ENSE 623 Systems Validation and Verification

Established Approaches to System Validation/Verification

Mark Austin

E-mail: austin@isr.umd.edu

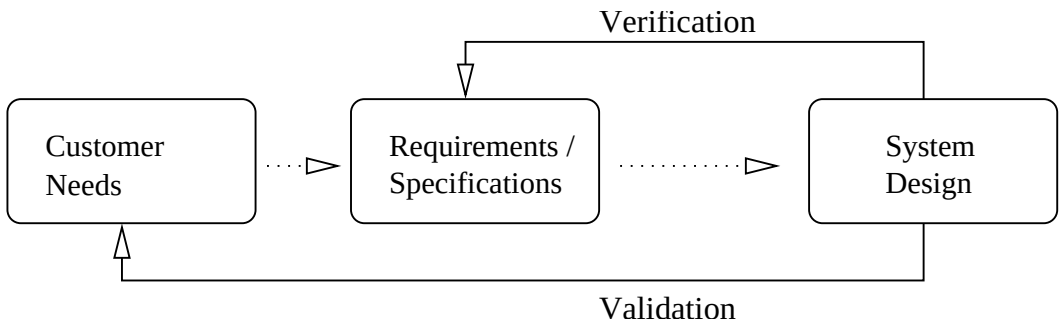
Institute for Systems Research, University of Maryland, College Park

Table of Contents

Definition

Definition and Complementary Roles

- Verification → “are we building the product right?”
- Validation → “are we building the right product?”



Historical Role

The Department of Defense glossary for defense acquisition (DoD, 1991) states:

> **Validation is the process of testing for technical appropriateness and adequacy near the end of the system life cycle process.**

Importance

Changes in System Development 1990-2010

During the past two decades:

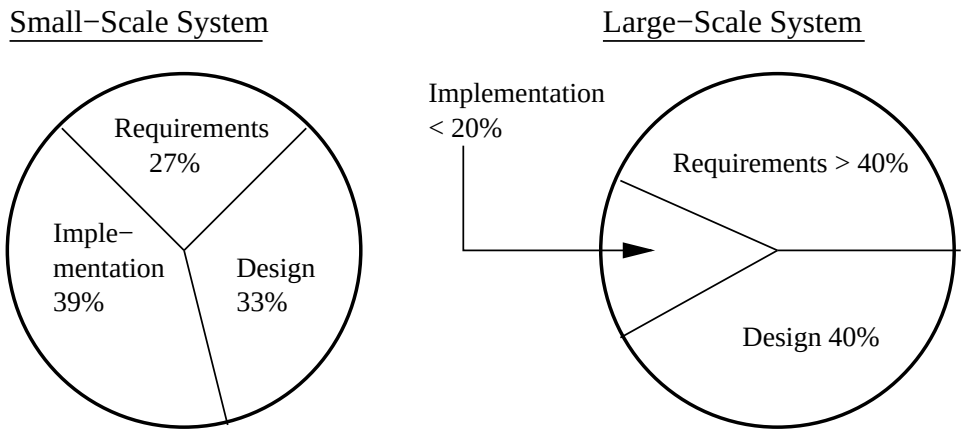
- Systems have become orders of magnitude more complex.
- Validation and verification of the complete system may require many tests to ensure the actual system has performance/behavior as intended.
- The associated costs may be prohibitive.

To help counter these trends:

Planning for validation and verification needs to begin in the early stages of requirements development.

Errors in Small-/Large-Scale Systems Development

The overall goal is to **maximize quality and minimize risk.**



Common Errors in Small-Scale Systems Development

1. Lack of separation between requirements, design, code.
2. Wide variations in quality of the system architecture.
3. Lack of attention to deal with abnormal conditions.
4. Incomplete interfaces.
5. System designer bias.

Errors in Small-/Large-Scale Systems Development

Common Errors in Large-Scale Systems Development

For large-scale systems the common errors are:

1. Incomplete requirements.
2. Failure to deal with emergent behaviors.
3. Information and data coupling among systems having concurrent behaviors.
4. Lack of attention to events and timing/data dependencies.

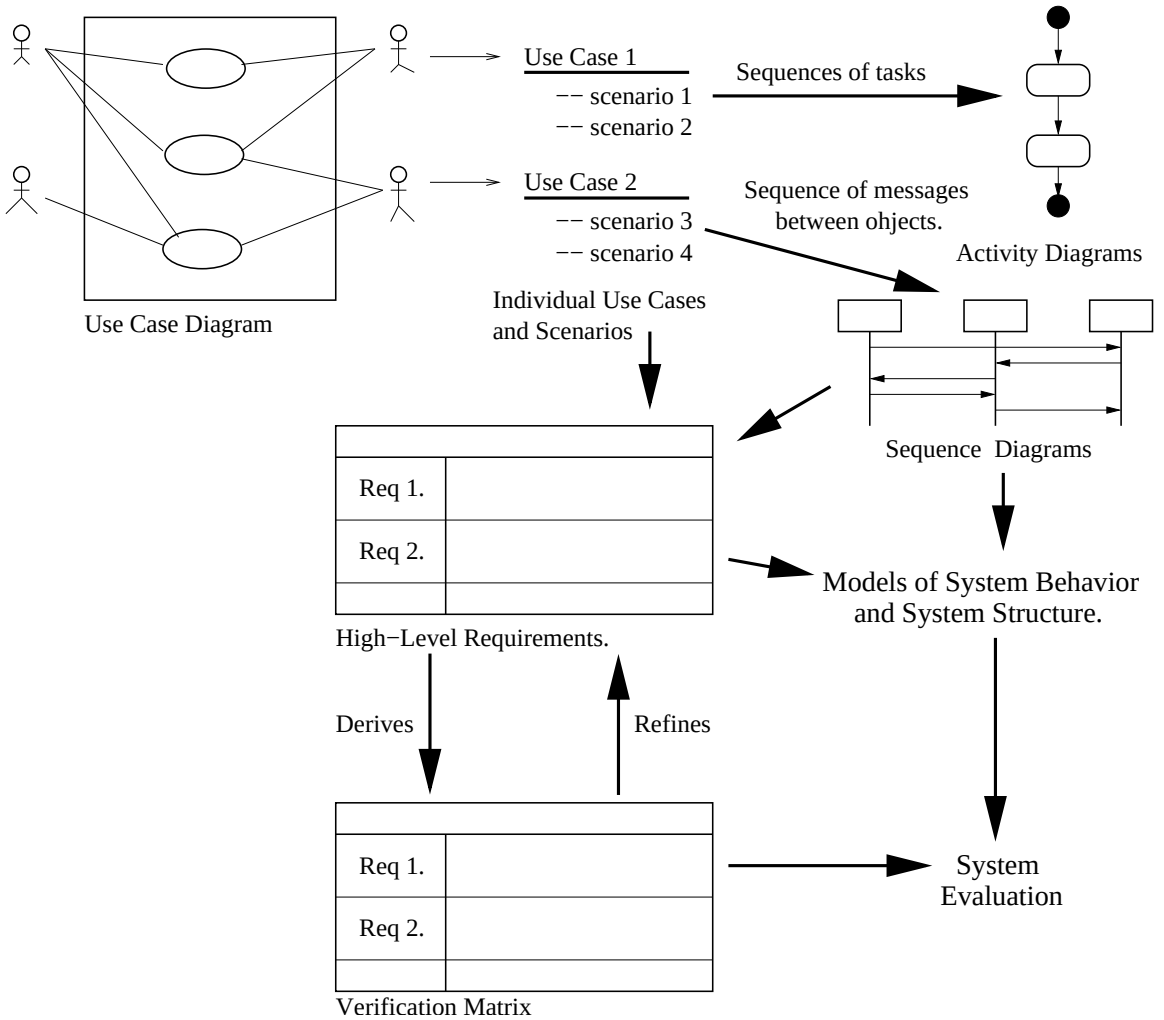
Development Strategy

Work towards:

> **Formal approaches to validation and verification, where savings come through lowering the cost to fix problems by finding them sooner.**

Flowdown from UML Diagrams

Flowdown from UML diagrams to Requirements and Verification Matrices



Types of System Validation/Verification

Types of Validation

The key purpose of validation is to check satisfaction of stakeholders. (i.e., Have you done the right job?).

> **Checking that a project has not strayed from its intended purpose can be interpreted as looking for faults in the development.**

If no failures are found then a certain degree of confidence in the design is justified.

Validation procedures fall into two types:

1. **Requirements Validation**

   Check for traceability. Have we missed any requirements? Do we have extraneous requirements?

2. **Product Validation**

   Check that product/system meets the needs and expectations of the stakeholders.

Requirements validation increases our confidence in the ability to synthesize a design.

Types of System Validation/Verification

Types of Verification

The key purpose of verification is to check compliance against specified requirements. (Have you done the job right?).

There are two types:

1. **Product and Process Qualification**

   Check for full compliance to specification.

   Product requalifications may be necessary when product is redesigned.

2. **Product Acceptance**

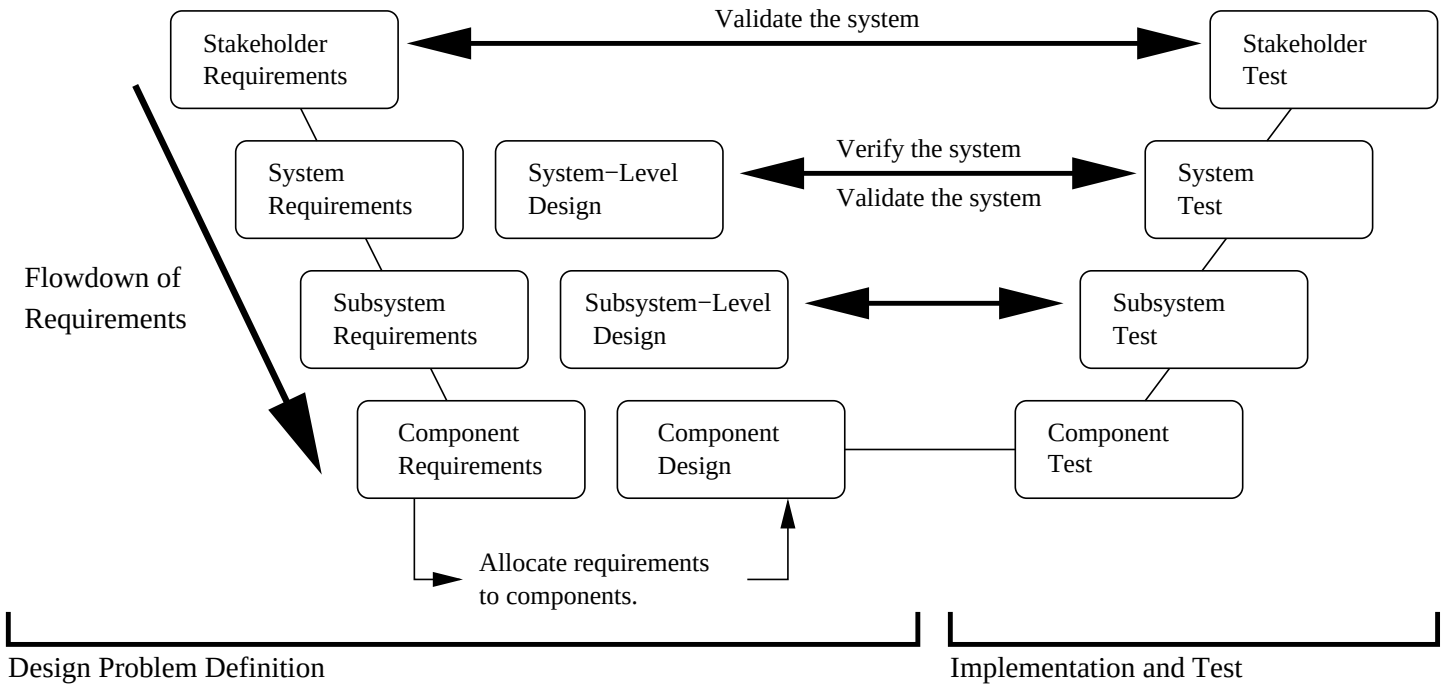
   Check for full compliance of key criteria.

