Component Testability

San José State University

San José State
UNIVERSITY

By Jerry Zeyu Gao, Ph.D.
San Jose State University
Email: jerry.gao@sjsu.edu
URL:/www.engr.sjsu.edu/gaojerry

Presentation Outline

- Basic concepts of software testability
- Understanding component testability
  - Component understandability
  - Component observability
  - Component traceability
  - Component controllability
  - Test support capability
- Design for component testability
  - Understanding different approaches to increasing component testability
  - Building BIT components
  - Building testable components
- Verification and evaluation of component testability
- Software testability measurement

Basic Concepts of Software Testability

Testability is a very important quality indicator of software and its components since its measurement leads to the prospect of facilitating and improving a software test process.

"The degree to which a system or component facilitate the establishment of test criteria and the performance of tests to determine whether those criteria have been met; the degree to which a requirement is stated in terms that permit the establishment of test criteria and performance of tests to determine whether those criteria have been met." (by IEEE Standard)

Software testability depends on our answer to the following questions:

→ **Do we construct a system and its components in a way that facilitates the establishment of test criteria and performance of tests based on the criteria?**

→ **Do we provide component and system requirements that are clear enough to allow testers to define clear and reachable test criteria and perform tests to see whether they have been met?**

Basic Concepts of Software Testability

Poor testability of components and programs indicate:

→ ***the poor quality of software and components***

→ ***the ineffective test process***

Although verifying and measuring software testability after implementation is useful for quality control, it is little too late to increase and enhance the testability of software and its components.

In a practice view, we need to focus on the two areas:

→ ***How to increase software testability by constructing highly testable components and systems?***

→ ***How to analyze, control, and measure the testability of software components in all phases of a software development process?***

Different Tasks and Activities Relating Testability

Requirements Analysis:

→ Clearly define testable and measurable requirements

→ Provide well-defined facilitating requirements for software testing

→ Review and evaluate system/component requirements to make sure they are testable and measurable

Software Design:

→ Conduct design for testability of software and components, i.e. come out architecture model and interfaces increasing testability.

→ Define design patterns, standards, and framework for testable components

→ Review and evaluate software design concerning software testability

Implementation:

→ Implement testable components and built-in tests

→ Generate software testing facilities and reusable framework

→ Review program/component code based on well-defined testability standards

Different Tasks and Activities Relating Testability

Testing:

- → ***Define achievable component test criteria and high quality tests/scripts***
- → ***Develop, set-up, and use test beds and facilities for components***
- → ***Perform component tests and monitor coverage based on defined criteria***
- → ***Verify, evaluate, and measure component testability***

Maintenance:

- → ***Update and review test criteria and tests based on component changes***
- → ***Maintain testable components and built-in tests***
- → ***Maintain component test framework and test beds***
- → ***Evaluate, verify, and measure the testability of components***

Understanding Component Testability

What is software component testability?

- R. S. Freedman defined component testability in a function domain by considering two factors:

(a) Component Observability and (b) Component Controllability

- R. V. Binder discussed testability of object-oriented programs by considering six factors:

(A) Representation, (B) Implementation, (C) Built-in Test

(D) Test Suite, (E) Test Support Environment, and (F) Process Capability

Component testability is two-fold:

- It refers to the degree to which a component is constructed to facilitate the establishment of component test criteria and the performance of tests to determine whether those criteria have been met.

- It refers to the degree to which testable and measurable component requirements are clearly given to allow the establishment of test criteria and performance of tests.

Understanding Component Testability

Software component testability depends on the following five factors:

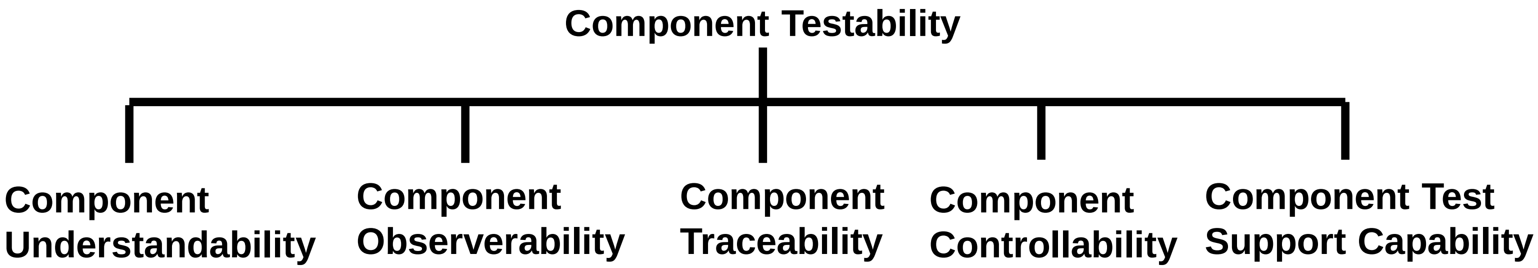
→ **Component understandability**

→ **Component observability**

→ **Component traceability**

→ **Component controllability**

→ **Component testing support capability**



Component Testability

Component Understandability

Component Observerability

Component Traceability

Component Controllability

Component Test Support Capability

Studying component testability focuses on two aspects:

1. **Studying how to construct testable components, including component development methods, guidelines, principles, and standards**
2. **Studying how to verify and measure component testability based on established test criteria**

Component Understandability

Component understandability depends on the following two factors:

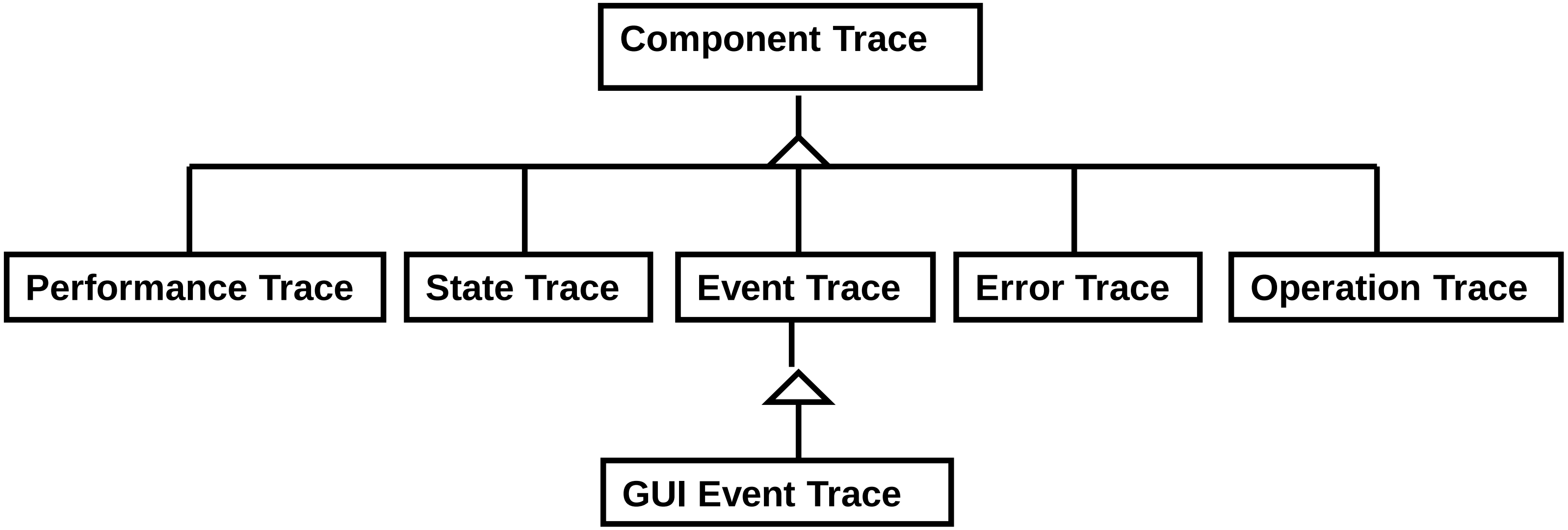
- → **Availability of component artifacts for component users, such as component function specifications, interfaces, programs, testing doc.**
- → **Availability of component artifacts for component developers, such as Component user manual and reference documents.**
- → **Understandability of component artifacts.**

Component observability indicates how easy it is to observe a program based on its operation behaviors, input parameter values, and actual outputs for a test case.

