Testing and Analysis Principles

Learning objectives

- Understand the basic principles undelying A&T techniques
- Grasp the motivations and applicability of the main principles

Main A&T Principles

- General engineering principles:
  - **Partition**: divide and conquer
  - **Visibility**: making information accessible
  - **Feedback**: tuning the development process
- Specific A&T principles:
  - **Sensitivity**: better to fail every time than sometimes
  - **Redundancy**: making intentions explicit
  - **Restriction**: making the problem easier

Sensitivity: better to fail every time than sometimes

- Consistency helps:
  - a test selection criterion works better if every selected test provides the same result, i.e., if the program fails with one of the selected tests, it fails with all of them (reliable criteria)
  - run time deadlock analysis works better if it is machine independent, i.e., if the program deadlocks when analyzed on one machine, it deadlocks on every machine

Sensitivity example

- The procedure stringCopy is sensitive: It is guaranteed to fail in an observable way if the source string is too long
- Run-time array bounds checking in many programming languages (including Java) is an example of the sensitivity principle applied at the language level.

```
1  /**
2  * Worse than broken: Are you feeling lucky?
3  */
4
5  #include <assert.h>
6
7     char before[ ] = "=Before=";
8     char middle[ ] = "Middle";
9     char after[ ] = "=After=";
10
11 void  show() {
12    printf("%s\n%s\n%s\n", before, middle, after);
13 }
14
15 void  stringCopy(char *target, const char *source, int howBig);
16
17 int  main(int argc, char *argv) {
18    show();
19    strcpy(middle, "Muddled");     /* Fault, but may not fail */
20    show();
21    strncpy(middle, "Muddled", sizeof(middle)); /* Fault, may not fail */
22    show();
23    stringCopy(middle, "Muddled",sizeof(middle)); /* Guaranteed to fail */
24    show();
25 }
26
27 /* Sensitive version of strncpy; can be counted on to fail
28 * in an observable way EVERY time the source is too large
29 * for the target, unlike the standard strncpy or strcpy.
30 */
31 void  stringCopy(char *target, const char *source, int howBig) {
32    assert(strlen(source) < howBig);
33    strcpy(target, source);
34 }
```

Redundancy: making intentions explicit

- Redundant checks can increase the capabilities of catching specific faults early or more efficiently.
  - Static type checking is redundant with respect to dynamic type checking, but it can reveal many type mismatches earlier and more efficiently.
  - Validation of requirement specifications is redundant with respect to validation of the final software, but can reveal errors earlier and more efficiently.
  - Testing and proof of properties are redundant, but are often used together to increase confidence