   Done on every unit or on a sample basis (in the case of mass manufacturing).

   Can be done before shipping or after installation.

Types of System Validation/Verification

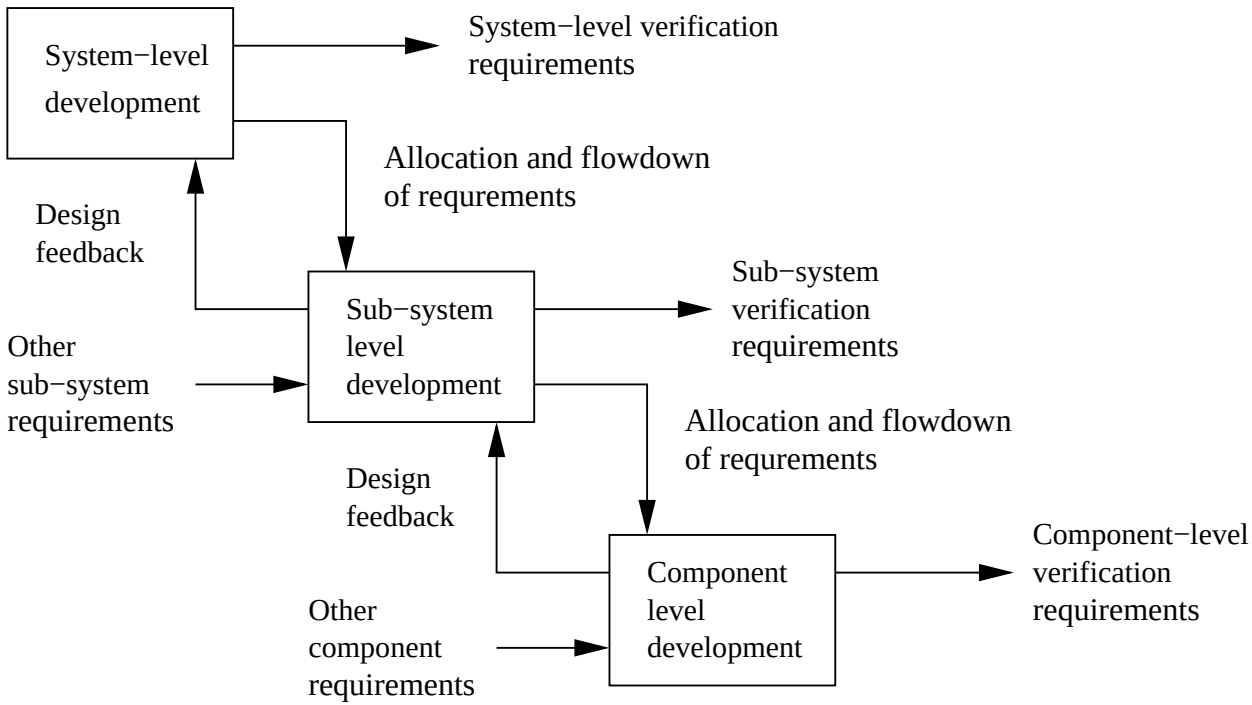
Levels of Validation/Verification



Types of System Validation/Verification

Verification Planning

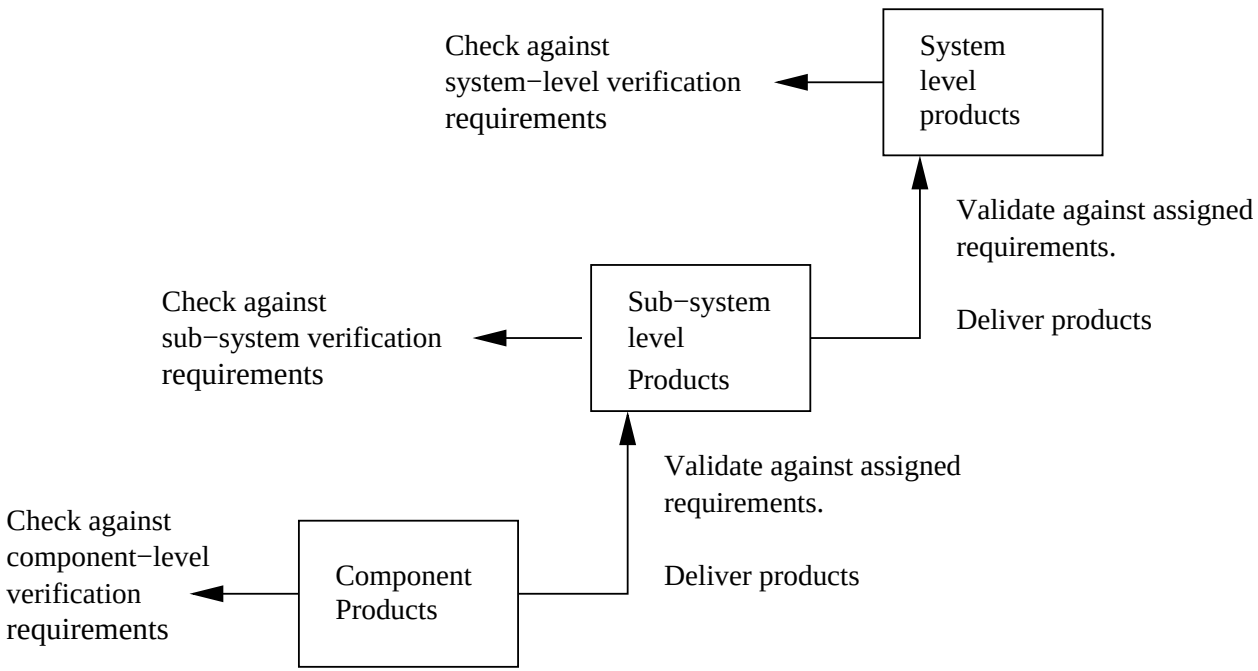
The detailed model of requirements flowdown is as follows (Source: Martin, 2003):



Types of System Validation/Verification

Verification Execution

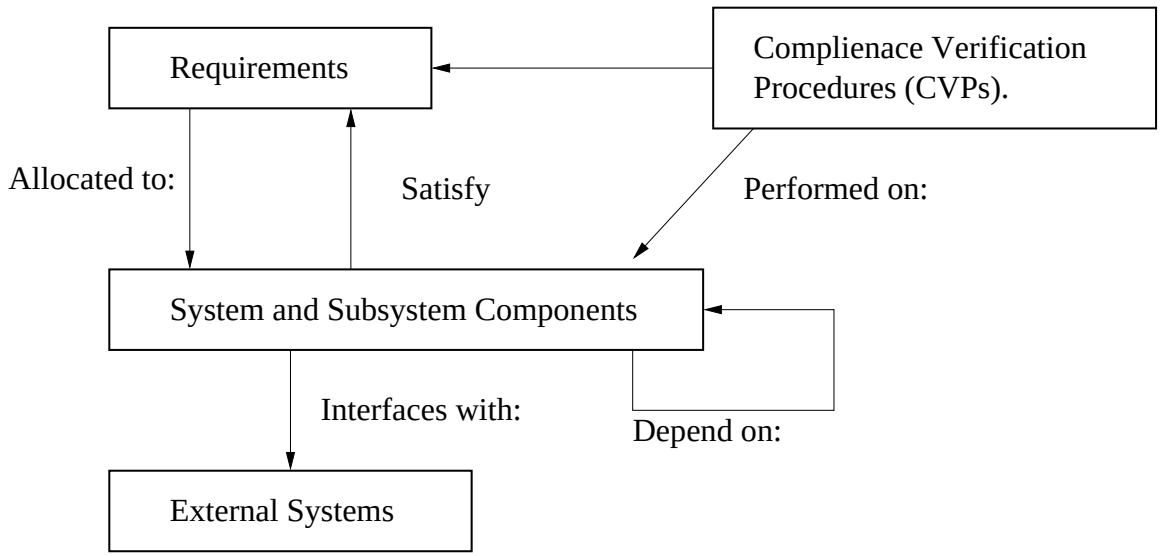
Testing and product delivery procedures begin at the component level and work toward the system and stakeholder tests (Source: Martin, 2003).