Component Traceability

Component traceability depends on the following five factors:

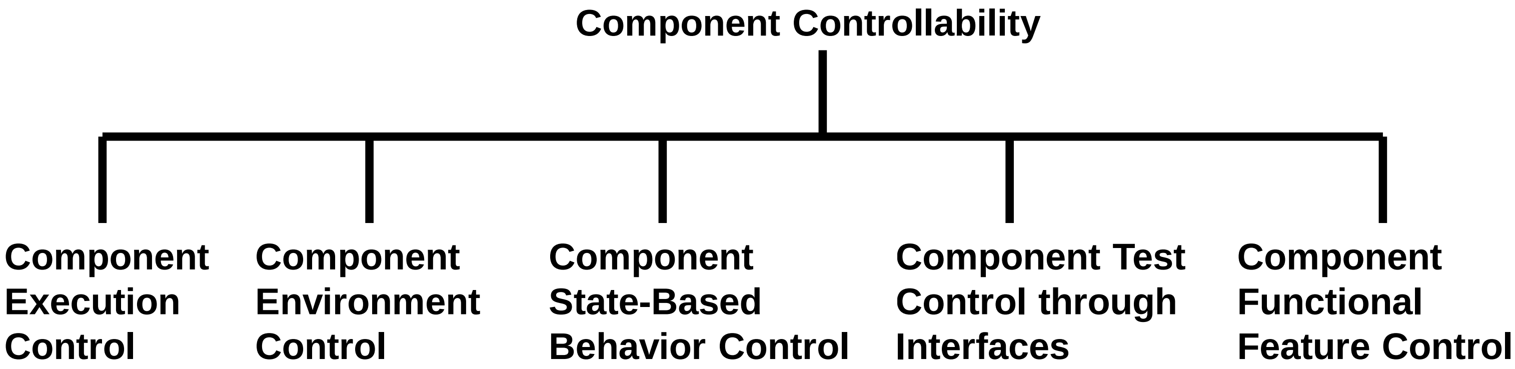
- → **Component error traceability**
- → **Component state behavior traceability**
- → **Component event traceability**
- → **Component function/operation traceability**
- → **Component performance traceability**



Component Controllability

Component controllability depends on the following five factors:

- → **Component execution control**
- → **Component environment control**
- → **Component state-based behavior control**
- → **Component test control through interfaces**
- → **Component functional feature control**



