



Software Testing

Outline

- Terminology
- Testing Activities
- Types of Testing

Famous Problems

- F-16 : crossing equator using autopilot
 - Result: plane flipped over
 - Reason?
 - Reuse of autopilot software from a rocket



- The Therac-25 accidents (1985-1987), quite possibly the most serious non-military computer-related failure ever in terms of human life (at least five died)
 - Reason: Concurrent programming error
- NASA Mars Climate Orbiter destroyed due to incorrect orbit insertion (September 23, 1999)
 - Reason: Unit conversion problem

Terminology

- **Failure:**
 - Any deviation of the observed behavior from the specified behavior
- **Erroneous state (“error”):**
 - The system is in a state such that further processing by the system can lead to a failure
- **Fault (“bug” or “defect”) :**
 - The mechanical or algorithmic cause of an error
- **Validation:**
 - Activity of checking for deviations between the **observed behavior** of a system and its **specification**

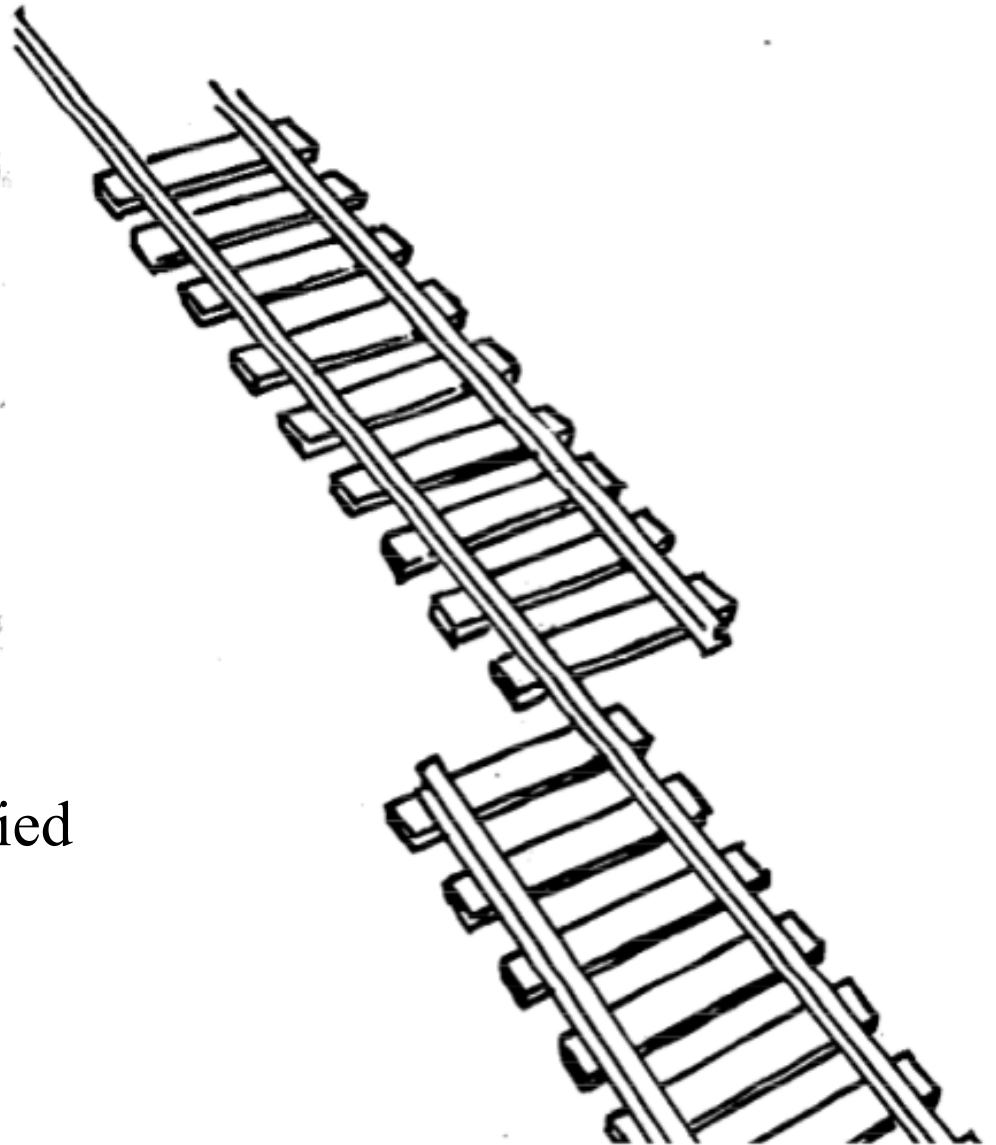
What is this?