Types of System Validation/Verification

Compliance Verification Procedures

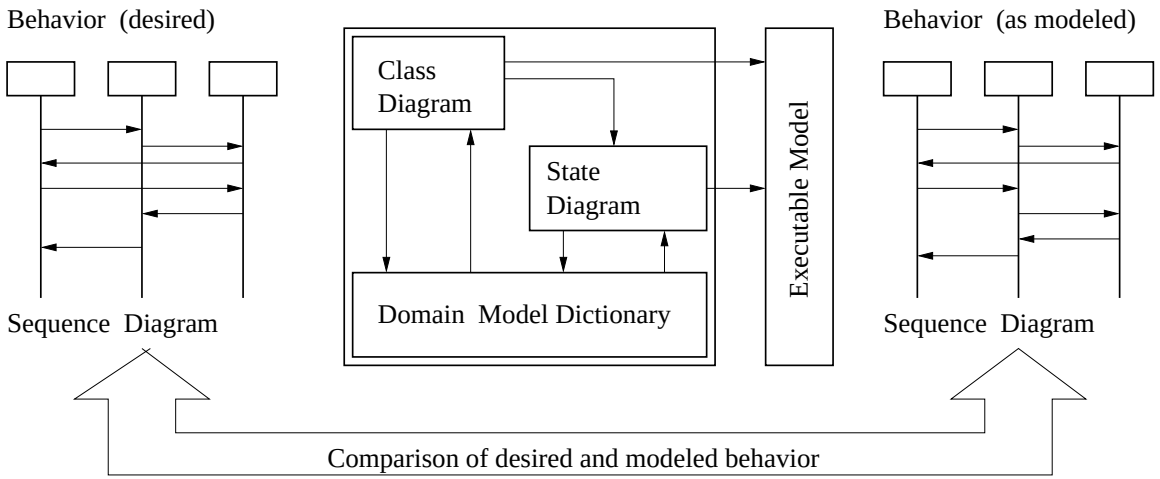
Compliance and verification procedures (CVPs) are developed to ensure that each requirement is satisfied.



Timely application of the verification procedures will help to unearth problems in the system development as early as possible.

Types of System Validation/Verification

Role of Simulation in Compliance Verification



At the end of each interaction, the “desired” and “as modeled” behaviors are compared:

1. When the comparison is good, we can proceed to the next (lower) level of object decomposition and/or to modeling of a new behavior (i.e., use case).
2. When the comparison is bad (or insufficient), the object classes and their state diagrams need to reengineered.

Detection of System Defects

Fundamental Law of Faults

- **Failures**

  A failure is an externally visible incorrect behavior of a system.

- **Errors**

  An error is an incorrect internal (system) state, which may or may not be externally detectable as a failure.

- **Faults**

  A fault is a mistake in a system which causes one or more errors and failures.

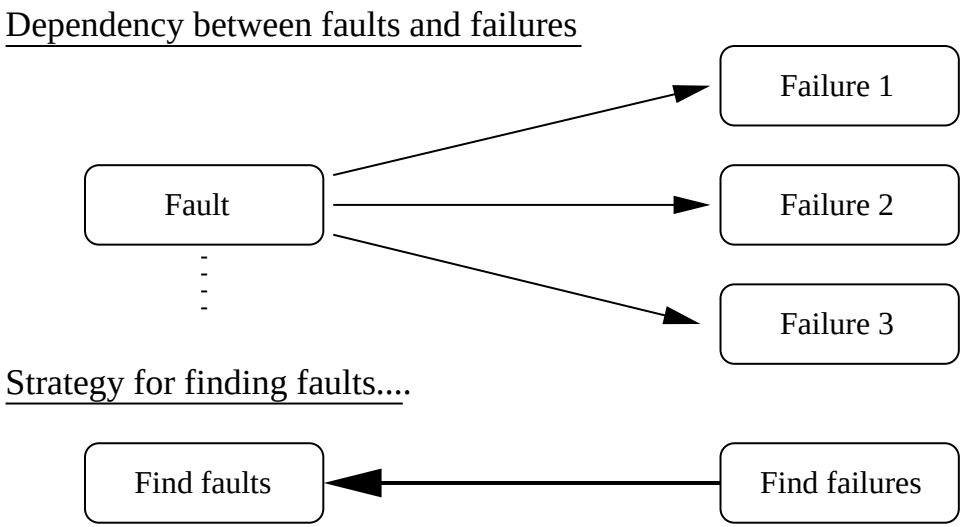
At a glance, it would appear that the purpose of system validation and verification procedures is the detection of system failures.

..But actually, we need to find and fix the cause of the failures (i.e., the underlying faults).

Detection of System Defects

Relationship between Faults and Failures

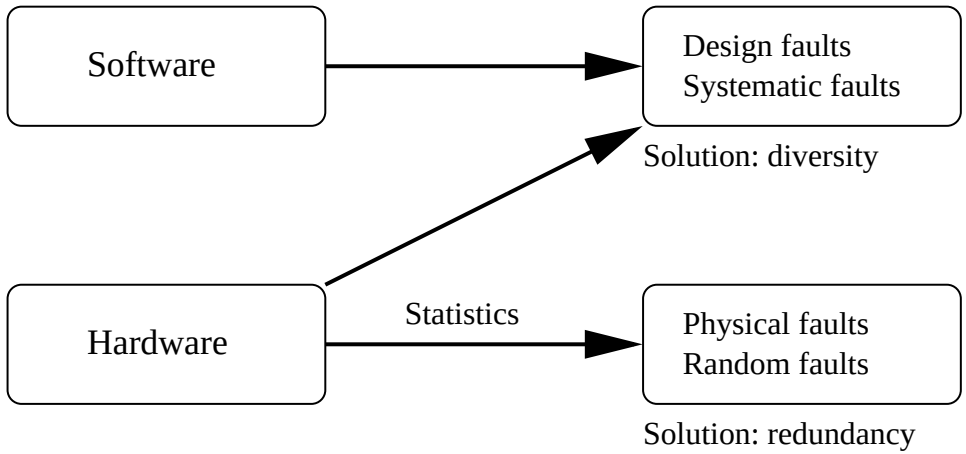
Generally, a single fault can cause many types of system failure – this actually makes the detection of faults easier.



It follows from the aforementioned definitions that testing procedures will be unable to determine the existence of a fault unless they can first identify any of the failures it causes.

Detection of System Defects

Identifying Faults in Hardware and Software



Software failures:

- Design failures due to faulty logic.

Hardware failures:

- Can use statistical analysis to quantify design failures due to physical faults/weaknesses.

Detection of System Defects

Detection and Removal of Defects

Normal techniques for the detection and removal of defects include:

- **Consistency Checking**

  Attempts to find inconsistencies between ...

  ... different parts of the specification and/or between the specification and a formal model of the design.

- **Simulation**

  Simulation procedures can detect inconsistencies between ...

  ... a user's notion of the required system behavior and the system behavior captured by the requirements specification.

Inspection and Testing

Inspection Procedures

- Inspection is a form of validation where one or more people study a system and its specification and attempt to satisfy themselves that the system is correct.

  Common measures of quality assurance include product/system finish, fit, location, dimensions, and so forth.

- Inspection procedures are most effective when they are conducted by individuals who were "not" part of the system development.

Limitations of Inspection Procedures

- People are quite limited in the total amount of detail they can synthesize and reason with at any given moment.

As a result ....

... inspection procedures are really only suitable for validation of simple system architectures.

Inspection and Testing

Design of Test Procedures

The most common form of system verification is testing – that is ...

> **... given the specification of a system, test cases are created which embody the fine microscopic facets of the specification.**

At each level of development, the goal of testing is to

> **... identify faults in the system development.**

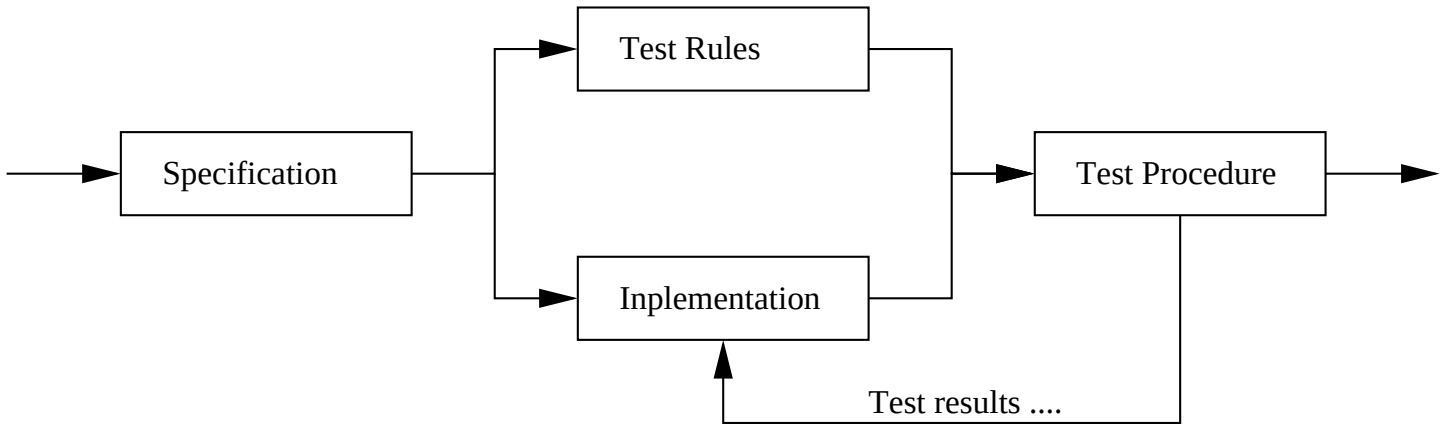
It is important to note that ...

> **... testing can show the presence of faults or bugs ... but not their absence.**

Inspection and Testing

Testing Procedure

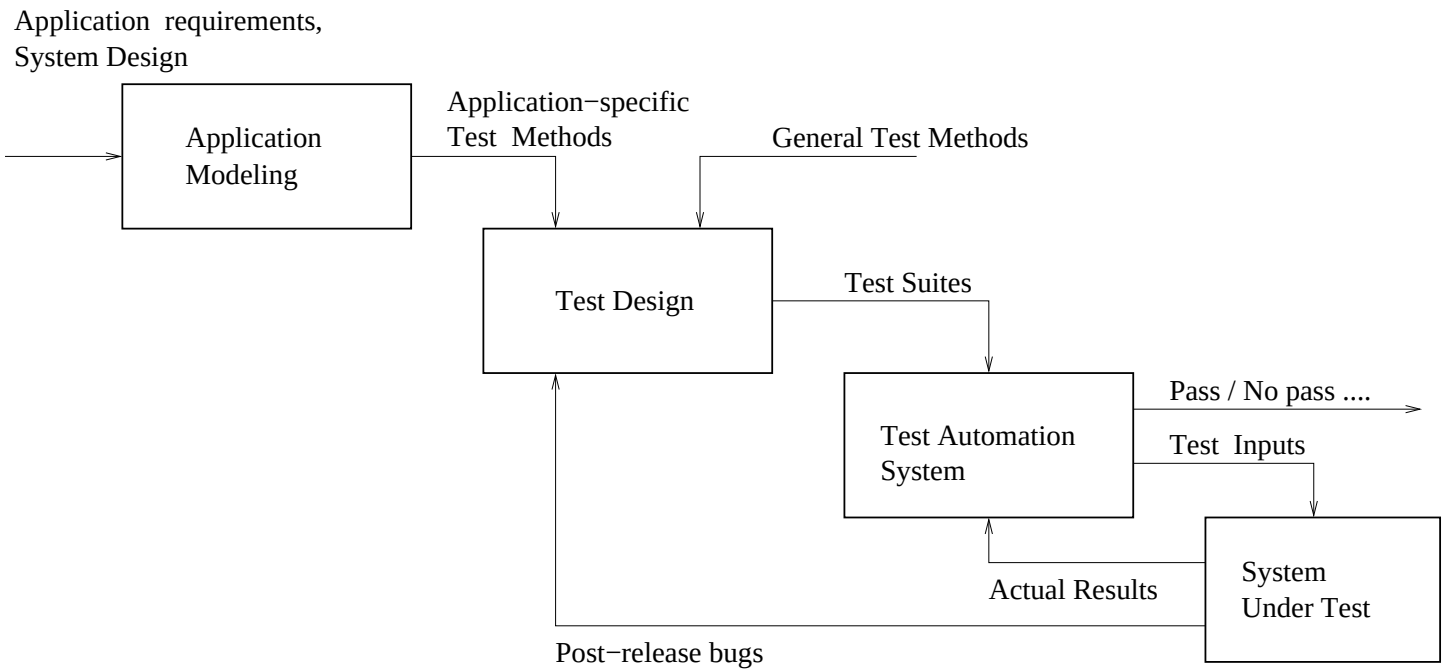
Testing requires a test specification, a suite of test rules, and a test protocol.