Component Controllability

Component Execution Control

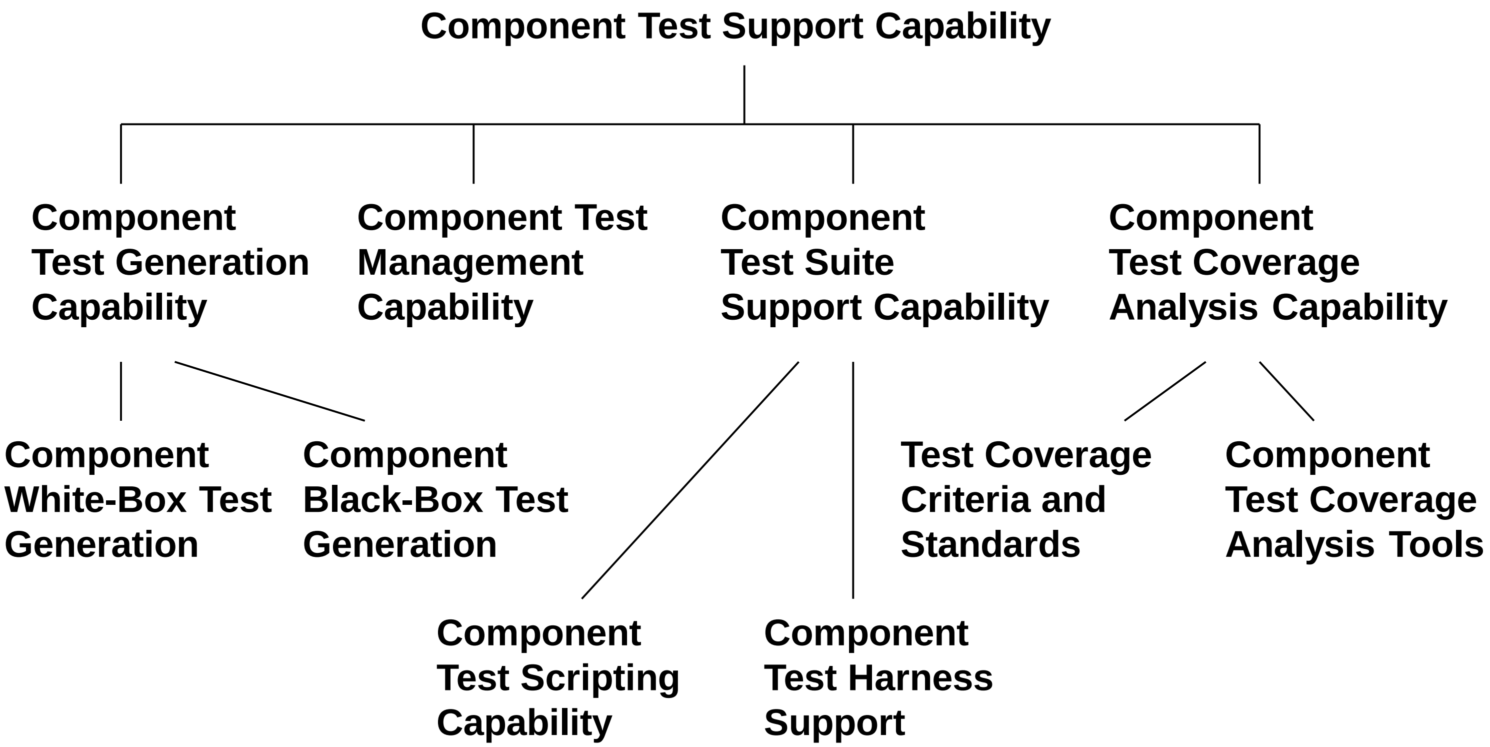
Component Environment Control

Component State-Based Behavior Control

Component Test Control through Interfaces

Component Functional Feature Control

Component Test Support Capability



Component Test Support Capability

Component Test Generation Capability

Component Test Management Capability

Component Test Suite Support Capability

Component Test Coverage Analysis Capability

Component White-Box Test Generation

Component Black-Box Test Generation

Component Test Scripting Capability

Component Test Harness Support

Test Coverage Criteria and Standards

Component Test Coverage Analysis Tools

Component Testability Issues and Challenges

Component testability issues in CBSE:

- → **How to construct components with high testability? (in other words, how to create testable software components?)**
- → **How to increase component testability in a component reuse process?**
- → **How to check component testability during a component development process?**
- → **How to measure component testability in a component development process?**

Challenges in studying component testability:

- → **Creating component testability models**
- → **Finding systematic methods to create testable components**
- → **Developing systematic methods to verify component testability**
- → **Defining measurement methods and metrics for component testability**

Design for Component Testability

Design for component testability refers to all engineering activities to enhance component testability for software components in a component development process.

Challenges in building testable components:

→ How to specify testability requirements for components?

→ How to construct components to achieve high testability? (including construction approaches, component architecture, test interface, ….)

→ How to support test automation for testable components?

→ How to verify generated component testability in a systematic solution?

→ How to measure and analyze the testability of components during a component development process in a systematic approach?

Three Common Approaches

Three common approaches to increase component testability:

- ***Method #1: Framework-based testing facility***

  → Creating well-defined framework (such as a class library) is developed to allow engineers to add program test-support code into components according to the provided application interface of a component test framework.

- ***Method #2: Build-in tests***

  → Adding test-support code and built-in tests inside a software component as its parts to make it testable.

- ***Method #3: Systematic component wrapping for testing***

  → Using a systematic way to convert a software component into a testable component by wrapping it with the program code that facilitates software testing.