A failure?

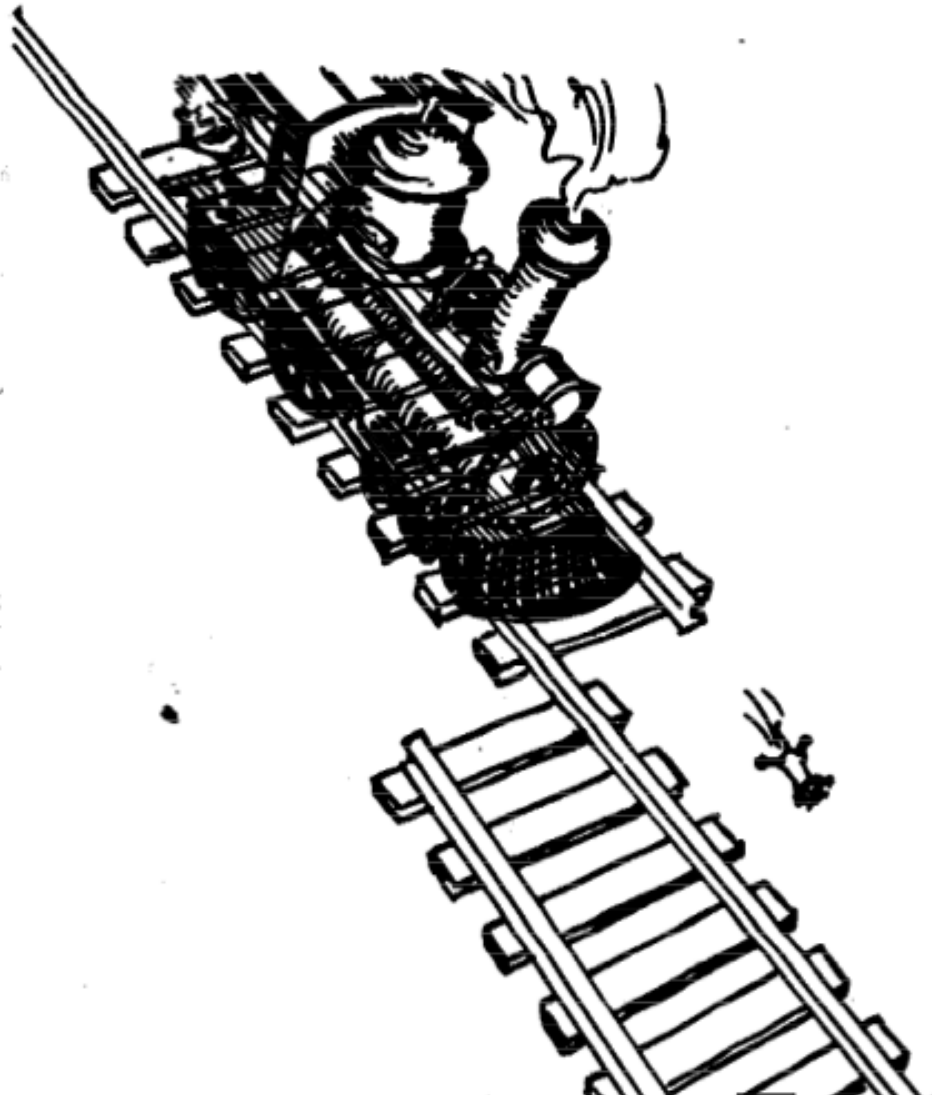
An error?

A fault?