Testing can only reveal errors (and not the lack of them).

Inspection and Testing

Systems Engineering View of Testing



Inspection and Testing

Guidelines for Design of Test Procedures

The key steps in test design development are as follows (Binder, 2000):

1. Identify, model and analyze the responsibilities of the system under test;
2. Design test cases based on this external perspective;
3. Add test cases based on system/code analysis, suspicions, and heuristics;
4. Develop expected results for each test case or choose an approach to evaluate the pass/no pass status of each test case.

There needs to be:

1. A description of the expected behavior,
2. A way of observing it, and
3. A way for determining whether the observed behavior conforms with the expected behavior.

Inspection and Testing

Test Execution and Automation

Typically, test execution will involve the following steps (Binder, 2000):

1. Establish that the implementation under test is minimally operational by exercising the interfaces between its parts, components, and/or sub-systems.
2. Execute the test suite; the result of each test is evaluated and classified as pass or no pass.
3. Use a coverage tool to instrument the implementation under test. Rerun the test suite and evaluate the reported coverage.
4. If necessary, develop additional tests to exercise uncovered system functionality (or code).
5. Stop testing when the coverage goal is met and all tests pass.

A test automation streamlines (the many steps in) the testing procedure.

Inspection and Testing

Interpretation of Test Outcomes

When the tests are executed, we should be able to ...

> **... conclude with confidence whether or not the design concept, together with currently available technology, offer a viable solution to the requirements item.**

When a test fails, then ...

> **... we can say with complete confidence that a failure has been detected.**

But what can we say when a system passes a test?

At the very minimum, we can say that ...

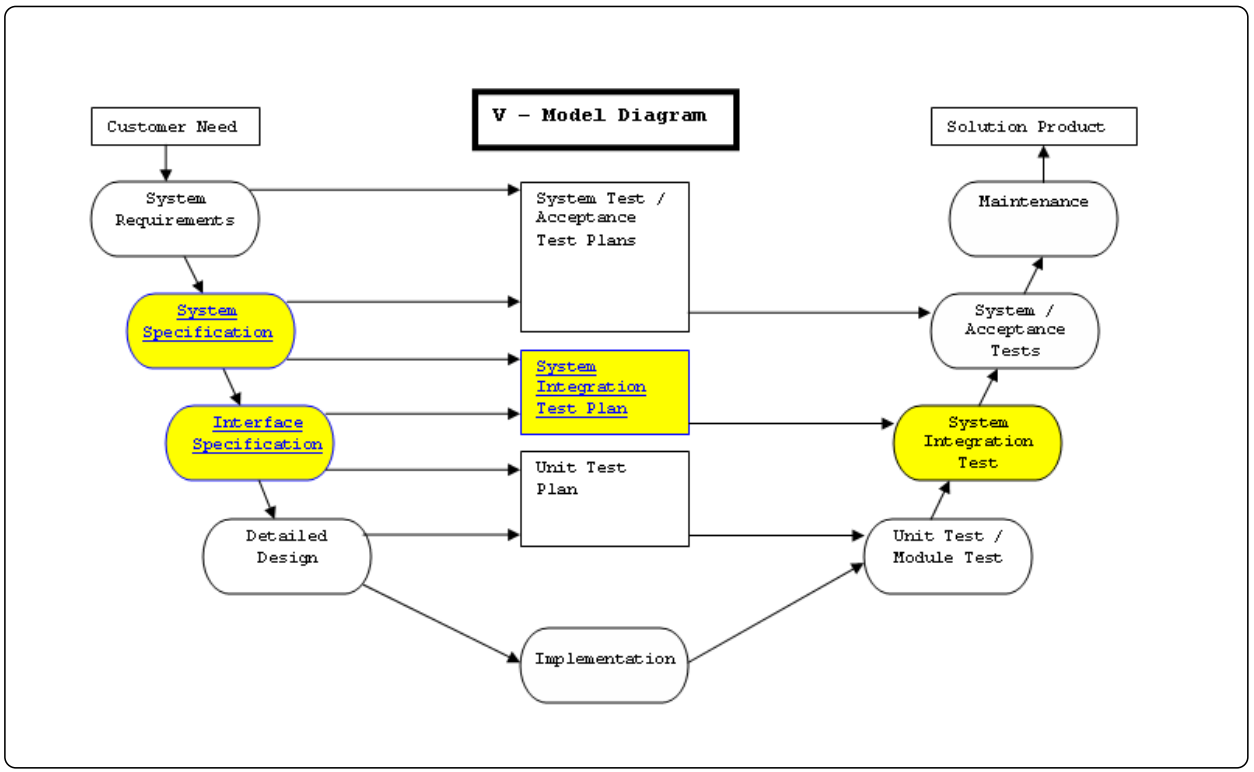
> **... the system passes the test for the specific specification (i.e., input data).**

The benefit of this observation can be small....

Integration Test Design

Integration Test in the V-Model of Development

Most large-scale software systems are built with components that must interoperate.



Integration Test Design

Integration Test Plan Questions

This strategy leads naturally to the following integration test plan questions (Binder, ch 13):

- What component interfaces will be the focus of the integration?

  Which sub-systems will be tested?

- In what sequence will the components and their interfaces be exercised?

- What stubs or drivers will be developed?

  A stub is a partial implementation of a component.

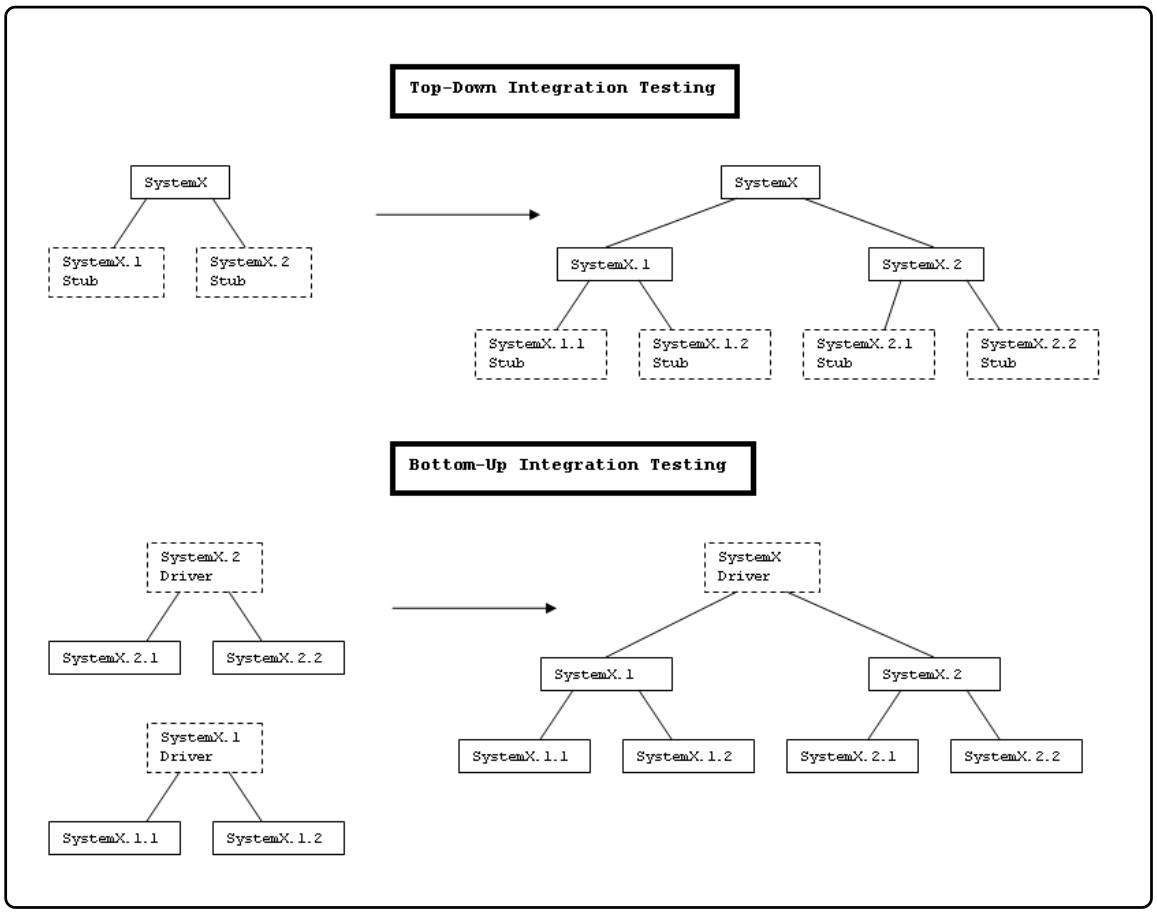
  A driver is a test program external to the component that will apply test cases to the component.

- What test pattern will be used?

- When will the integration testing be considered complete?

Top-Down versus Bottom-Up Testing

Finding the Right Approach ...



Top-Down versus Bottom-Up Testing

Comparison of Top-Down and Bottom-Up Integration Strategies

Criteria	Comparison
Architectural Validation	1. Top-down testing is better suited than bottom-up testing for early detection of system architecture errors and high-level design errors. 2. Early detection reduces the cost of fixing the errors.
System Demostration	1. A top-down approach to testing allows the organization to quickly gain confidence in a skeletal system that can then be used for demonstration purposes. 2. A bottom-up approach uses drivers at the highest system levels which would likely be more cumbersome to demonstrate.