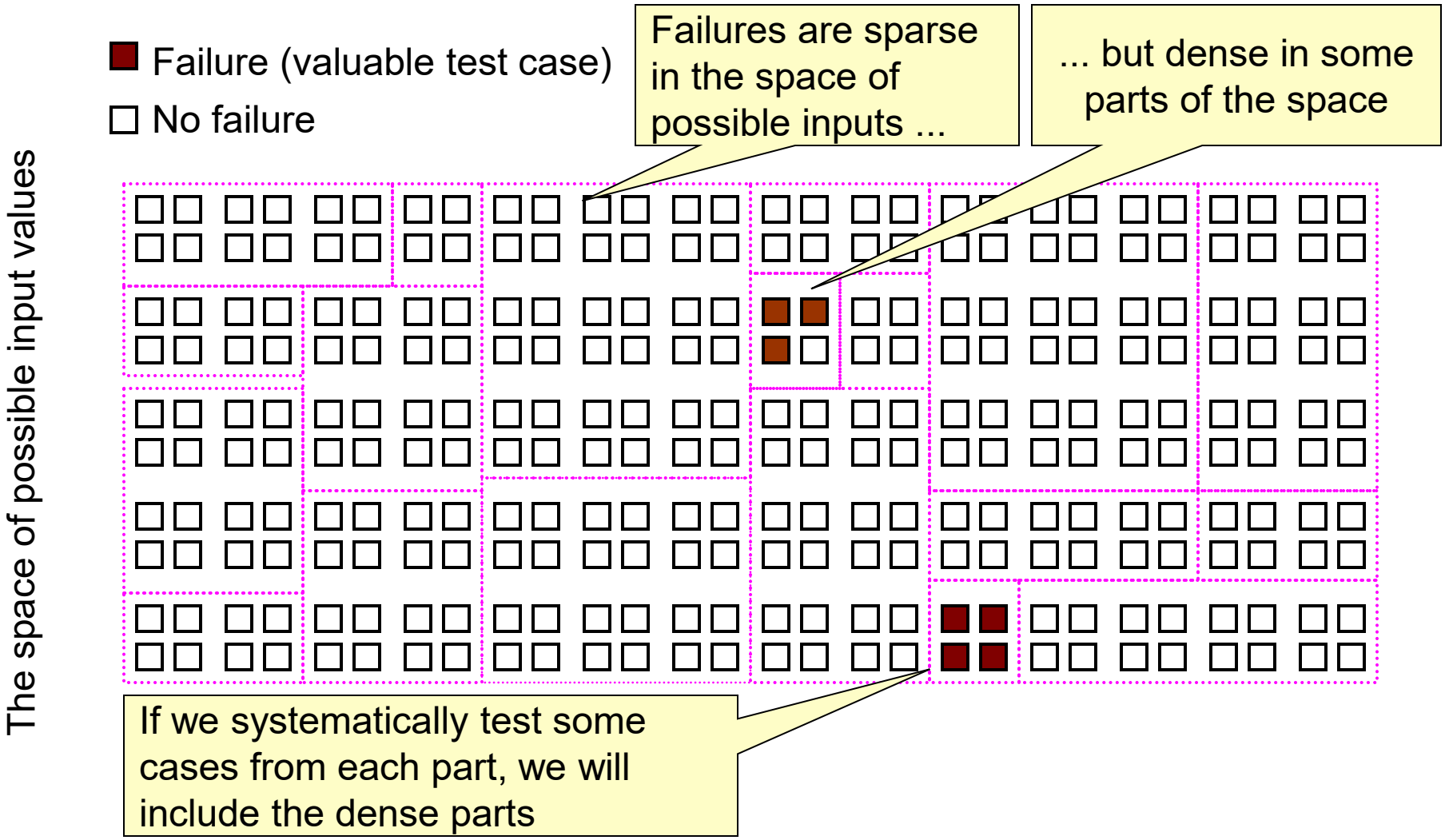
Restriction: making the problem easier

- Suitable restrictions can reduce hard (unsolvable) problems to simpler (solvable) problems
  - It is impossible (in general) to show that pointers are used correctly, but the simple Java requirement that pointers are initialized before use is simple to enforce.
  - it is impossible (in general) to show that type errors do not occur at run-time in a dynamically typed language, but statically typed languages impose stronger restrictions that are easily checkable.

Partition: divide and conquer

- Hard testing and verification problems can be handled by suitably partitioning the input space:
  - both structural and functional test selection criteria identify suitable partitions of code or specifications (partitions drive the sampling of the input space)
  - verification techniques fold the input space according to specific characteristics, grouping homogeneous data together and determining partitions

Partitioning (ideal situation)



Visibility: Judging status

- The ability to measure progress or status against goals
  - X visibility = ability to judge how we are doing on X, e.g., schedule visibility = “Are we ahead or behind schedule,” quality visibility = “Does quality meet our objectives?”
  - Involves setting goals that can be assessed at each stage of development
    - The biggest challenge is **early assessment**, e.g., assessing specifications and design with respect to product quality
- Related to observability
  - Example: Choosing a simple or standard internal data format to facilitate unit testing

Feedback: tuning the development process

- Learning from experience: Each project provides information to improve the next
- Examples
  - Checklists are built on the basis of errors revealed in the past
  - Error taxonomies can help in building better test selection criteria
  - Design guidelines can avoid common pitfalls

⇒ Software Analytics https://seanalytics.github.io/

Summary

- The discipline of test and analysis is characterized by 6 main principles:
  - Sensitivity: better to fail every time than sometimes
  - Redundancy: making intentions explicit
  - Restriction: making the problem easier
  - Partition: divide and conquer
  - Visibility: making information accessible
  - Feedback: tuning the development process
- They can be used to understand advantages and limits of different approaches and compare different techniques

Exit Quiz

Indicate which principles guided the following choices:

1. Use an externally readable format also for internal files, when possible.
2. Collect and analyze data about faults revealed and removed from the code.
3. Separate test and debugging activities; that is, separate the design and execution of test cases to reveal failures (test) from the localization and removal of the corresponding faults (debugging).
4. Distinguish test case design from execution.
5. Produce complete fault reports.
6. Use information from test case design to improve requirements and design

specifications.