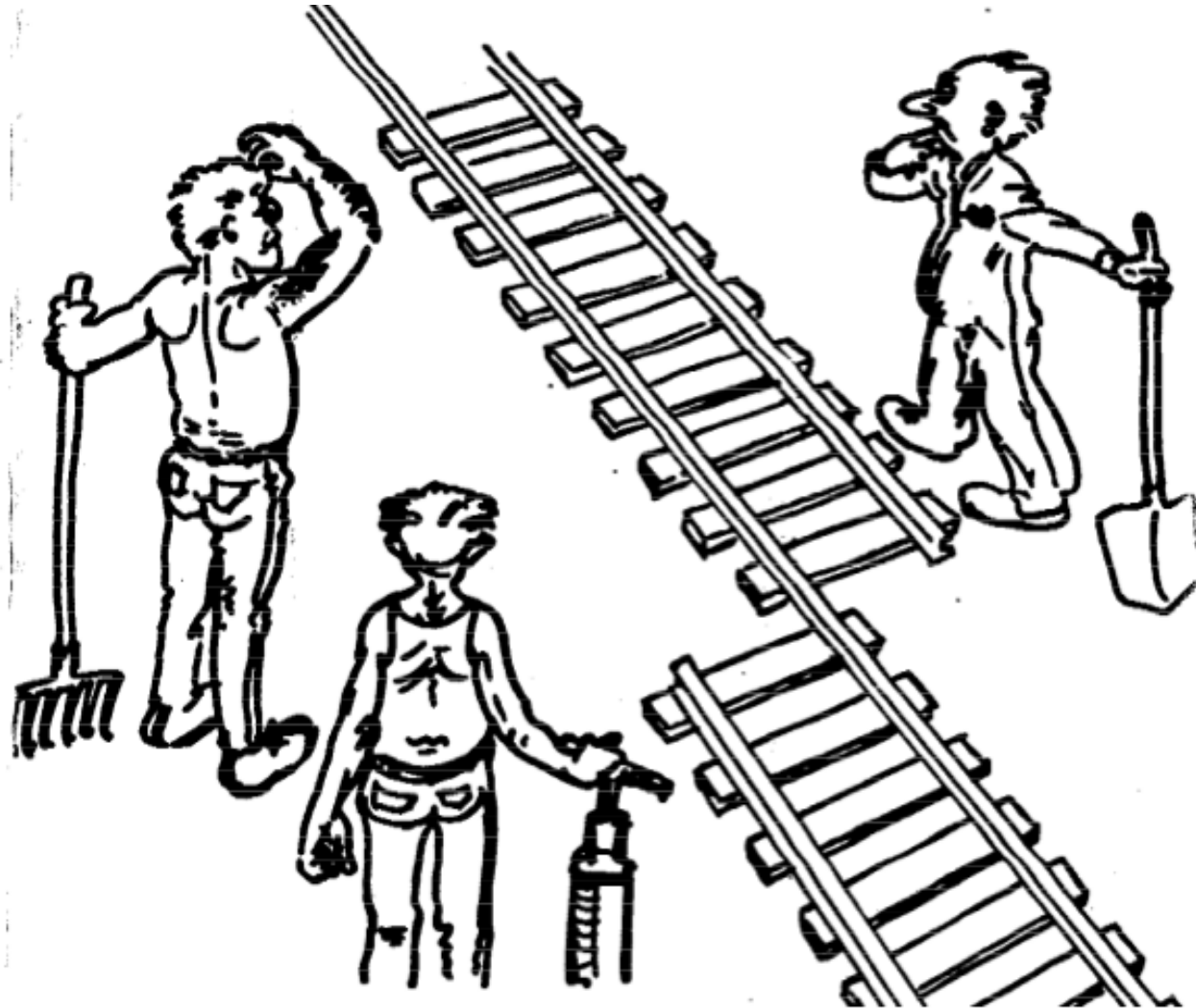
We need to describe specified
and desired behavior first!