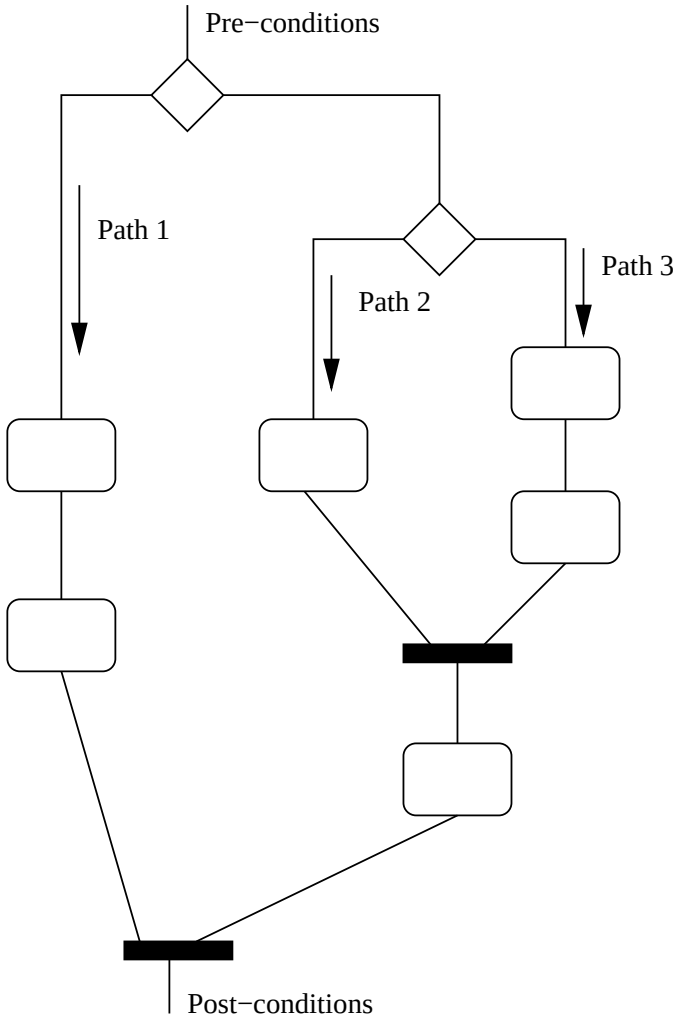
Top-Down versus Bottom-Up Testing

Comparison of Top-Down and Bottom-Up Integration Strategies

Criteria	Comparison
Test Implementation	1. Top-down testing will generally place more of a burden on the development team since meaningful stub behavior will be required for the system to be tested. Stubs can become quite complex. 2. Reusable components provide stable behavior and therefore developers do not need to be quite as creative when creating the drivers that drive those low-level components.
Test Observation	1. Top-down and bottom-up testing are about equal on this criteria. 2. Low-level components may need an artificial environment in order to probe their internal behavior.

Coverage-Based Testing

Exhaustive Enumeration of System Behavior Pathways



Coverage-Based Testing

Challenges in Coverage-Based Testing

- Complete coverage may not be possible because some pathways may not be finite (e.g., infinite loops).
- Existence of infeasible paths – that is, owing to inadvertent system design, system elements may never play an active role in the system behavior, no matter what input/data is selected.

Metrics for Coverage-Based Testing

- The percentage of system elements exercised by the test data;
- The percentage of system elements exercised under the application of “all the test data.”

Specification-Based Testing

Ideas and Approach

In specification based testing we ...

... produce a test suite on the basis of a specification.

The existence of a formal specification or model introduces the possibility ...

... of automating test generation, thus making test generation more efficient.

Test Criterion

Test suites are designed to a given property, a test criterion, e.g.,

- A notion of coverage.
- A fault domain.

Given a fault domain F, it is sometimes possible to produce a test suite that determines correctness relative to F: if the implementation under test really does behave like an element of F and it passes our test then it must be correct.

Specification-Based Testing

Parallel Modeling of System and Verification Requirements

Methodology (proposed by Adrian Marsh, 2004):

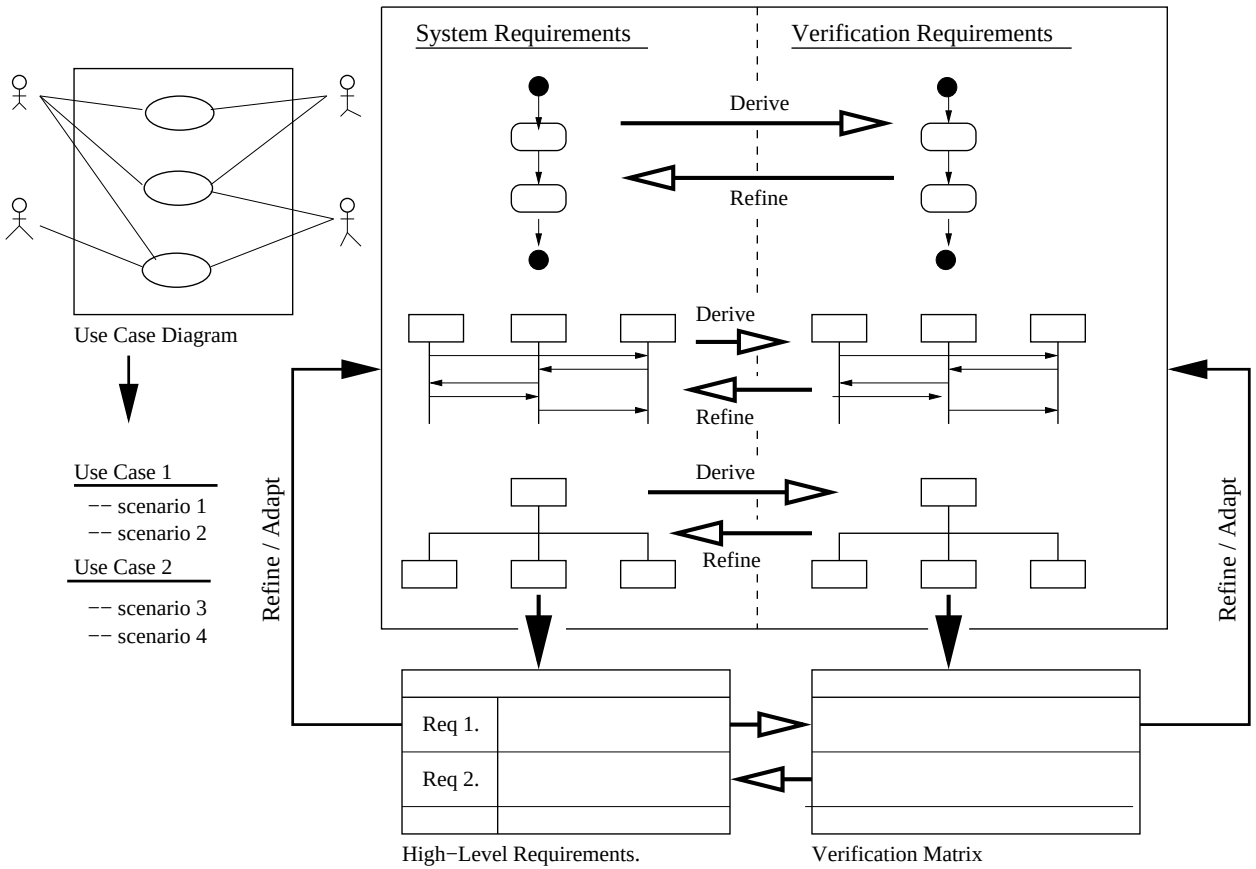
- UML models for system behavior and structure are constructed in parallel with descriptions of verification methods for the system.
- High-level requirements correspond to performance requirements on snippets of system behavior, interface requirements on system structure, and so forth.
- Verification requirements are described with:
  - Verification procedures presented in a tabular format.
  - Simplified UML-like diagrams for verification procedures at each step of system behavior or element of system structure.

Potential benefits include:

- The tight integration of system functionality/structure with verification procedures.
- Verification procedures are derived "directly from" requirements for expected system behavior/structure.

Specification-Based Testing

Parallel Modeling of System and Verification Requirements



Specification-Based Testing

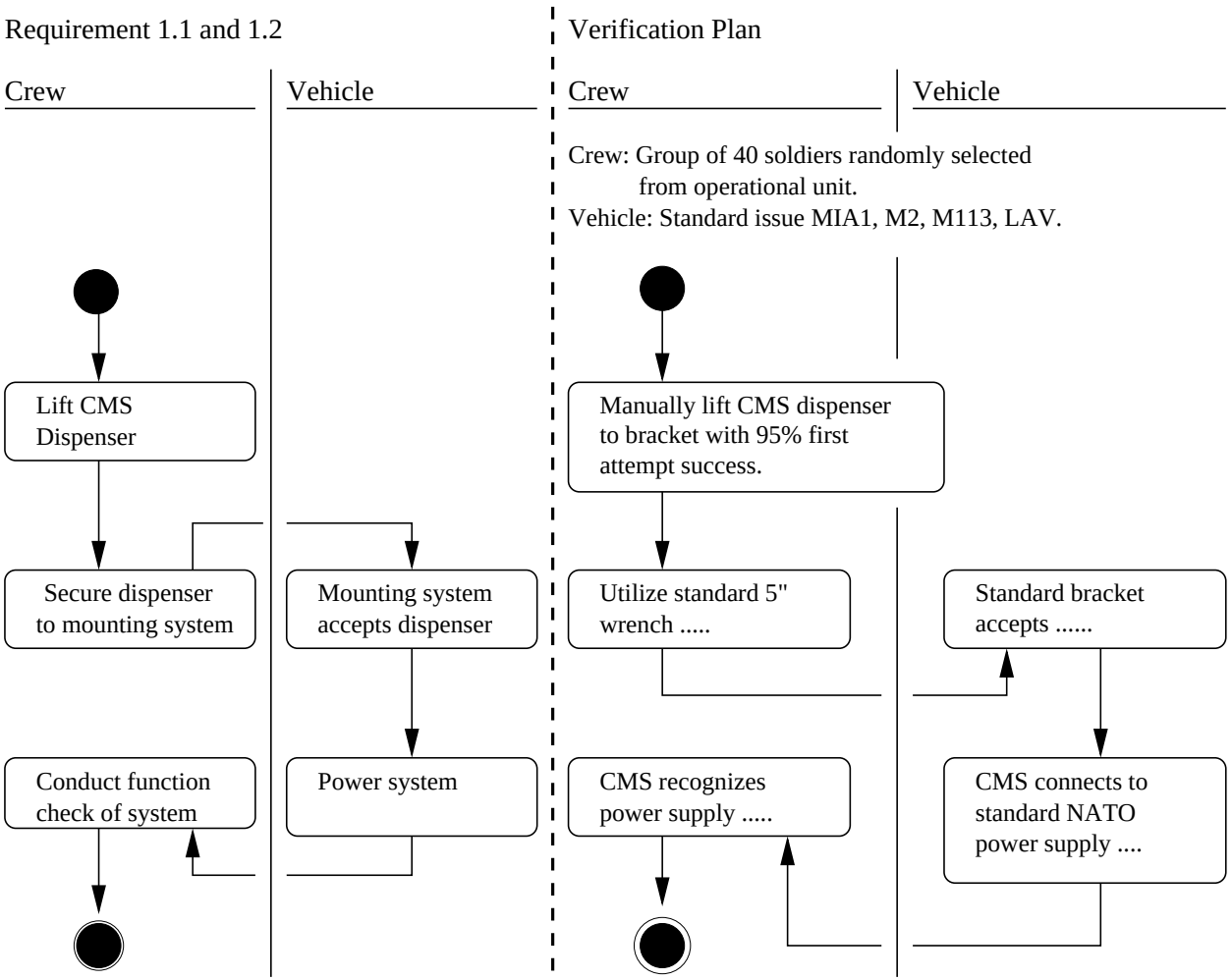
Simple Example

Requirements for interaction of Crew and Vehicle functionalities.....

