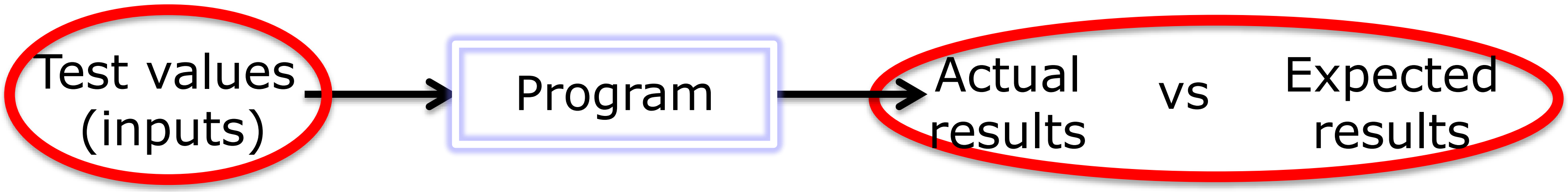
Faults, Errors, Failures & Software Quality

Credit: [Ammann and Offutt, "Introduction to Software Testing"
& "Model-Driven Test Design" (Chapter 2)]
Software Quality Slides from CSC 179/234 Fall 2017

What is Software Testing?

- Testing = process of finding input values to **check** against a software *(focus of this course)*

- Debugging = process of finding a fault given a failure



1. Testing is fundamentally about choosing finite sets of values from the input domain of the software being tested

2. Given the test inputs, compare the actual results with the expected results

Today’s Objective

- Understand the differences between **faults, errors, and failures**
- Understand how **faults, errors, and failures** affect the program
- Understand the **four conditions** that must be satisfied when designing tests
  - **Reachability, Infection, Propagation, Revealability**

    *R*IPR model
- Discuss some potential term projects ideas
- Software Quality (Next lecture!)

Recap: Fault, Error, and Failure

Fault: a static defect in the software's source code

Cause of a problem

Error: An incorrect internal state that is the manifestation of some fault

Erroneous program state caused by execution of the defect

Failure: External, incorrect behavior with respect to the requirements or other descriptions of the expected behavior

Propagation of erroneous state to the program

Example

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

Few questions:

* There is a simple **fault** in **`numZero`**

* Where is the fault location in the source code?

* How would you fix it?

* Can the fault location be **reached**? How does it **corrupt program state**? Does it always corrupt the program state?

* If the program state is corrupted, does **`numZero`** **fail**? How?

Example – Let's Analyze

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

Fault: a defect in source code

`i = 1` [should start searching at 0, not 1]

Error: erroneous program state caused by execution of the defect

`i` becomes 1 [array entry 0 is not ever read!]

Failure: propagation of erroneous state to the program

Happens as long as `arr.length` > 0 and `arr[0] = 0`

Example – Test Cases

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

Fault: `i = 1` [should start searching at 0, not 1]

Test 1: [4, 6, 0], expected 1

Error: **i** is 1, not 0, on the first iteration

Failure: none

Test 2: [0, 4, 6], expected 1

Error: **i** is 1, not 0, error **propagates** to the variable **count**

Failure: **count** is 0 at the return statement

Example – State Representation

```
     public static int numZero (int[] arr)
     {  // Effects: If arr is null throw NullPointerException
        // else return the number of occurrences of 0 in arr
L1      int count = 0;
L2      for (int i = 1; i < arr.length; i++)
L3         if (arr[i] == 0)
L4            count++;
L5      return count;
     }
```

Program State notation `<var`$_1$` = v`$_1$`, …, var`$_n$` = v`$_n$`, PC = program counter>`

Sequence of states in the execution of `numZero({0, 4, 6})`

1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >
2: < arr={0, 4, 6}, `count=0`, PC=[`i=1` (L2)] >
3: < arr={0, 4, 6}, `count=0`, `i=1`, PC=[`i<arr.length` (L2)] >
…
< arr={0, 4, 6}, `count=0`, PC=[`return count;` (L5)] >

Example – Error State

Error state

The first different state in execution in comparison to an execution to the state sequence of what would be the correct program

If the code had i=0 (correct program), the execution of **`numZero({0, 4, 6})`** would be