7. Provide interfaces for fully inspecting the internal state of a class.

Test and Analysis Activities within a Software Process

Learning objectives

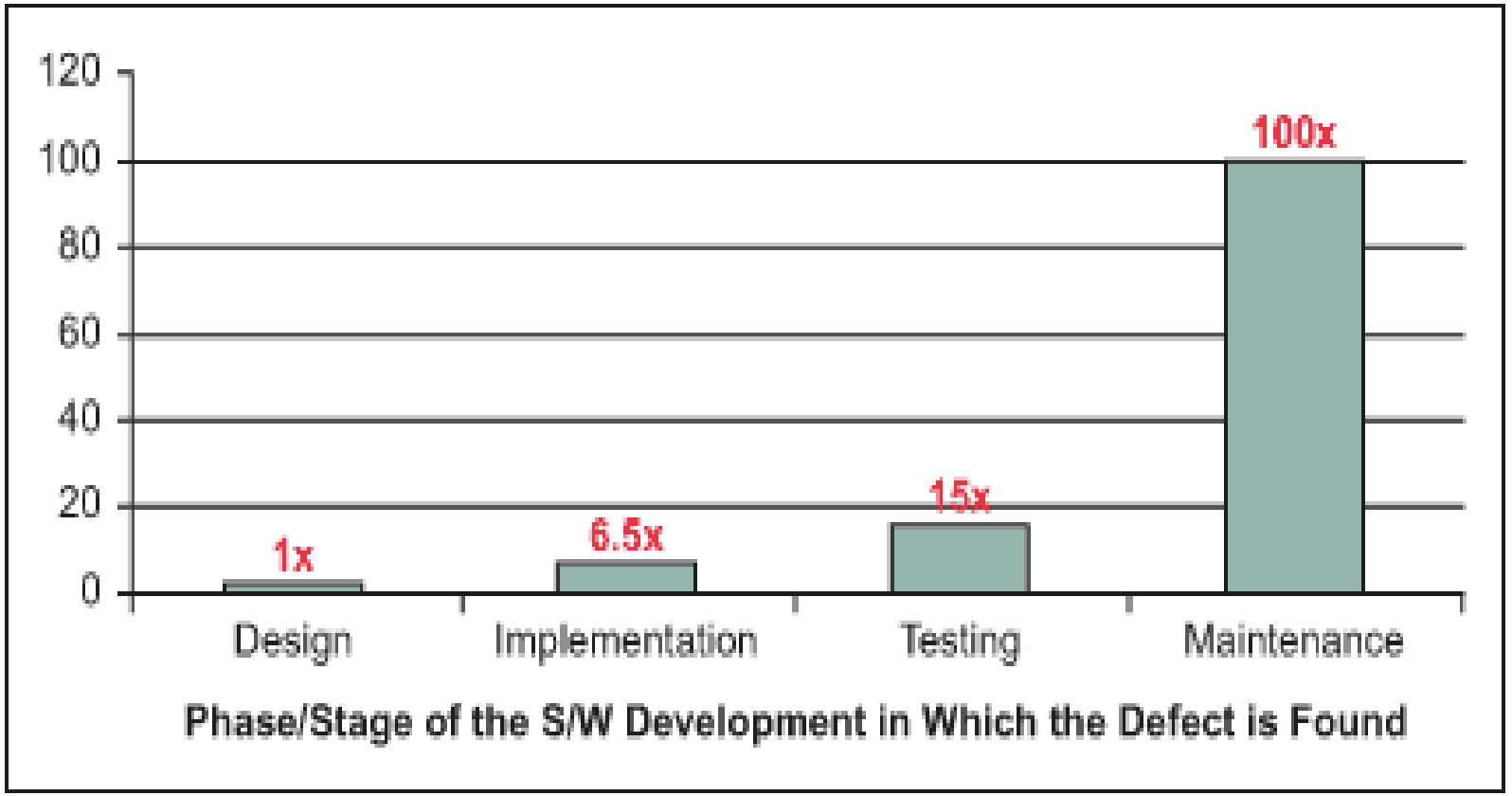
- Understand the role of quality is the development process
- Build an overall picture of the quality process
- Identify the main characteristics of a quality process
  - visibility
  - anticipation of activities
  - feedback