Requirement	Description
Req. 1.1	The Crew shall be able to manually attach to the CMS dispenser.
Req. 1.2	The CMS shall be lifted by a 2 soldier crew.

We can create a side-by-side concurrent visualization of the requirement(s), activity diagram, and verification plan.

Specification-Based Testing



UML Diagram to Verification Matrix Checklist

Idea and Class Diagram Example

We need ...

... a checklist of things to test for in each type of UML diagram and, of course, this list will depend on the key concepts conveyed by each type of UML diagram.

UML Diagram views for System Structure and Key Concepts

Diagram	Key Concepts
Class Diagram	• Classes • Associations • Dependencies • Generalizations • Interfaces

UML Diagram to Verification Matrix Checklist

Sample Activity Diagram Checklist

- Test all activity nodes.
- Test all control of flow decision points.
- Test all “fork” and “join” points.
- Test all pathways through the activity diagram.

Sample Sequence Diagram Checklist

- Test all messages.
- Test all “conditional fragments” within an interaction.

Specification-Based Testing with DAS-BOOT

Background

DAS-BOOT (developed at UC Irvine) is an experimental specification-based tool for testing object-oriented systems. Typically,

> **...the "test coverage criterion" will define structures of the component under test that must be covered to satisfy the criterion.**

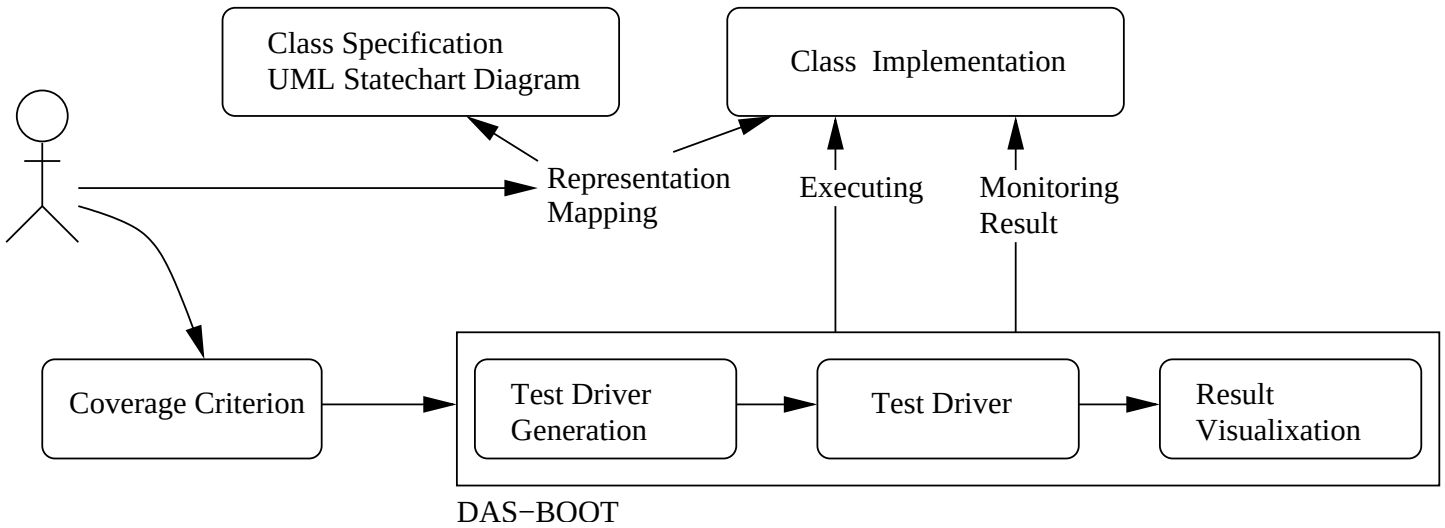
For example, "branch coverage" requires that every branch in the component under test be executed by at least one test case.

Current Prototype

- Implements improved specification-based coverage criteria suitable for testing object-oriented software systems whose behavioral specification is modeled as a finite state machine (FSM);
- Generates test cases, test drivers and embedded test oracles with little interaction required by the human tester.

Specification-Based Testing with DAS-BOOT

Flowchart for the DAS-BOOT Testing Process

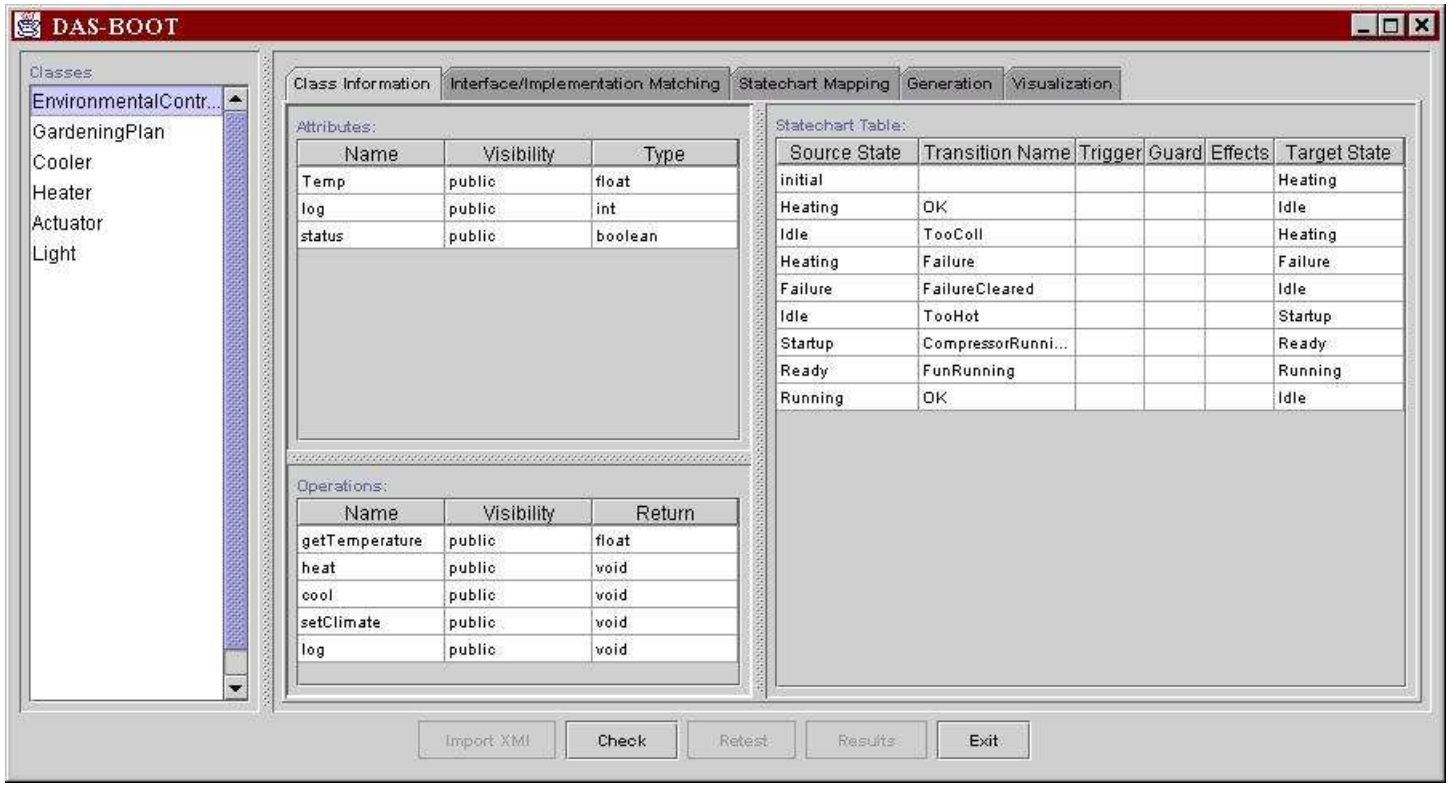


DAS-BOOT takes as input:

- A Java class to be tested,
- A Statechart specification of the class behavior, and
- A FSM-based test coverage criterion.