1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >
2: < arr={0, 4, 6}, `count=0`, PC=[`i=0` (L2)] >
3: < arr={0, 4, 6}, `count=0`, **i=0**, PC=[`i<arr.length` (L2)] >
…

Instead, we have

1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >
2: < arr={0, 4, 6}, `count=0`, PC=[`i=1` (L2)] >
3: < arr={0, 4, 6}, `count=0`, **i=1**, PC=[`i<arr.length` (L2)] >
…

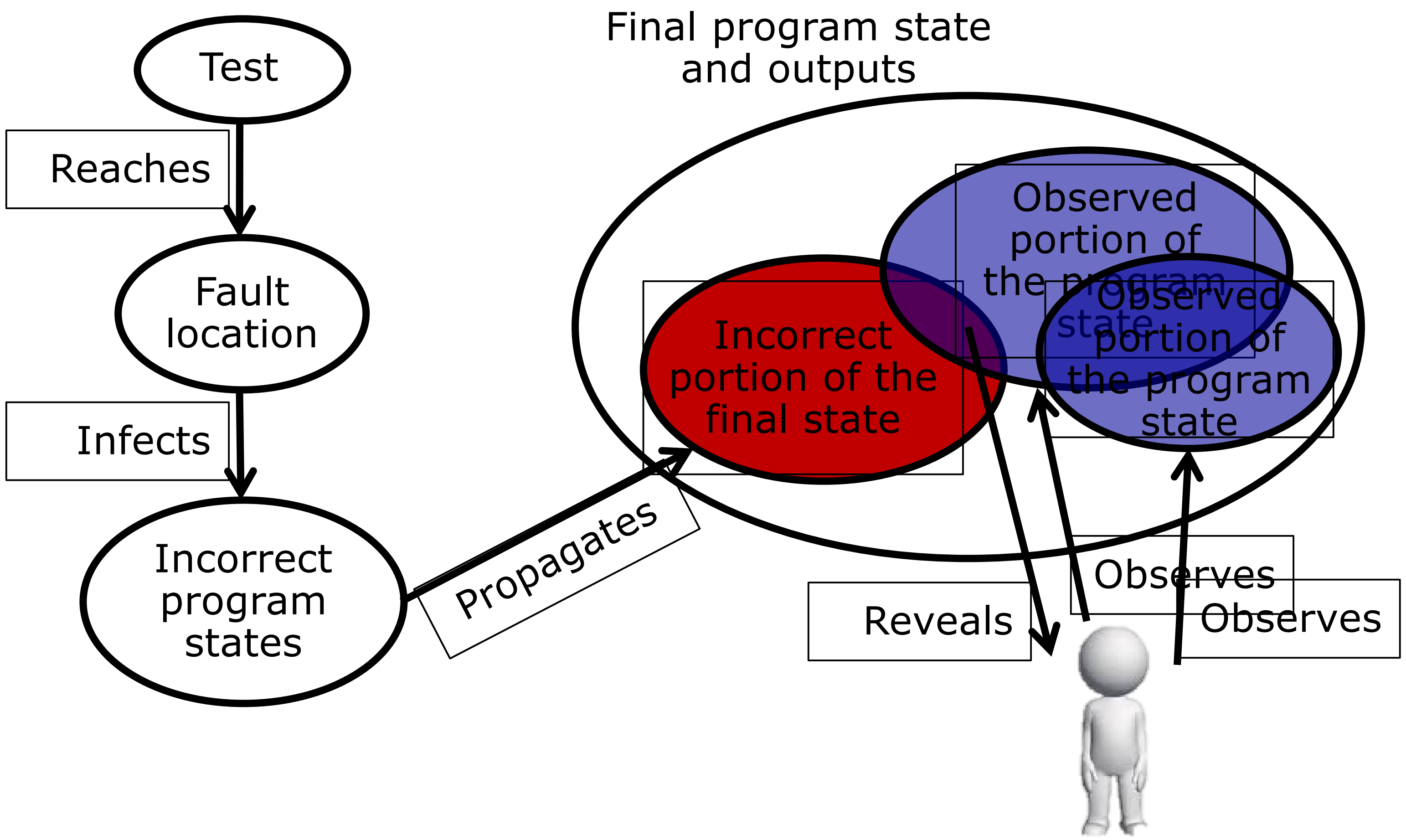
The first error state is immediately after **`i=1`** in line L2