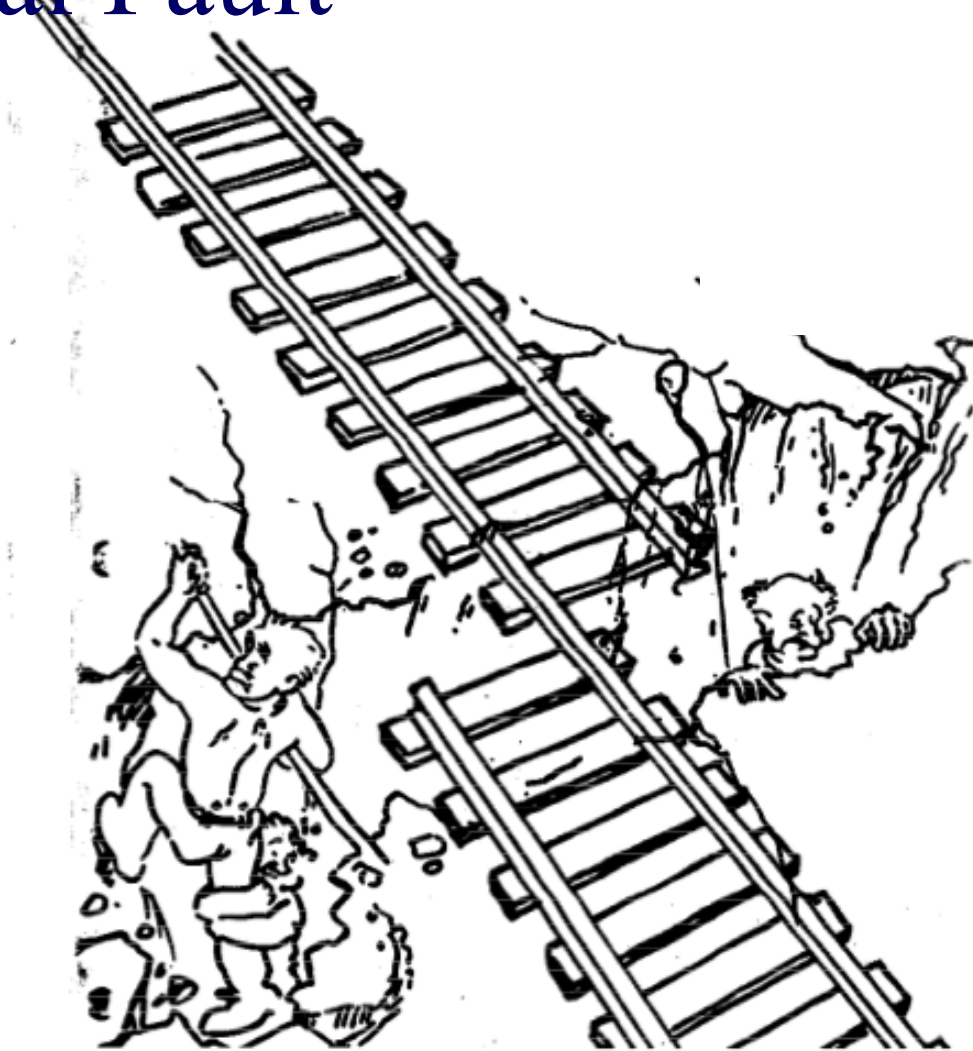
Erroneous State (“Error”)