Built-in Test Components

Definition:

According to Y. Wang et al, a built-in test component is a special type of software component in which special member functions are included as its source code for enhancing software testability and maintainability.

Major features:

- **Built-in test components are able to operate in two modes:**
  - **a) normal mode – a component behaviors as its specified functions.**
  - **b) maintenance mode – its internal built-in tests can be activated by interacting a tester (or user).**
- **Built-in tests as a part of a component. (see an example)**

Major limits:

- **Only limited tests can be built-in tests due to component complexity**
- **It is costly to change and maintain built-in tests during a component development process.**

Comparison of Three Approaches

Different Perspectives	Framework-Based Testing Facility	Built-in Tests	Systematic Component Wrapping for Testing
Programming Overhead	Low	High	Very Low
Testing Code Separated from Source Code	No	No	Yes
Software Tests inside Components	No	Yes	No
Test Change Impact on Components	No	High	No
Software Change Impact on Component Testing Interfaces	No	Yes	No
Component Complexity	Low	Very High	High
Usage Flexibility	High	Low	Low
Applicable Components	In-house components and newly developed components	In-house components and newly developed component	In-house components and COTS as well as newly constructed components

What Is A Testable Component?

"A testable bean is a testable software component that is not only deployable and executable, but is also testable with the support of standardized components test facilities." (by Jerry Zeyu Gao et al.)

The basic requirements of a testable bean:

Requirement #1: A testable bean should be deployable and **executable.**

A JavaBean is a typical example.

Requirement #2: **A testable bean must be traceable by supporting basic component tracking capability that enables a user to monitor and track its behaviors.**

Requirement #3: **A testable bean must provide a consistent, well-defined, and built-in interface, called component test interface, to support external interactions for software testing.**

Requirement #4: **A testable bean must include built-in program code to facilitate component testing by interacting with the two provided test interfaces to select tests, set up and run tests, and check test results.**

Why Do We Need Testable Components?

The major goal of introducing testable components is to find a new way to develop software components which are easily to be observed, traced, tested, deployed, and executed.

The major advantages of testable components:

- ***Increasing component testability by enhancing component understandability, observability, controllability, and test support capability.***

- ***Standardizing component test interfaces and interaction protocols between components and test management systems and test suite environments.***

- ***Reducing the effort of setting up component test beds by providing a generic plug-in-and-test environment to support component testing and evaluation.***

- ***Providing the basic support for a systematic approach to automating the derivation of component test drivers and stubs.***