RIPR Model

Four conditions necessary for a failure to be observed

- **R**eachability
  - The **fault** is reached
- **I**nfection
  - Execution of the fault leads to an incorrect program state (**error**)
- **P**ropagation
  - The infected state must cause the program output or final state to be incorrect (**failure**)
- **R**evealability
  - The tester must observe part of the incorrect portion of the program state

RIPR Model


Final program state and outputs
Test
Reaches
Fault location
Infects
Incorrect program states
Propagates
Incorrect portion of the final state
Observed portion of the program state
Observed portion of the program state
Reveals
Observes
Observes

[AO, p.21]

Example – RIPR (Error, No Failure)

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

Reach a fault (i.e., execute the fault)

Cause the program state to be incorrect (i.e., error)

Does not propagate (i.e., no failure)

- One possible test is [4, 6, 0] – now, design some more

How does RIPR model help designing tests?

Example – RIPR (Error, Failure)

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

Cause the program state to be incorrect (i.e., error)

Propagate (i.e., failure)

- One possible test is [0, 4, 6] – now, design some more

How does RIPR model help designing tests?

Wrap-up

- Faults, errors, failures
- Fault location
- Infected state
- RIPR model
- Observability and revealibility

What's Next?

- Software Quality
- Term projects ideas
- Model-Driven Test Design (MDTD)

Questions?

CSc-179: Introduction to Software Testing

Term projects ideas

General Testing Practice: Apply concepts related to course materials

- **Senior Projects (Two groups)**
- **CSC-133 (Assignments 3, One group)**
- **One or Two of the each of the following areas:**
  1. **Static Analysis**
  2. **Web based testing**
  3. **Mobile Computing**
  4. **Model based testing**
  5. **Test-Driven Development (TDD) vs Behavior-Driven Development (BDD)**
- **New research based projects (Two groups)**

Static Analysis

- **Static analysis tools** are generally used by developers as part of the development and component testing process. Here the code is not executed or run but the tool itself is executed, and the source code we are interested in is the input data to the tool.

- **Tutorial:**
- https://en.wikipedia.org/wiki/Static_program_analysis
- What is Static Analysis?

More information:

Experiences Using Static Analysis to Find Bugs

Integrate static analysis into a software development process

Mobile Computing Testing

- An increased in demanding: **Mobile** application **testing** is the process through which applications are **tested** for required quality, functionality, compatibility, usability, performance and other characteristics.

- **Tutorial:**
- https://www.tutorialspoint.com/mobile_testing/index.htm

- **Tools:**
- **APPIUM Tutorial for Android & iOS Mobile Apps Testing**
- http://appium.io/index.html?lang=en

More information:

Mobile Application Testing: A Tutorial

Model Based Testing

- Model based testing is a software testing technique where run time behavior of software under test is checked against predictions made by a model. A model is a description of a system's behavior. Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.

- **Tutorial:**
- Model-based testing
- **Model Based Testing Tutorial: What is, Tools & Example**

- **Tools:**

Tricentis Tosca : The models generated are dynamic and synchronized with the application under test, making test and test portfolio maintenance very easy.

Web Based Testing

- Web application testing, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested (https://www.tutorialspoint.com/software_testing_dictionary/web_application_testing.htm).
- **Tutorial:**
- Web based Testing
- **Tools:**

Selenium (https://www.seleniumhq.org/)

Tricentis Tosca (https://www.tricentis.com/software-testing-tools/)

Research: Automated Debugging

- Automated debugging, where your tool try to automatically isolate the part of the code that causes a bug.

- **Topics**: delta debugging, statistical debugging, ESC/Java
- Are Automated Debugging Techniques Actually Helping Programmers?

Andreas Zeller: Yesterday, My Program Worked. Today, It Does Not. Why?, FSE, 1999Tool: https://www.st.cs.uni-saarland.de/publications/files/zeller-esec-1999.pdf

James A. Jones et al., Visualization of test information to assist fault localization, ICSE, 2002

Flanagan et al., Extended Static Checking for Java, PLDI , 2002 **Ten-Year Most Influential Paper Award**

David Hovemeyer and William Pugh, Finding bugs is easy, OOPSLA, 2004
https://baoz.net/wp-content/2009/02/finding-bugs-is-easy.pdf

Some Sample source codes

- Google Summer Source Code: https://summerofcode.withgoogle.com/archive/2018/projects/
- Sample Android Apps:

  https://github.com/codepath/android_guides/wiki/Sample-Android-Apps
- Sample Hotel booking Projects:

  https://github.com/topics/hotel-booking
- Attendance Management Systems:

  https://github.com/topics/attendance-management-system

Software Quality

What is software ?