Software Qualities and Process

- Qualities **cannot be added** after development
  - Quality results from a set of inter-dependent activities
  - Analysis and testing are crucial but far from sufficient.
- Testing is not a phase, but a **lifecycle**
  - Testing and analysis activities **occur from early** in requirements engineering through delivery and subsequent evolution.
  - Quality **depends on every part** of the software process

- An essential feature of software processes is that software test and analysis is **thoroughly integrated** and not an afterthought

Cost of defects …



- Cost of correcting an error in requirement specifications increases as we move through lifecycle phases

The Quality Process

- Quality process: set of activities and responsibilities
  - **focused** primarily on ensuring **adequate dependability**
  - In contrast to other processes concerned with, e.g., project schedule or product usability
- The quality process provides a framework for
  - selecting and arranging activities
  - considering interactions and trade-offs with other important goals.

Interactions and tradeoffs

example

high dependability vs. time to market

- Mass market products:
  - better to achieve a reasonably high degree of dependability on a tight schedule than to achieve ultra-high dependability on a much longer schedule

- Critical medical devices:
  - better to achieve ultra-high dependability on a much longer schedule than a reasonably high degree of dependability on a tight schedule

Properties of the Quality Process

- **Completeness**: Appropriate activities are planned to detect **each important class** of faults.
- **Timeliness**: Faults are detected at a point of high leverage (as **early** as possible)
- **Cost-effectiveness**: Activities are chosen depending on cost and effectiveness
  - cost must be considered over the whole development cycle and product life
  - the dominant factor is usually the cost of repeating an activity through many change cycles.

Planning and Monitoring

- The quality process
  - **Balances** several activities across the whole development process
  - Selects and arranges them to be as **cost-effective** as possible
  - Improves **early visibility**
- Quality goals can be achieved only through careful planning
- Planning is integral to the quality process

Process Visibility

- A process is visible to the extent that one can answer the question
  - How does our **progress** compare to our plan?
  - Example: Are we on schedule? How far ahead or behind?
- The quality process has not achieved adequate visibility if one cannot gain strong **confidence** in the quality of the software system **before** it reaches final testing
  - quality activities are usually placed as **early** as possible
    - design test cases at the earliest opportunity (not ``just in time")
    - uses analysis techniques on software artifacts produced before actual code.
  - motivates the use of **"proxy"** measures
    - Ex: **the number of faults** in design or code is not a true measure of reliability, but we may count faults discovered in design inspections as an **early indicator** of potential quality problems

A&T Strategy

Details in Ch. 21 of textbook

- Planning involves the development of both *strategies* and (detailed) *plans*
- A&T Strategies: Company- or project-wide standards that must be satisfied
  - procedures required, e.g., for obtaining quality certificates
  - techniques and tools that must be used
  - documents that must be produced

