§1 Fundamentals of Software Testing

§1.1 Defination of Software Defect

IEEE (1983) 729：

- From an **internal** perspective, software defects refer to **various problems such as errors and defects** that exist during the **development or maintenance** process of software products;
- From an **external** perspective, software defects refer to the **failure or violation** of **a certain function** that a system needs to implement.

ISO 29119：

- **A flaw**（缺陷） in a component or system that can cause it to **fail to perform its required function**.
- **Any condition** that **deviates**（偏离） from expectation based on requirements specifications, design documents.

Defects may be found during, but not limited to, reviewing, testing, analysis, compilation, or use of software products or applicable documentation

§1.2 RIP/PIE Model

Software Fault A *static defect* in the software (i.e., defect)

```
public static void CSta (int [] numbers)
   {
     int length = numbers.length;
     double mean, sum;
     sum = 0.0;
     for (int i = 1; i < length; i++)//i=0
     {
       sum += numbers [i];
     }
     mean = sum / (double) length;
     System.out.println ("mean: " + mean);
   }
```

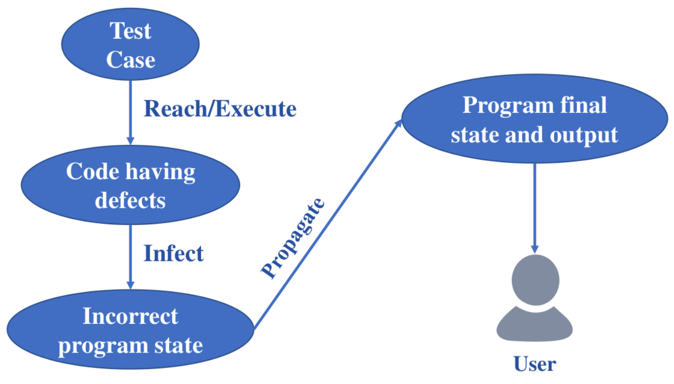
Software Error An *incorrect internal state* that is the manifestation（表现） of some defect

With Test Input: [3, 4, 5]
Expected: sum = 3 + 4 + 5
Actual: sum = 4 + 5

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

Expected mean = 4
Actual: mean = 3

§1.2.1 Three Conditions



- **Reachability/Execution**: The location or locations in the program that contain the defect must be reached
- **Infection**: The state of the program must be incorrect
- **Propagation**: The infected state must propagate（传播） to cause some output of the program to be incorrect

Does a software defect necessarily cause software errors?

Not necessarily. A defect may or may not result in an error. A defect in an unused or rarely executed part of the code may not manifest as an error during normal usage. However, defects in critical areas or frequently used features are more likely to cause errors.

Does a software error necessarily cause software failure?

Again, not necessarily. An error in a system may or may not lead to a failure. Some errors might be handled gracefully by the software, causing it to recover and continue functioning correctly. However, certain errors can propagate and eventually lead to a failure if they are not properly handled or if they affect critical aspects of the system

Difference between QA and software testing

Quality Assurance (QA):

Objective: QA is a broader approach that focuses on *ensuring the entire software development process is efficient, effective, and aligned with the quality standards and objectives of the organization*.
Scope: QA involves processes, methodologies, and activities that span the entire software development life cycle (SDLC). It includes activities such as requirement analysis, process improvement, project management, and overall organizational quality management.
Responsibilities: QA is concerned with *preventing defects rather than identifying them*. It aims to improve processes to ensure that defects are minimized from the start.
Role: QA is often seen as a management function that oversees the entire development process to ensure that the final product meets the required quality standards and customer expectations.

Software Testing:

Objective: Software testing is a specific activity within the QA process. It involves the systematic and controlled execution of a software application to identify defects, ensure that it behaves as expected, and meets specified requirements.
Scope: Testing is focused on the validation and verification of the actual software product. It includes various types of testing, such as unit testing, integration testing, system testing, acceptance testing, and more.
Responsibilities: Testing is primarily concerned with finding and fixing defects in the software. Testers aim to ensure that the software functions correctly, performs well, and meets the specified requirements.
Role: Testing is often considered an integral part of the development process, with dedicated testing teams responsible for planning, designing, and executing tests.

