life Cycle and the goal is to ensure that the products meet specifications/requirements.

Differences between Software Quality Assurance and Software Quality Control

Criteria	Software Quality Assurance (SQA)	Software Quality Control (SQC)
Definition	SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products). The activities establish and evaluate the processes that produce products.	SQC is a set of activities for ensuring quality in software products. The activities focus on identifying defects in the actual products produced.
Focus	Process focused	Product focused
Orientation	Prevention oriented	Detection oriented
Breadth	Organization wide	Product/project specific
Scope	Relates to all products that will ever be created by a process	Relates to specific product
Activities	• Process Definition and Implementation • Audits • Training	• Reviews • Testing

Software Testing Myths

Myth	Fact
Quality Control = Testing.	Testing is just one component of software quality control, which includes other activities such as Reviews.
The objective of Testing is to ensure a 100% defect- free product.	The objective of testing is to uncover as many defects as possible while ensuring that the software meets the requirements. Identifying and getting rid of all defects is impossible.
Testing is easy.	Testing can be difficult and challenging (sometimes, even more so than coding).
Anyone can test.	Testing is a rigorous discipline and requires many kinds of skills.
Automated testing eliminates the need for manual testing.	100% test automation cannot be achieved. Manual Testing, to some level, is always necessary.


HOW
WHY
WHERE
WHEN
WHERE
WHY
WHERE
WHO
WHEN
WHAT
WHAT
WHY
WHEN
WHAT
WHO
WHEN
WHERE
WHY
HOW