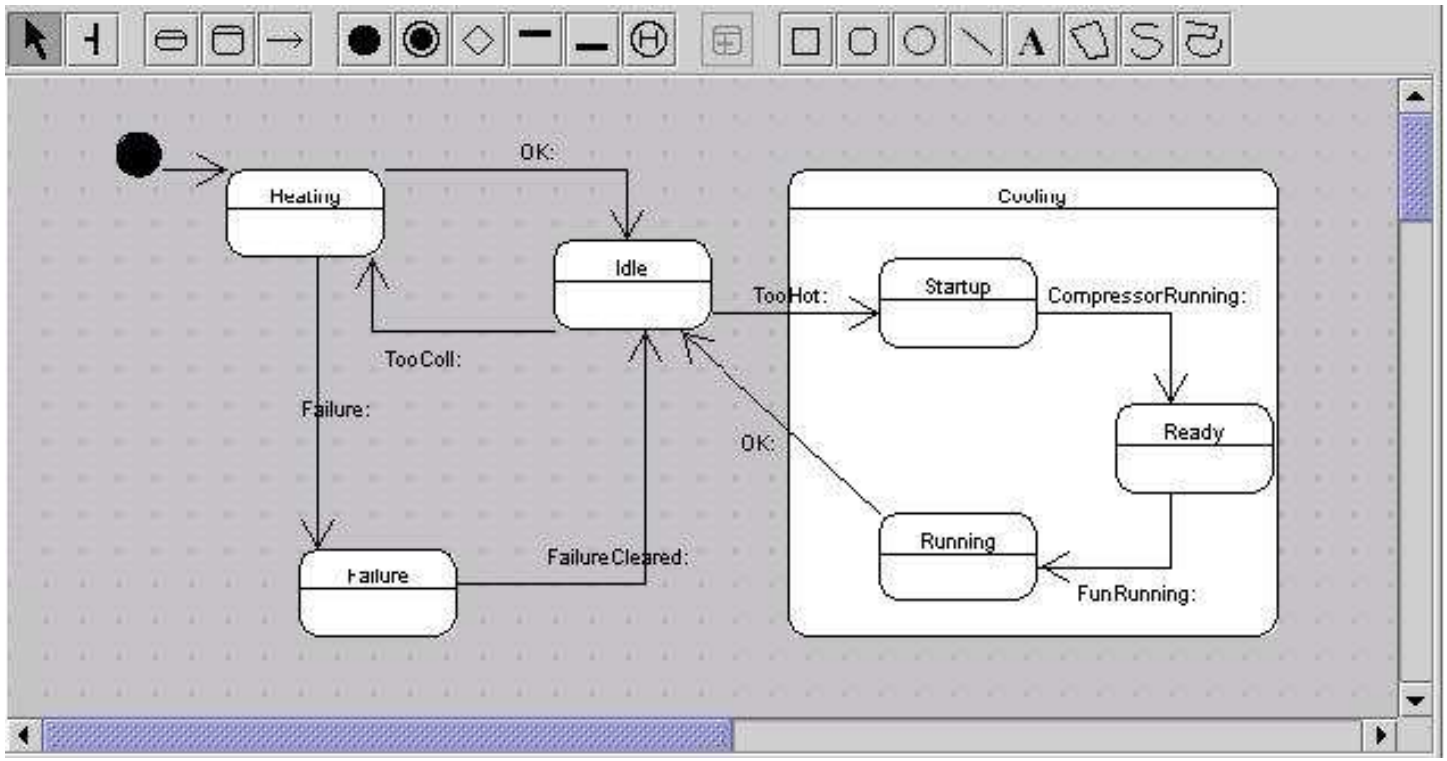
Specification-Based Testing with DAS-BOOT

Screenshots: Main window in DAS-BOOT.



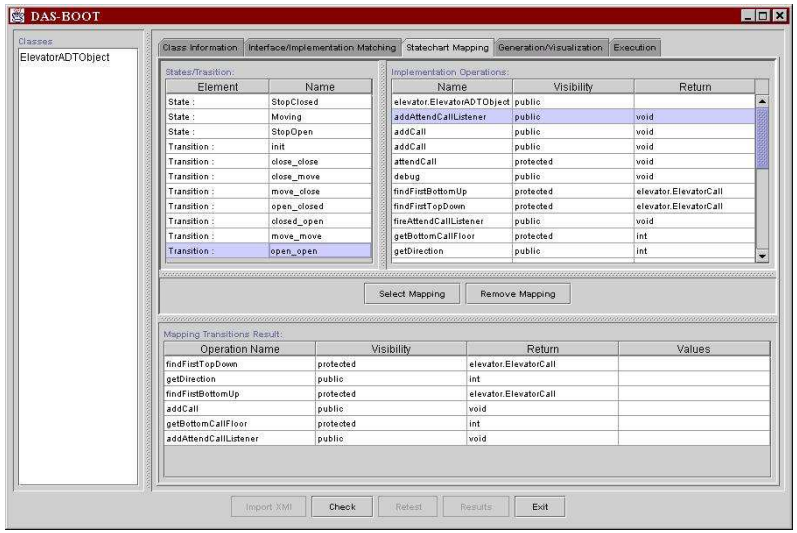
Specification-Based Testing with DAS-BOOT

Screenshots: Statechart corresponding to state-transition matrix.



Specification-Based Testing with DAS-BOOT

Screenshots: Statechart transitions, implementation operations, and mapping transitions.

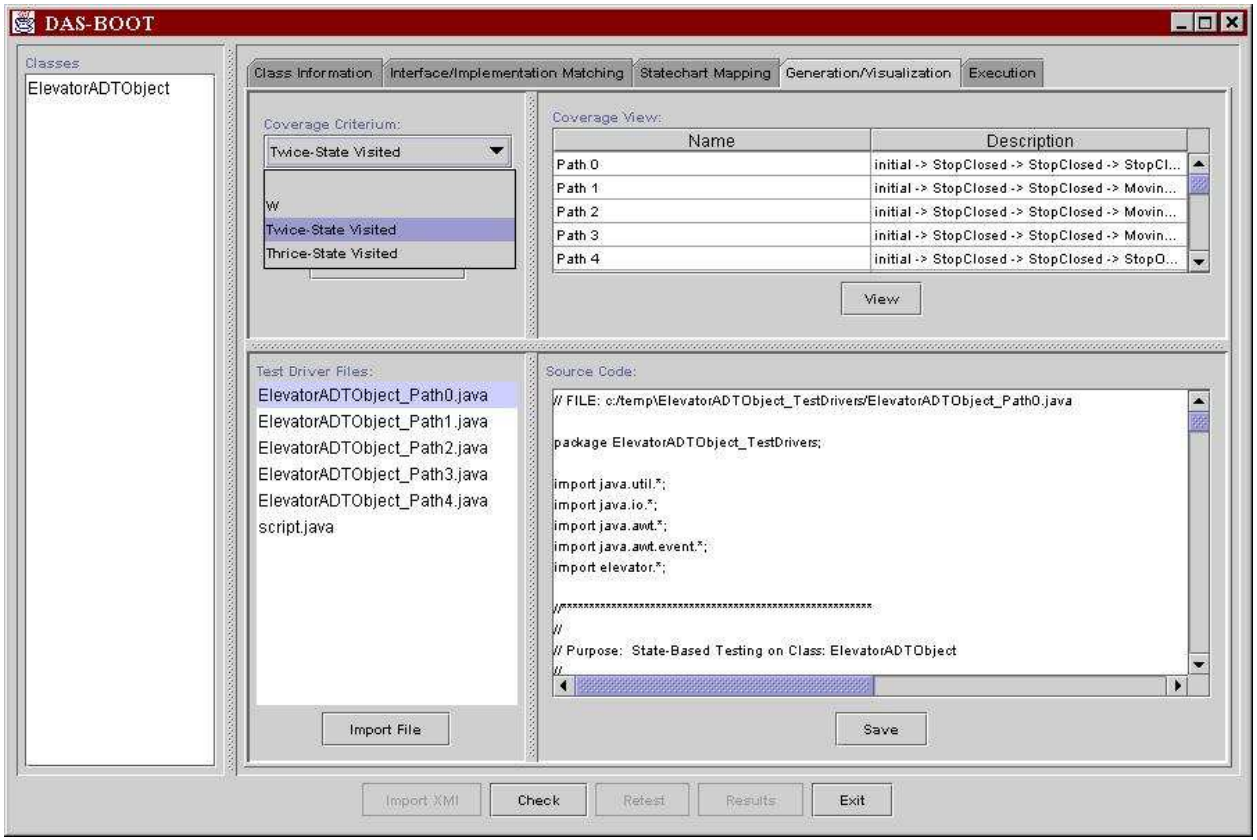


Representation mappings associate the following:

1. Statechart transitions with Java class operations(s).
2. Statechart states with Java class attribute(s).
3. Specific states of an object with ranges of attribute values.

Specification-Based Testing with DAS-BOOT

Screenshots: Execution and evaluation of test cases.



Role of Traceability in Validation/Verification

Traceability mechanisms ...

... support the capture and usage of trace data (i.e., to document, parse, organize, edit, interlink, change, and manage) requirements and traceability links between them.

Prior to the Development of Requirements

Two types of traceability mechanism are important:

- **Looking forward to requirements**

  When the stakeholder needs change (and/or assumptions on technology are modified), traceability mechanisms enable systems engineers to identify the requirements that need to be updated.

- **Looking backward from requirements**

  Looking backward from requirements helps the systems engineers and stakeholders understand the need for particular requirements.

Role of Traceability in Validation/Verification

Traceability Mechanisms linking Requirements to the System Design

- **Looking forward from requirements**

  Mechanisms connect requirements to individual sub-systems/components.

- **Looking backward to requirements**

  Mechanisms connect individual sub-systems/components to requirements.

Early Validation of Requirements

Early validation of requirements ...

... improves the likelihood that the stakeholders' needs have been properly understood and accounted for in the high-level requirements.

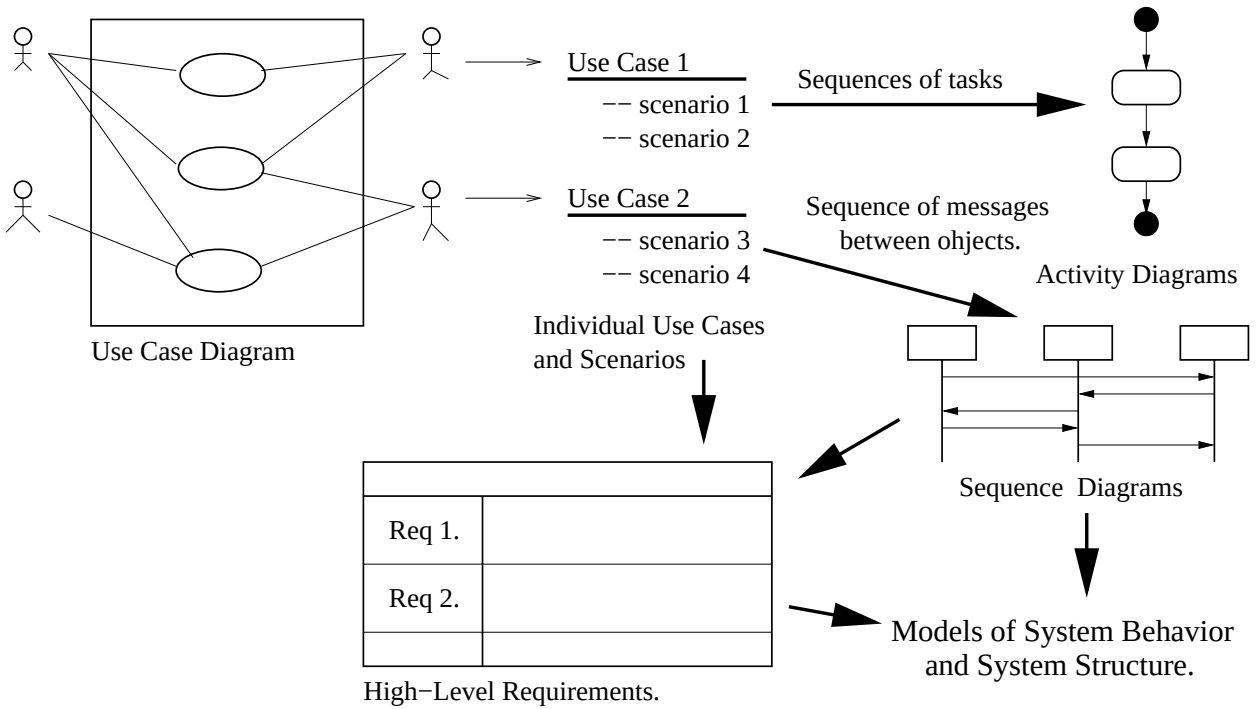
Traceability mechanisms from goals/scenarios through to high-level requirements ...