- IEEE definition:

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

- Four components of software:
  - — Computer programs (the code)
  - — Procedures/Documentation
  - — Data necessary for operating the software system

What is Quality?

- How good or bad something is
- A high level of value or excellence
- A characteristic or feature that someone or something has: something that can be noticed as a part of a person or thing

Source: http://www.merriam-webster.com/dictionary/quality

Note: Quality is a complex concept – it means different things to different people, and is highly context dependent (Naik & Triphathy – Software testing & quality assurance)

Quality can be defined as.....

- Conformance to requirements
  - Quality is measured against the requirements
- Fitness for the purpose
  - If the purpose of an automobile is to be fast, efficient, comfortable and safe — then that's the definition of a quality automobile. But how you measure safety …?
- Level of satisfaction experiences
  - If you are traveling with an airline company, you are looking for end-to-end experience from calling to make the reservation, check-in, flying, safety, etc. Customers judge experiences by, airline service, atmosphere, safety, food services, etc

IEEE Definition of "Software Quality"

1. The degree to which a system, component, or process meets specified **requirements**.

2. The degree to which a system, component, or process meets customer or user **needs** or **expectations**.

Giving us the following points

The definition gives us the following points:

– Requirements specifications define the basis for high-quality software. A software product that does not follow its specifications is not a high quality product.

– The theory of software engineering defines the framework where high-quality software must be developed. If used software engineering methods are not within the framework, the result is not a high-quality program.

– A high-quality software follows both implicit and explicit software requirements.

– One of the implicit software requirement is that is it well tested.

Base on your experiences, can you name a few causes to Software Failures ?

Some possible causes of Software Errors

1. Faulty requirements definition by the client
2. Client-developer communication failures
3. Deliberate deviations from software requirements
4. Logical design fault
5. User interface and procedure fault
6. Coding defects
7. Non-compliance with documentation and coding instructions
8. Shortcomings of the testing process
9. Documentation errors

Source: Software Quality Concepts & Practice (Daniel Galin)

Causes of Software Failures (Cont)

1. Faulty requirements definition by the client

Examples:
+ Missing or incomplete requirements
+ Ambiguous requirements

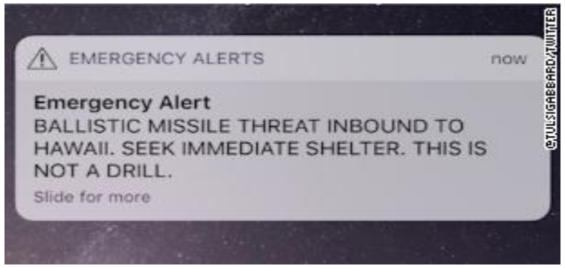
Causes of Software Failures (Cont)

4. Logical design fault

Examples:

+ Definitions that represent software requirements by means of erroneous algorithms.

+ Designing around technology instead of around the user experiences.
See: Here's what went wrong with the Hawaii false alarm (source: edition.cnn.com/2018/01/31/us/hawaii-false-alarm-investigation-findings/index.html)



EMERGENCY ALERTS now

Emergency Alert
BALLISTIC MISSILE THREAT INBOUND TO HAWAII. SEEK IMMEDIATE SHELTER. THIS IS NOT A DRILL.
Slide for more

- didn't have reasonable safeguards in place to prevent human error from resulting in the transmission of a false alert.
- Hawaii's alert origination software didn't differentiate between testing and the live alert environment. Both alerts had the same interface and the same confirmation language, regardless of whether the message was real or a drill "

Causes of Software Failures (Cont)

6. Coding defect

Examples:

+ Logical Defect: (Compiled ok, but it gives incorrect result)

```
float average(float a, float b)
{
    return a + b / 2;     /* should be (a + b) / 2 */
}
```

+ Arithmetic Defect: (Program might aborted at runtime)

  Defect: age/x when x = 0. (x is not initialized!)