A&T Plan

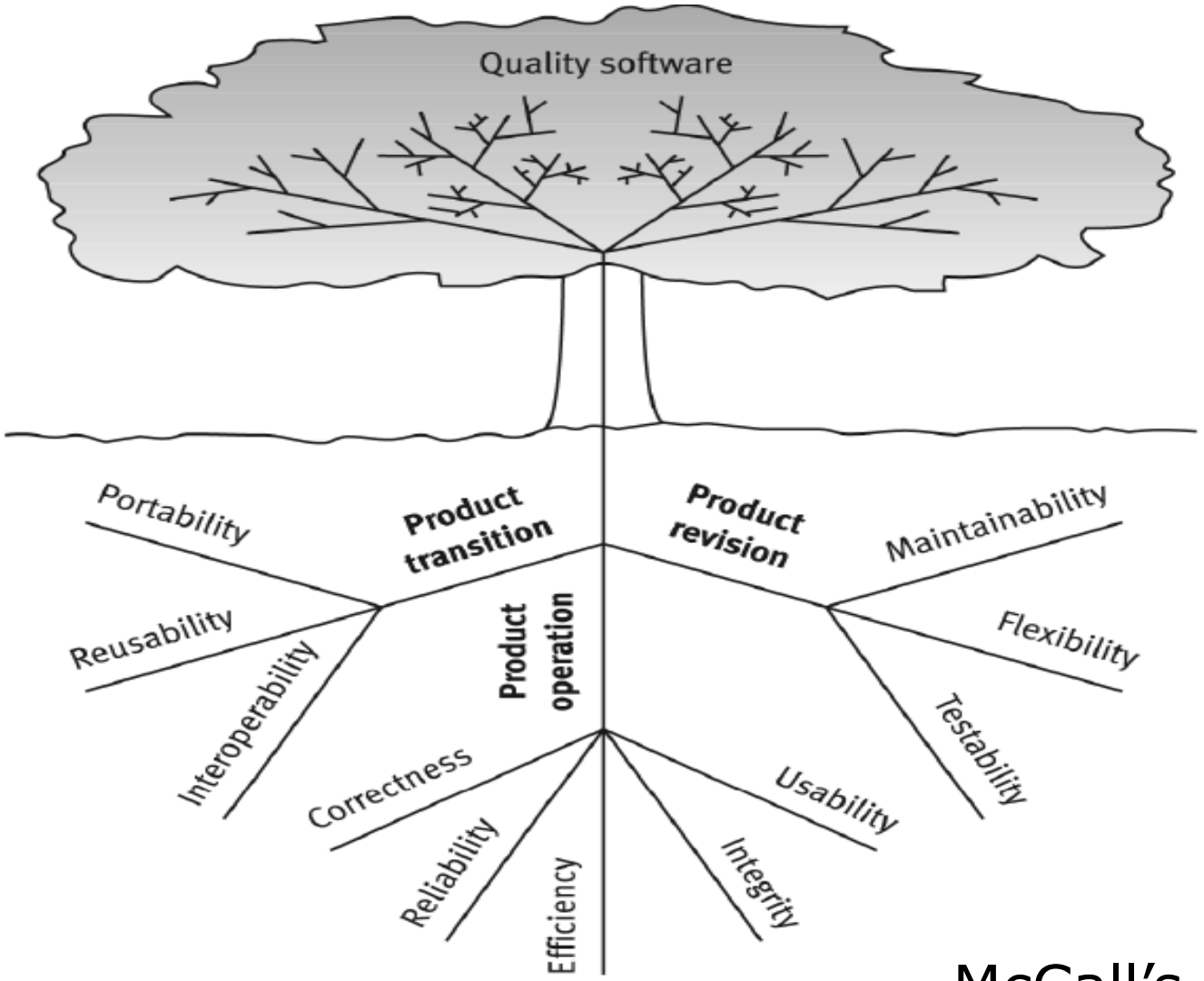
Details in Ch. 24 of textbook

- A comprehensive description of the quality process that includes:
  - objectives and scope of A&T activities
  - documents and other items that must be available
  - items to be tested
  - features to be tested and not to be tested
  - analysis and test activities
  - staff involved in A&T
  - constraints
  - pass and fail criteria
  - schedule
  - deliverables
  - hardware and software requirements
  - risks and contingencies

Quality Goals

- Process qualities (visibility,....)
- Product qualities
  - internal qualities (maintainability, reusability, traceability....)
  - external qualities
    - usefulness qualities:
      - usability, performance, security, portability, interoperability
    - dependability
      - correctness, reliability, safety, robustness