Algorithmic Fault



Mechanical Fault

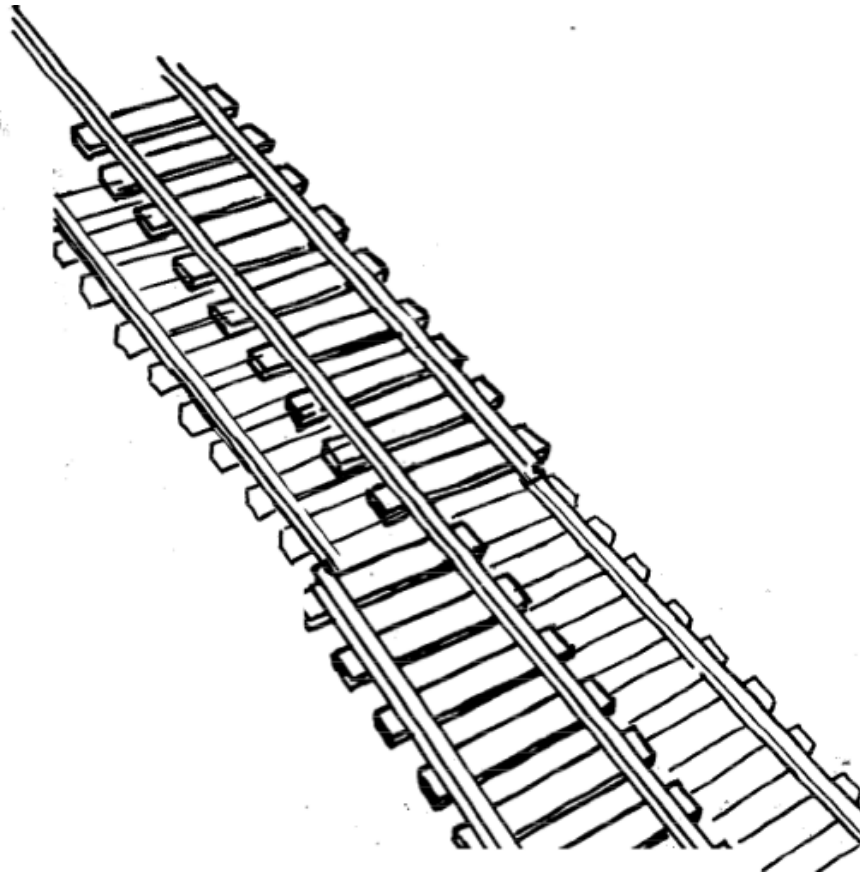


Examples of Faults and Errors

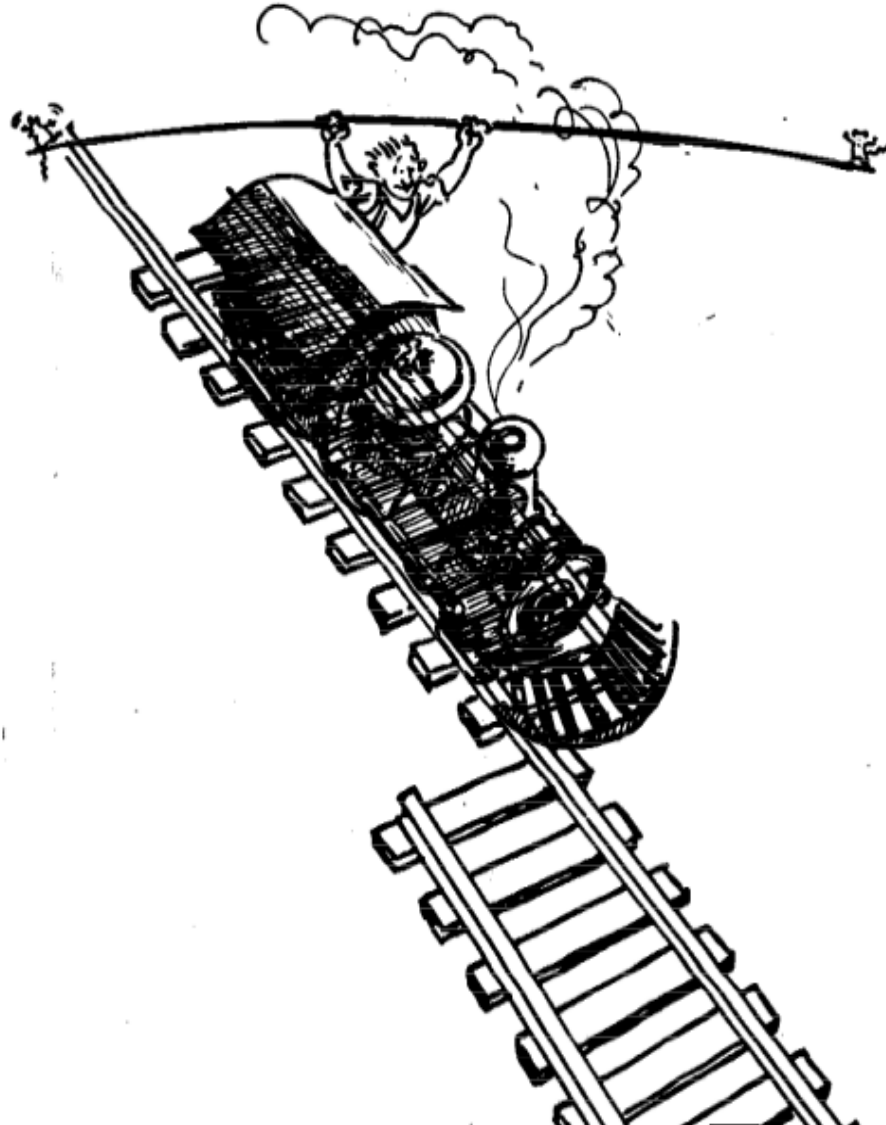
- Faults in the Interface specification
 - Mismatch between what the client needs and what the server offers
 - Mismatch between requirements and implementation
- Algorithmic Faults
 - Missing initialization
 - Incorrect branching condition
 - Missing test for null
- Mechanical Faults (very hard to find)
 - Operating temperature outside of equipment specification
- Errors
 - Null reference errors
 - Concurrency errors
 - Exceptions