Causes of Software Failures (Cont)

8. Short coming of testing process

Some examples:

+**Test planning and scheduling problems** often occur when there is <u>no separate test plan</u>. Test case descriptions are often mistaken for overall test plans. Stop testing too early.

+ **Stakeholder involvement and commitment problems** include having the wrong testing mindset (that the purpose of testing is to show that the software works <u>instead of finding defects</u>.

+ **Test organizational and professionalism problems** include a lack of independence, unclear testing responsibilities, and inadequate testing expertise.

Source: https://insights.sei.cmu.edu/sei_blog/2013/04/common-testing-problems-pitfalls-to-prevent-and-mitigate.html

“Good” requirement document

- The D2L (Desire2Learn) problem:
- — The software fulfills the basic requirements.
- — The software suffers from poor performance in important areas (maintenance, reliability…)
- WHY?

Lack of predefined requirements to cover these important aspects!

- Classify quality requirements into quality factors.

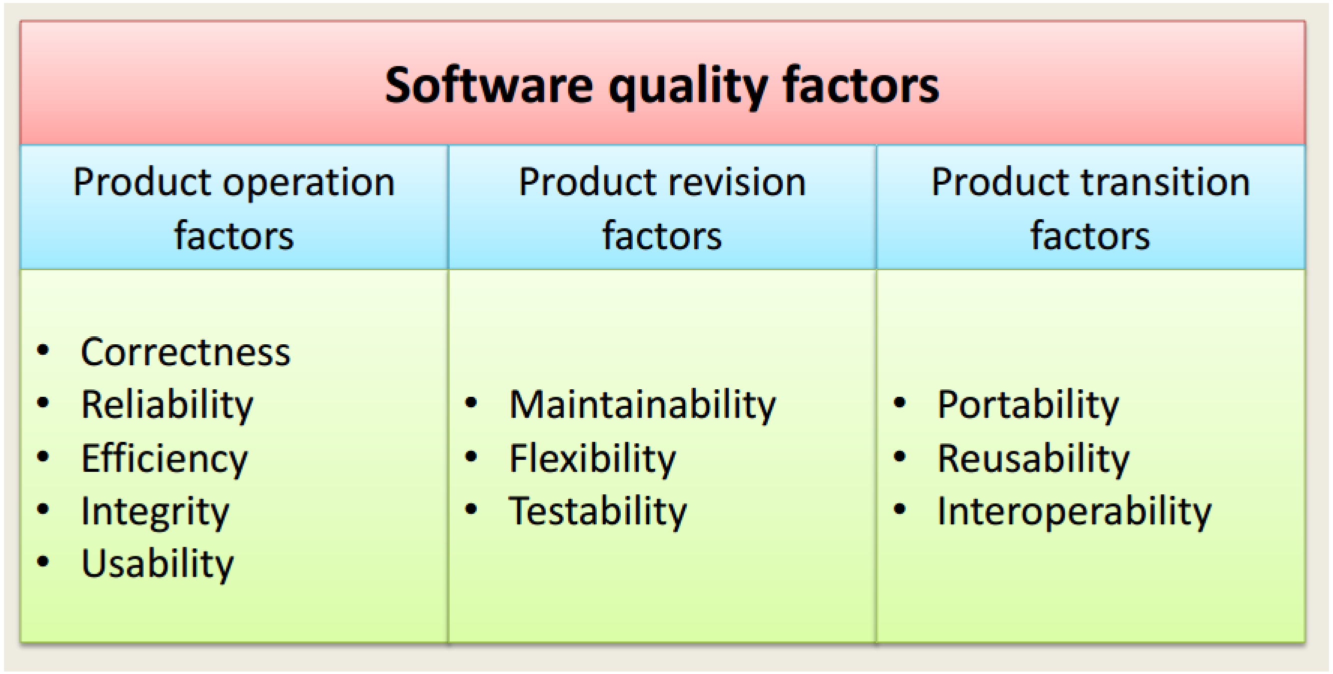
— McCall’s factor model (1977)

Introduction McCall 's Model

- **Jim McCall** (McCall, Richards & Walters, 1977) introduced this model (also known as the General Electrics Model of 1977).
- This model, as well as other contemporary models, originates from the US military (it was developed for the US Air Force, promoted within DoD) and is primarily aimed towards the system developers and the system development process.
- This quality model attempts to bridge the gap between users and developers by focusing on a number of software quality factor that reflect both
  - ✓ users' views
  - ✓ developers' priorities.
- At NASA, the criteria for evaluation of software quality are taken from McCall's software quality factor model.