Characteristics of software quality

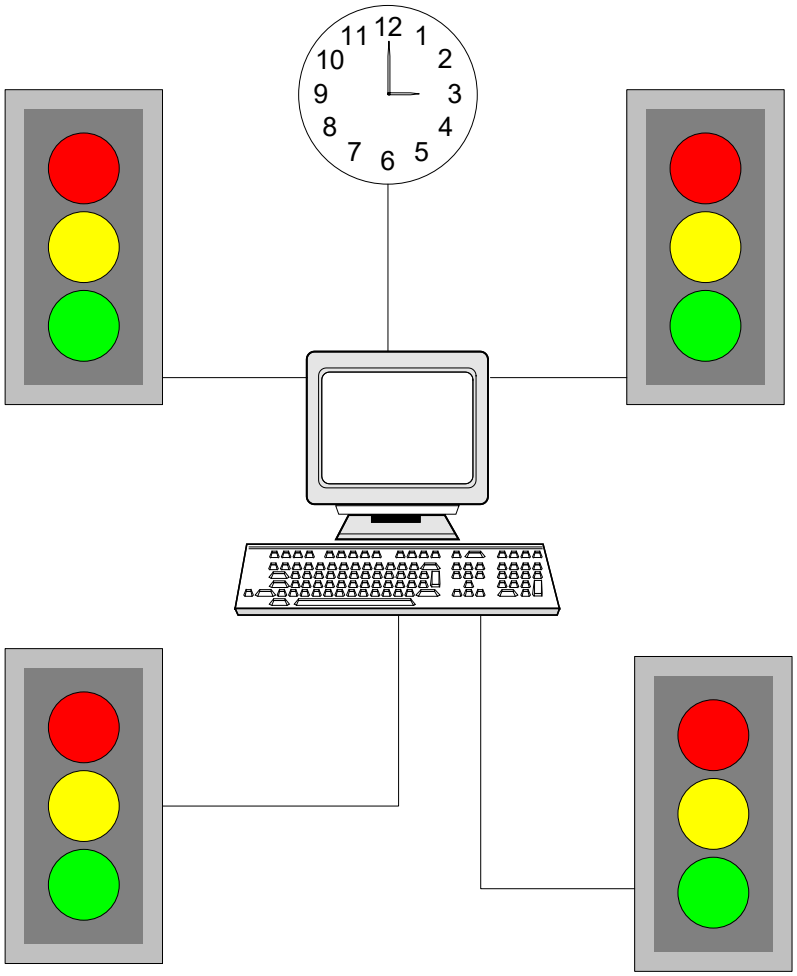


McCall's Quality Factor Tree

Dependability Qualities

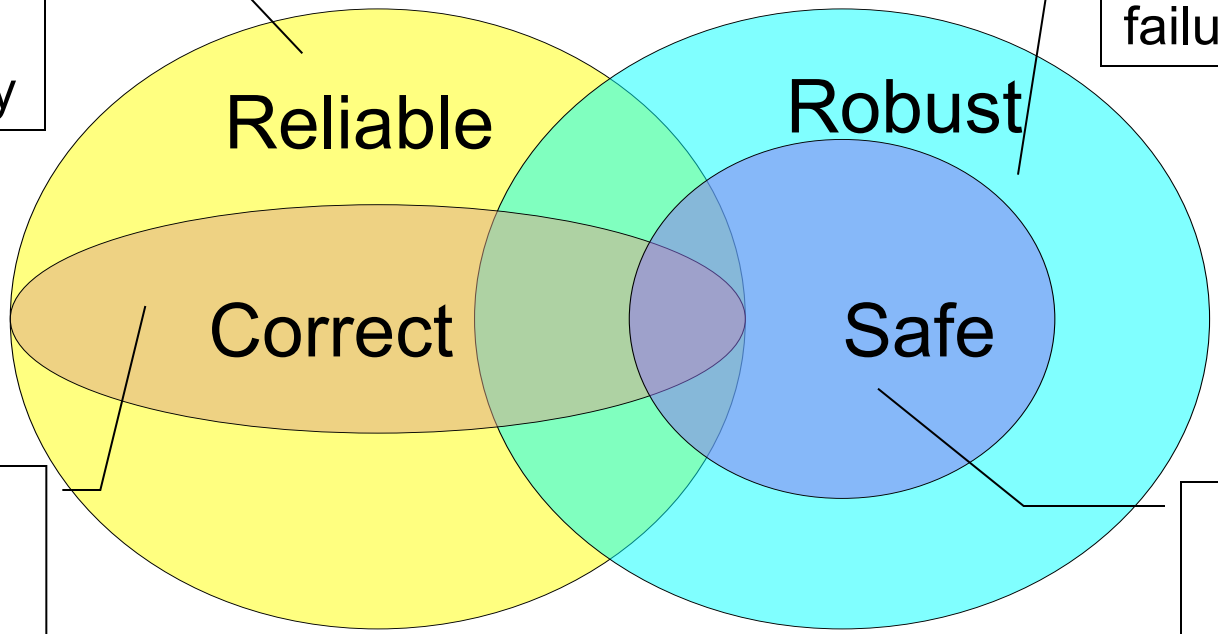
- Correctness:
  - A program is correct if it is consistent with its specification
    - seldom practical for non-trivial systems
- Reliability:
  - likelihood of correct function for some ``unit" of behavior
    - relative to a specification and usage profile
    - statistical approximation to correctness (100% reliable = correct)
- Safety:
  - preventing hazards
- Robustness
  - acceptable (degraded) behavior under extreme conditions