Principles of Building Testable Components

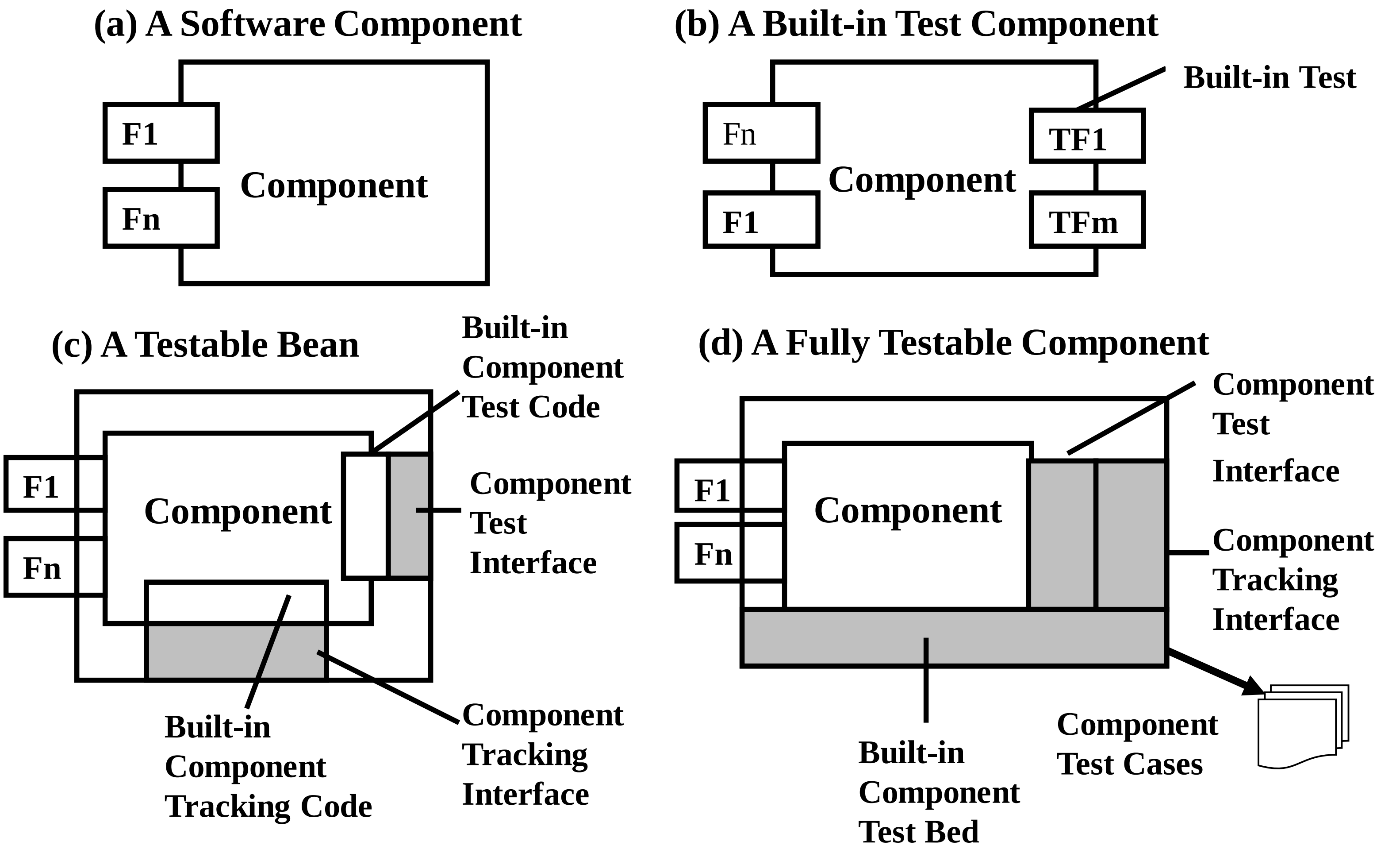
The essential needs in constructing testable components are:

- ***Well-defined component models concerning test support***
- ***Consistent test interfaces between components and external test tools and facilities***
- ***Effective ways and mechanisms to construct testable components***

The basic principles of building testable components:

- ***It is essential to minimize the development efforts and program overheads when we increase component testability by providing systematic mechanisms and reusable facilities.***
- ***It is important to standardize component test interfaces for testable beans so that they can be tested in a reusable test bed using a plug-in-and-play approach.***
- ***It is always a good idea to separate the component functional code from the added and built-in code that facilitates component testing and maintenance.***

Different Architecture Models for Testable Components



(a) A Software Component

(b) A Built-in Test Component

(c) A Testable Bean

(d) A Fully Testable Component

F1
Fn
Component
Built-in Test
Fn
F1
Component
TF1
TFm
Built-in Component Test Code
F1
Fn
Component
Component Test Interface
Built-in Component Tracking Code
Component Tracking Interface
Component Test Interface
F1
Fn
Component
Component Tracking Interface
Built-in Component Test Bed
Component Test Cases

Maturity Levels for Testability

Evaluating the maturity levels of a test process concerning testability:

Level #1- Initial – At this level, component developers and testers use an ad hoc approach to enhance component testability in a component development process.

Level #2- Standardized – At this level, component testability requirements, design methods, implementation mechanisms, and verification criteria are defined as standards.

Level #3- Systematic – At this level, a well-defined component development and test process and systematic solutions are used to increase component testability at all engineering phases.

Level #4-Masurable – At this level, component testability can be evaluated and measured using systematic solutions and tools in all component development phases.

Verification of Component Testability

Verification of component testability:

→ Check component testability of software components using well-defined verification means during a component development process.