Source:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.1138&rep=rep1&type=pdf
http://www.dtic.mil/dtic/tr/fulltext/u2/a049014.pdf

Software quality factors (McCall's Quality Model – 1977)

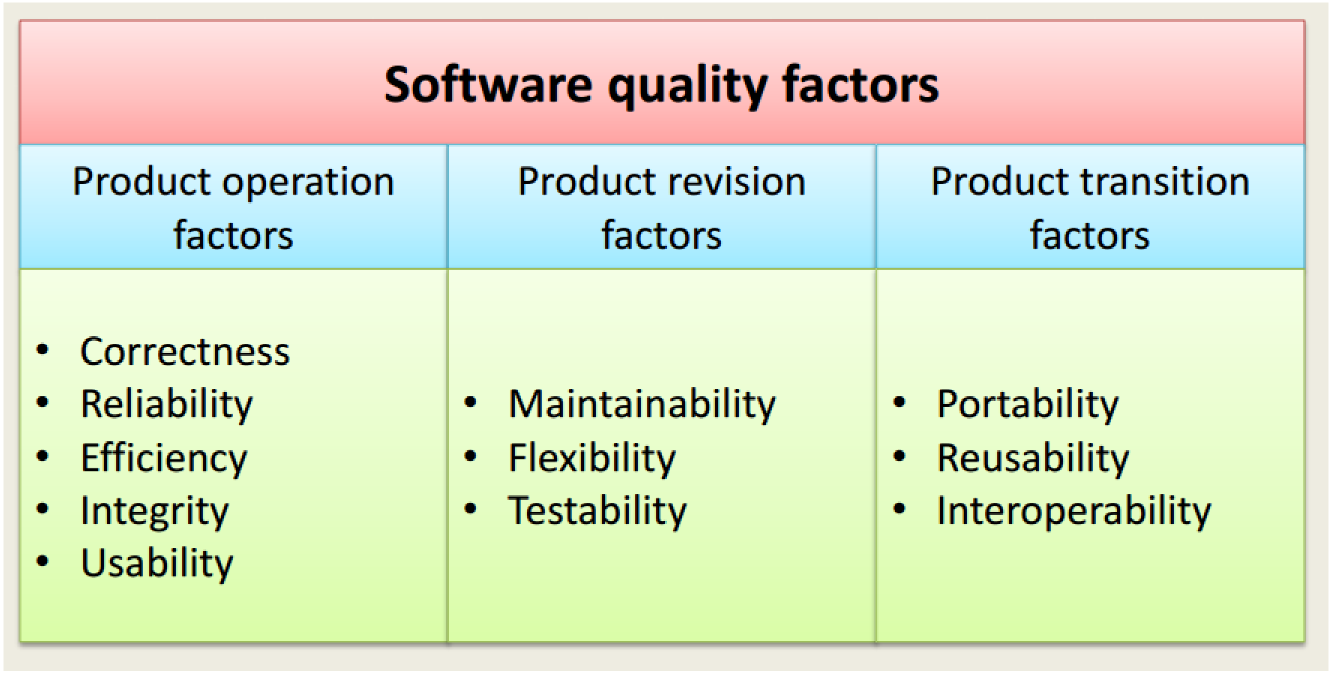


Software quality factors		
Product operation factors	Product revision factors	Product transition factors
• Correctness • Reliability • Efficiency • Integrity • Usability	• Maintainability • Flexibility • Testability	• Portability • Reusability • Interoperability

(Basic operational characteristics) (Ability to change) (Ability to adapt to new environments)

This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.