Example of Dependability Qualities



- **Correctness, reliability:** let traffic pass according to correct pattern and central scheduling
- **Robustness, safety:** Provide degraded function when possible; never signal conflicting greens.
  - Blinking red / blinking yellow is better than no lights; no lights is better than conflicting greens

Relation among Dependability Qualites



reliable but not correct: failures occur rarely

robust but not safe: catastrophic failures can occur

correct but not safe or robust: the specification is inadequate

safe but not correct: annoying failures can occur

Analysis

- analysis includes
  - manual inspection techniques
  - automated analyses
- can be applied at any development stage
- particularly well suited at the early stages of specifications a design

Inspection

- can be applied to essentially any document
  - requirements statements
  - architectural and detailed design documents
  - test plans and test cases
  - program source code
- may also have secondary benefits
  - spreading good practices
  - instilling shared standards of quality.
- takes a considerable amount of time
- re-inspecting a changed component can be expensive
- used primarily
  - where other techniques are inapplicable
  - where other techniques do not provide sufficient coverage

Inspection examples

- Modern code review tools such as Gerrit
- https://www.youtube.com/watch?v=DyAX8ws5OIc

Automatic Static Analysis

- More limited in applicability
  - can be applied to some formal representations of requirements models
  - not to natural language documents
- are selected when available
  - substituting machine cycles for human effort makes them particularly cost-effective.

Automatic Static Analysis examples

- See the demo of this tool https://www.youtube.com/watch?v=CAbyd3sLLCQ

Testing

- Executed late in development
- Start as early as possible
- Early test generation has several advantages
  - Tests generated independently from code, when the specifications are fresh in the mind of analysts
  - The generation of test cases may highlight inconsistencies and incompleteness of the corresponding specifications
  - tests may be used as compendium of the specifications by the programmers

Improving the Process

- Long lasting errors are common
- It is important to structure the process for
  - Identifying the most critical persistent faults
  - tracking them to frequent errors
  - adjusting the development and quality processes to eliminate errors
- Feedback mechanisms are the main ingredient of the quality process for identifying and removing errors

Organizational factors

- Different teams for development and quality?
  - separate development and quality teams is common in large organizations
  - indistinguishable roles is postulated by some methodologies (extreme programming)
- Different roles for development and quality?
  - **test designer** is a specific role in many organizations
  - mobility of people and roles by rotating engineers over development and testing tasks among different projects is a possible option

Example of Allocation of Responsibilities

- Allocating tasks and responsibilites is a complex job: we can allocate
  - **Unit testing**
    - to the **development team** (requires detailed knowledge of the code)
    - but the quality team may control the results (structural coverage)
  - **Integration, system and acceptance testing**
    - to the quality team
    - but the development team may produce scaffolding and oracles
  - **Inspection and walk-through**
    - to mixed teams
  - **Regression testing**
    - to quality and maintenance teams
  - **Process improvement related activities**
    - to external specialists interacting with all teams

Allocation of Responsibilities and rewarding mechanisms: case A

- allocation of responsibilities
  - Development team responsible development measured with LOC (lines of code) per person month
  - Quality team responsible for quality
- possible effect
  - Development team tries to maximize productivity, without considering quality
  - Quality team will not have enough resources for bad quality products
- result
  - product of bad quality and overall project failure

Allocation of Responsibilities and rewarding mechanisms: case B

- allocation of responsibilities
  - Development team responsible for both development and quality control
- possible effect
  - the problem of case A is solved
  - but the team may delay testing for development without leaving enough resources for testing
- result
  - delivery of a not fully tested product and overall project failure

Summary

- Test and Analysis are complex activties that must be sutiably planned and monitored
- A good quality process obeys some basic principles:
  - visibility
  - early activities
  - feedback
- aims at
  - reducing occurrences of faults
  - assessing the product dependability before delivery
  - improving the process

Brainstorming

Identify some correctness, robustness and safety properties of automatic car parking system.

https://youtu.be/cBctth36jik