Static verification approach –

Using various verification methods to check the generated component artifacts in all phases, including component requirements, interface specifications, design logic, implementation, and test cases and results.

→ This enhances component testability by discovering testability issues in all phases of a component development process

Statistic verification approach –

Using statistical methods to analyze and estimate component testability by examining how a given component will behave when it contains faults.

→ This suggests the testing intensity or testing difficulty in discovering a fault at a specific location.

→ This suggests the number of tests necessary to gain quality confident.

Verification of Component Testability

Static verification approach –

→ **Component specification phase:**

Checking component requirements are clearly specified so that they can be tested and measured for a given test criteria.

How to specify them? How to verify them for testability?

→ **Component design phase:**

Checking component design for testability -> focusing how the current component design to meet the given testability requirements, including component model, architecture, interfaces for testing, test facility design

How to verify design artifacts for component testability?

→ **Component implementation phase:**

Checking if component design for testability has been properly implemented

→ **Component testing phase:**

Checking component tests based on the given test criteria

Measuring component testability based on a component testability model

Verification of Component Testability

Statistical verification approach –

→ Use a statistical approach to examine how a given program behave when it contains a fault.

Example: Jeffrey Voas proposed a verification approach (sensitivity analysis) to check program testability.

Its major objective is to predict the probability of a software failure occurring if the particular software contains a fault for a given set of test set for black-box testing.

Jerrfrey Voas' method involves three estimations at each location:

→ Execution probability

→ Infection probability

→ Propagation probability

Measurement of Software Testability

What is software testability measurement?

→ **Software testability measurement refers to the activities and methods that study, analyze, and measure software testability during a product development cycle.**

How to measure software testability? Three types of measurement methods:

- **Program-based measurement methods for software testability**

  Example, J. –C. Lin's program-based method to measure program testability by considering the single faults in a program.

- **Model-based measurement methods for software testability**

  Example, C. Robach and Y. Le Traon use the data flow model to measure software testability. Similarly, J.-C. Lin and S.-W. Lin's approach.

- **Dependability assessment methods for software testability**

  Example, A. Bertolino and L. Strigni's black-box approach which measures software testability based on the dependency relationships between inputs and corresponding outputs.

Measurement of Software Testability

Program-based measurement methods for software testability

Example, J. –C. Lin's program-based method to measure program testability by considering the single faults in a program.

The basic idea of this approach is similar to software mutation testing.

To compute the testability of a software at a specific location based on a single failure assumption:

- **A single fault is instrumented into the program at a specific location.**
- **The newly instrumented program is compiled and executed with an assumed input distribution.**
- **Three basic techniques (execution, infection, and propagation estimation) are used to compute the probability of failure that would occur when that location has a fault.**

Measurement of Software Testability

Model-based measurement methods for software testability

Example, C. Robach and Y. Le Traon use the data flow model to measure software testability. Similarly, J.-C. Lin and S.-W. Lin's approach.

-> A white-box based approach for testability measurement

Three steps:

1. **Normalizing a program before the testability measurement using a systematic tool.**
   - **Structure normalization and block normalization**

2. **Identifying the testable elements of the target program based on its normalized data flow model.**
   - **Including number of non-comment lines, nodes, edges, p-uses, defs, uses, d-u paths, and dominating paths.**

3. **Measuring the program testability based on data flow testing criteria.**
   - **Including ALL-NODES, ALL-EDGES, ALL-P-USES, ALL-DEFS, ALL-USES, ALL-DU-PAIRS and ALL-DOMINATING PATH.**

Measurement of Software Testability

Dependability assessment methods for software testability

Example, A. Bertolino and L. Strigni's black-box approach which measures software testability based on the dependency relationships between inputs and corresponding outputs.

- **A black-box approach for testability measurement**
- **Testability is computed based on the probability of a test of the program based on a given input setting is rejected by the program due to its faculty.**

The basic approach consists of the following steps:

- **Perform an oracle in a manual (or systematic) mode to decide whether a given program behave correctly on a given test.**
- **The oracle decides the test outcome by analyzing the behavior of the program against its specification.**
- **Observes the input and the output of each test against the expected output, and looks for failures.**