Software quality factors (McCall's Quality Model – 1977)



Software quality factors		
Product operation factors	Product revision factors	Product transition factors
• Correctness • Reliability • Efficiency • Integrity • Usability	• Maintainability • Flexibility • Testability	• Portability • Reusability • Interoperability

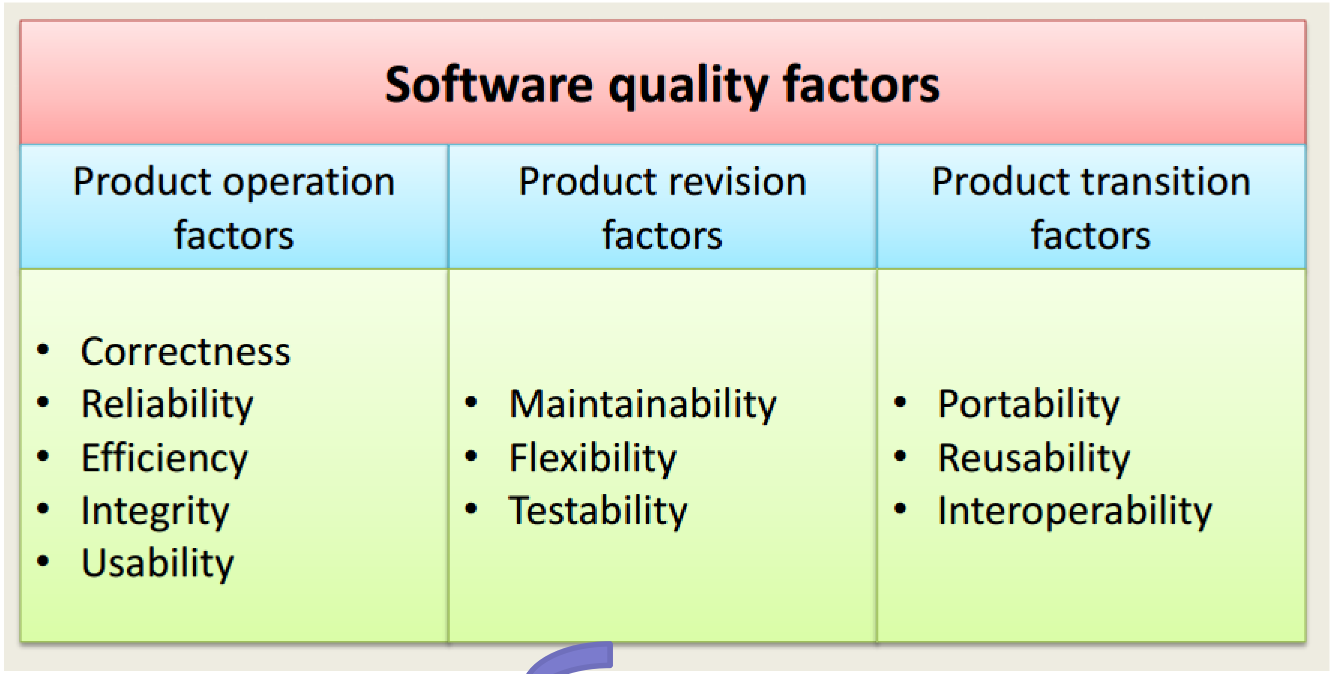
Identifies quality factors that influence the extent to which the software fulfils its specification

Product operation factors

- Correctness: Does it do what customer wants ? (meeting specifications)
  - — output mission, accuracy, completeness of required output
  - — up-to-date, availability of the information
  - — standards for coding and documenting the system
- Reliability: Does it do it accurately all of the time ? (successful performance)
  - — minimize failure rate
- Efficiency: Does it quickly solve the intended problem ? (enough computing resources)
  - — resources needed to perform software function (processing, data storage, communication, battery consumption, etc.)
- Integrity: Is it secure ? (access limitations to people)
  - — software system security, access rights
- Usability: Can I run it ? (efforts in learning/operating)
  - — ability to learn, perform required task

Again: These factors influences the extent to which the software fulfils its specification

Software quality factors (McCall's Quality Model – 1977)



Software quality factors		
Product operation factors	Product revision factors	Product transition factors
• Correctness • Reliability • Efficiency • Integrity • Usability	• Maintainability • Flexibility • Testability	• Portability • Reusability • Interoperability