... help the stakeholders to see how their needs have been transformed into requirements (and before the expensive production begins).

Role of Traceability in Validation/Verification

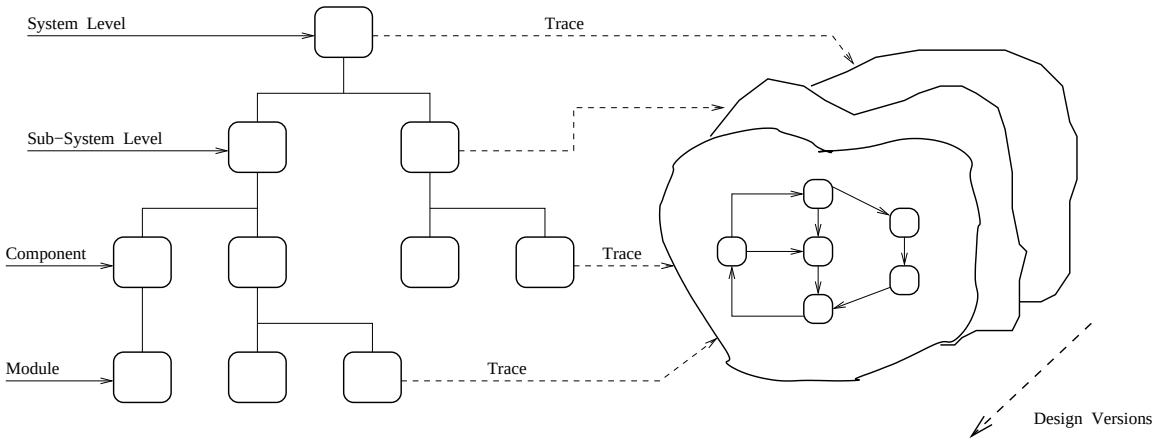
Phase 1: From Goals/Scenarios to Requirements

Pathway from use cases to scenarios, activity diagrams, system objects, and sequence diagrams.



Role of Traceability in Validation/Verification

Phase 2: From Requirements to System Architectures and Physical Design



Product-related traceability is supported by two types of traceability link:

- **Satisfies Links**

  Ensures requirements are satisfied by the system.

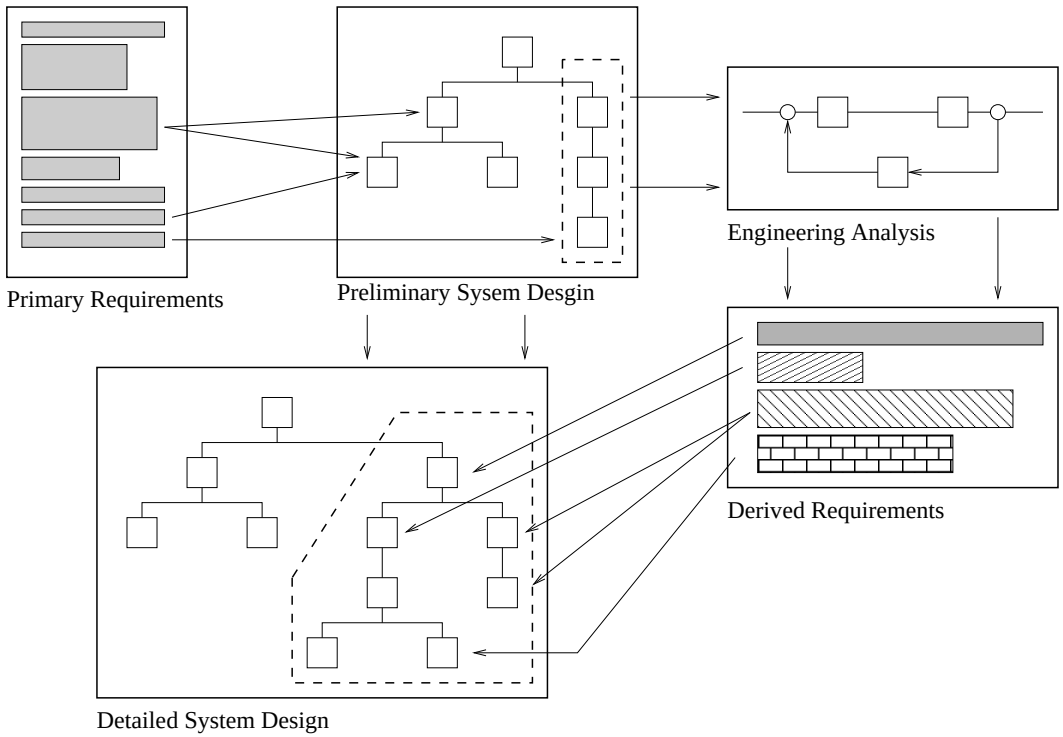
- **Dependency Links**

  Manage dependencies among objects.

Role of Traceability in Validation/Verification

Phase 3: From Requirements to Derived Requirements

Engineering analyses are conducted to gain insight into the likely behavior of a system.



These estimates of performance are, in effect, derived requirements for expected system performance.

Writing Verification Requirements

Definition

Verification requirements define ...

> **... the method for determining whether or not a particular (design) requirement has been fully complied with in the design.**

For each design requirement, there should be ...

> **...one or more verification requirements and ideally, both should be written at the same time by engineers who have the specialized design and verification knowledge.**

This strategy of requirements development ...

> **... reduces the likelihood of vague requirements because it forces the author to articulate how the design requirement will be quantitatively evaluated.**

Writing Verification Requirements

Example (from O'Grady, 1998)

Suppose that a weight requirement is:

Design Requirement	Item	Description
Req. 1.1	Weight	Weight of the item shall be less than or equal to 134 pounds.

Writing Verification Requirements

Example (from O'Grady, 1998)

The verification requirement could be:

Verification Requirement	Item	Description
Req. 2.1	**Weight**	The item weight shall be determined by a scale, the calibration for which is correct, with an accuracy of plus or minus 6 ounces. The item shall be placed on the scale located on a level, stable surface and a reading taken. The measured weight shall be less than 134 pounds and 11 ounces.

Writing Verification Requirements

Guidelines for Writing Good Verification Requirements

- Is the requirement stated correctly?

  In addition to correct use of the language, the requirement should avoid using unnecessarily difficult and overly specialized words.

- Is the requirement unambiguous?

  The requirement should be perfectly clear so that multiple people cannot have multiple interpretations of its meaning.

- Are the values of the requirement correct?

Design-Requirements to Verification-Requirements Traceability

In addition:

- Traceability mechanisms are needed to connect the “design requirements” to the “verification requirements” and associated compliance verification procedures.

Verification Traceability Matrices

Sample Verification Traceability Matrix

Design Requirement	Verification Method				Verification Requirement	Level of Application
	Test	Analysis	Demo	Exam		
Req 1.1 ...	X	...	...	...	....	...
Req 1.2 ...	...	...	X	...	....	...
Req 1.3 ...	...	X	...	...	....	...

Points to note:

- The traceability matrix coordinates (design) requirements verification methods, textual descriptions of the verification requirements, and the levels at which the verification action will be applied.
- Appropriate verification methods for each requirement are indicated by an X.

Verification Traceability Matrices

Testing

- Evidence that stated requirements have been met is established by subjecting the system item (or a mathematical representation of the system) to a series of planned simulations.
- These simulations will be based on scientific principles.
- Their execution may involve test equipment.
- Performance is quantitatively measured in response to external stimuli applied to the system under consideration.

Analysis

- Evidence that stated requirements have been met is established through the use of technical or mathematical models or simulations, algorithms, equations, charts, diagrams ... derived from scientific principles.
- At each level of system abstraction, appropriate features of behavior are studied to see if they comply with the required characteristics.

Verification Traceability Matrices

Demonstration

- Evidence that stated requirements have been met is established through the use of technical or mathematical models or simulations, algorithms, equations, charts, diagrams ... derived from scientific principles.
- At each level of system abstraction, appropriate features of behavior are studied to see if they comply with the required characteristics.

Examination

- Usually, examination is a non-destructive form of inspection that determines if the characteristics of system items satisfy the non-functional requirements (e.g,. size, position. color ..etc).
- Standard procedures include visual inspection, supported by simple instruments (e.g. a ruler).

Test-Driven Development

Motivation

- Standard approaches to system development emphasize development of requirements and system design alternatives.
- Validation/verification comes after these entities are in place.

Test-driven development procedures have their home in software development and are based on a very simple tenet (Koskala, 2008):

> **... you only ever write code to fix failing tests.**

Test-Driven Development

Traditional versus Test-Driven Development Cycles

Tradtional Approach to System Development

Test−driven Development Cycle

This procedure puts the notion of design on an uncomfortable footing.

Thus, instead of talking about design, we talk about ...

... refactoring to better convey the idea that a current design is transformed toward a better design.

References

- Binder R.V., **Testing Object-Oriented Sytsems**, Addison-Wesley, 2000.
- DAS-BOOT: Design and Specification-Based Object-Oriented Testing. For details, see http://www.isr.uci.edu/flyers/pdf/DAS-BOOT.pdf.
- Koskala L., **Test-Driven: Practical TDD and Acceptance TDD for Java Developers**, Manning Publications, 2008.
- Marsh A., Visualizing Verification, Scholarly Paper, Master of Science in Systems Engineering, University of Maryland, College Park, April 2004.
- Martin J.N., Overview of the EIA 632 Standard - Processes for Engineering a System, Chairman of the EIA 632 Working Group, Raytheon Systems, 2003.
- O'Grady J.O., **System Validation and Verification**, CRC Press, 1998.