Explain the steps for Bug Life Cycle

New:

Description: A new bug is identified or reported by a tester, developer, or end user.
Status: The bug is in the "New" state, indicating that it has been identified but not yet assigned or verified.

Assigned:

Description: The bug is assigned to a developer or a development team for further investigation and resolution.
Status: The bug is marked as "Assigned" to indicate that someone is actively working on it.

Open:

Description: The developer reviews the bug report and confirms that it is a valid issue that needs attention.
Status: The bug transitions to the "Open" state, indicating that it is acknowledged and accepted as a legitimate problem.

In Progress:

Description: The developer starts working on fixing the bug and implementing the necessary changes in the code.
Status: The bug is marked as "In Progress" to indicate that development work is ongoing.

Fixed:

Description: The developer believes they have fixed the bug, and the changes are implemented in the code.
Status: The bug is marked as "Fixed" to indicate that the developer has addressed the reported issue.

Ready for Retest:

Description: The fixed code is ready for testing to verify whether the bug has been successfully resolved.
Status: The bug transitions to the "Ready for Retest" state, signaling that it is awaiting confirmation from the testing team.

Reopened (if applicable):

Description: If the testing team finds that the bug persists or if new issues arise, they can reopen the bug.
Status: The bug returns to the "Open" or "In Progress" state for further investigation and resolution.

Retest:

Description: The testing team performs retesting to verify whether the reported bug has been successfully fixed.
Status: If the bug is confirmed as fixed, it moves to the "Closed" state. If not, it may be reassigned for further development.

Closed:

Description: The bug is considered resolved, and the testing team confirms that the reported issue has been successfully fixed.
Status: The bug is marked as "Closed," indicating that the issue is resolved and the development cycle for that particular bug is complete.

§2 Testing Principles

§2.1 Test Types

§2.1.1 By Structure

Black-box testing Take the software as a black box and do not care what's going on inside

- Equivalence Parititioning
- Boundary Value Analysis
- Decision Table
- Cause-Effect Graph

White-box testing Take the software as a white/glass box and can see the internal structure of code.

- 逻辑覆盖（Logic Coverage）
- 基础路径覆盖（Basic Path Coverage）
  在控制流图（Control Flow Graph, CFG）的基础上，通过分析控制结构得出基本路径

§2.1.2 By Execute or not

Static testing without executing program

- Software document oriented
- Source code oriented
  - Code Checking
  - Stack Structure Analysis
  - Code Quality Measurement

Dynamic testing with executing program

- Functional Verification and Interface Testing
- Coverage Analysis
- Performance Analysis

§2.1.3 by Phases/Levels:

Unit testing 测试最小的可测量单元（例如函数/类）

Integrated testing 测试子系统（由通过单元测试的模块组成）

System testing 在实际环境下进行测试，保证满足用户需求

Acceotance Testing 由用户来进行，来判断是否满足其需求或合同

Equivalence Partitioning:

Type: Dynamic
Reason: Equivalence Partitioning is a dynamic testing technique. It involves the selection of test cases based on dividing the input domain into equivalence classes and is performed during the execution of the code.

Use Case Testing:

Type: Dynamic
Reason: Use Case Testing involves the execution of scenarios based on use cases to verify the system's behavior. It is a dynamic testing technique applied during the runtime of the software.

Data Flow Analysis:

Type: Static
Reason: Data Flow Analysis is a static analysis technique. It involves examining how data flows through a program without executing the program. It's used to find potential issues related to data flow within the code.

Exploratory Testing:

Type: Dynamic
Reason: Exploratory Testing is a dynamic testing technique where testers explore the software without predefined test cases. It involves interaction with the application during its execution.

Decision Testing:

Type: Dynamic
Reason: Decision Testing is a dynamic technique. It focuses on testing the decision points or conditional statements in the code during execution.

Inspections:

Type: Static
Reason: Inspections are a static testing technique. They involve a formal process of reviewing and examining the software artifacts, such as code or design documents, without executing the code.