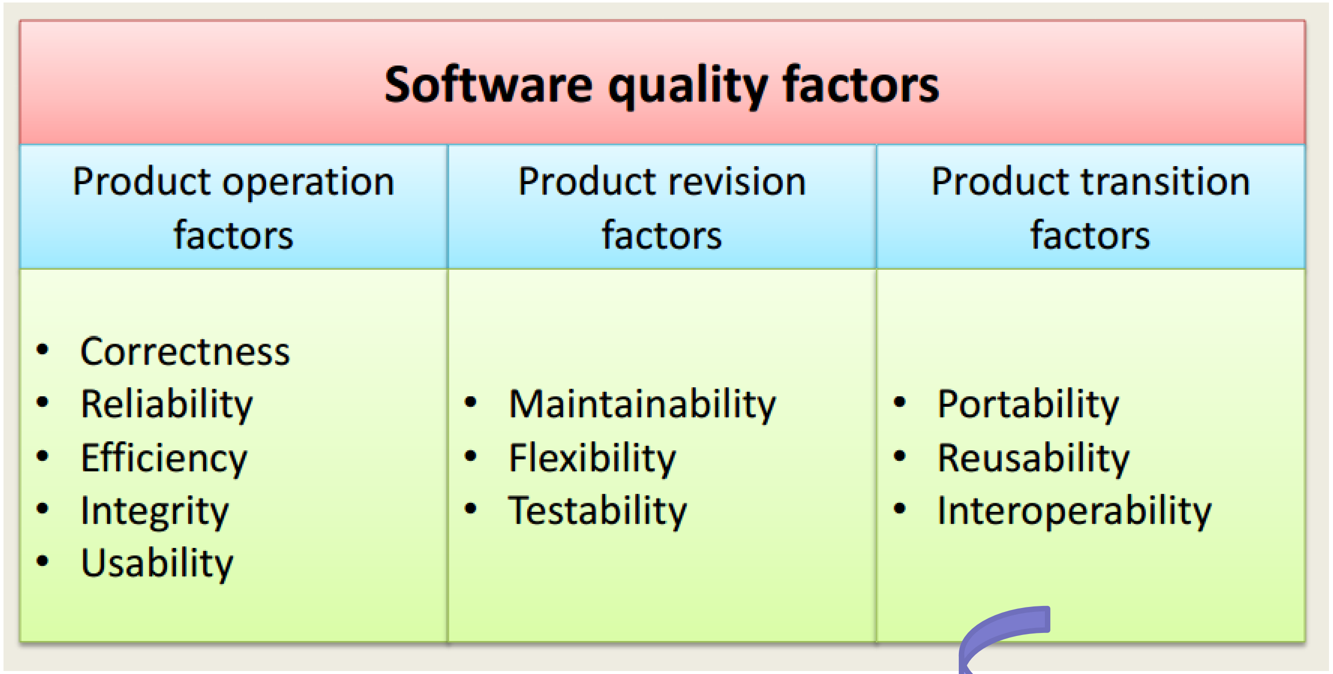
Identifies quality factors that influence the ability to change/ revisit the software product

Product revision factors

- Deal with the complete range of software maintenance activities:
  — corrective maintenance,
  — adaptive maintenance and
  — perfective maintenance.
- Maintainability: Can it be fixed ? (fixing bugs and errors)
  — effort to identify and fix software failures (modularity, documentation, etc.)
- Flexibility: Can it be changed ? (modifying an operational program)
  — degree of adaptability (to new customers, tasks, etc.)
- Testability: Can it be tested ? (ensuring performance)
  — support for testing (e.g. log files, automatic diagnostics, etc.)

Again: these factors influences the ability to change/ revisit the software product

Software quality factors (McCall's Quality Model – 1977)



Software quality factors		
Product operation factors	Product revision factors	Product transition factors
• Correctness • Reliability • Efficiency • Integrity • Usability	• Maintainability • Flexibility • Testability	• Portability • Reusability • Interoperability

Identifies quality factors that influence the ability to adapt the software to new environments

Product transition factors

- Portability: Can it be used on another machine ? (platform dependence)
  — adaptation to other environments (hardware, software)
- Reusability: Can parts of it be reused ? (generic coding)
  — use of software components for other projects
- Interoperability: Can it interface with other system ? (coupling system)
  — ability to interface with other components/systems

Again: These quality factors influences the ability to adapt the software to new environments

Wrap-up

- Faults, errors, failures
- Fault location
- Infected state
- RIPR model
- Observability and revealibility
- Software Quality
- McCall’s quality model
- Term projects ideas

What’s Next?

- Model-Driven Test Design (MDTD)