How do we deal with Errors, Failures and Faults?

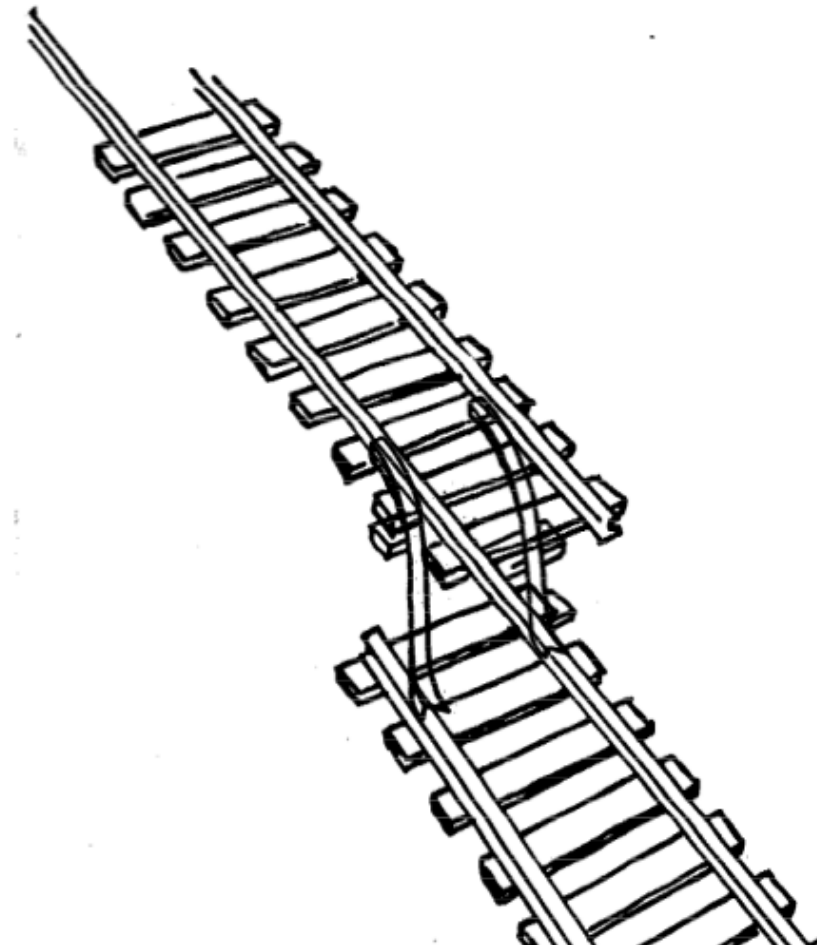
Modular Redundancy



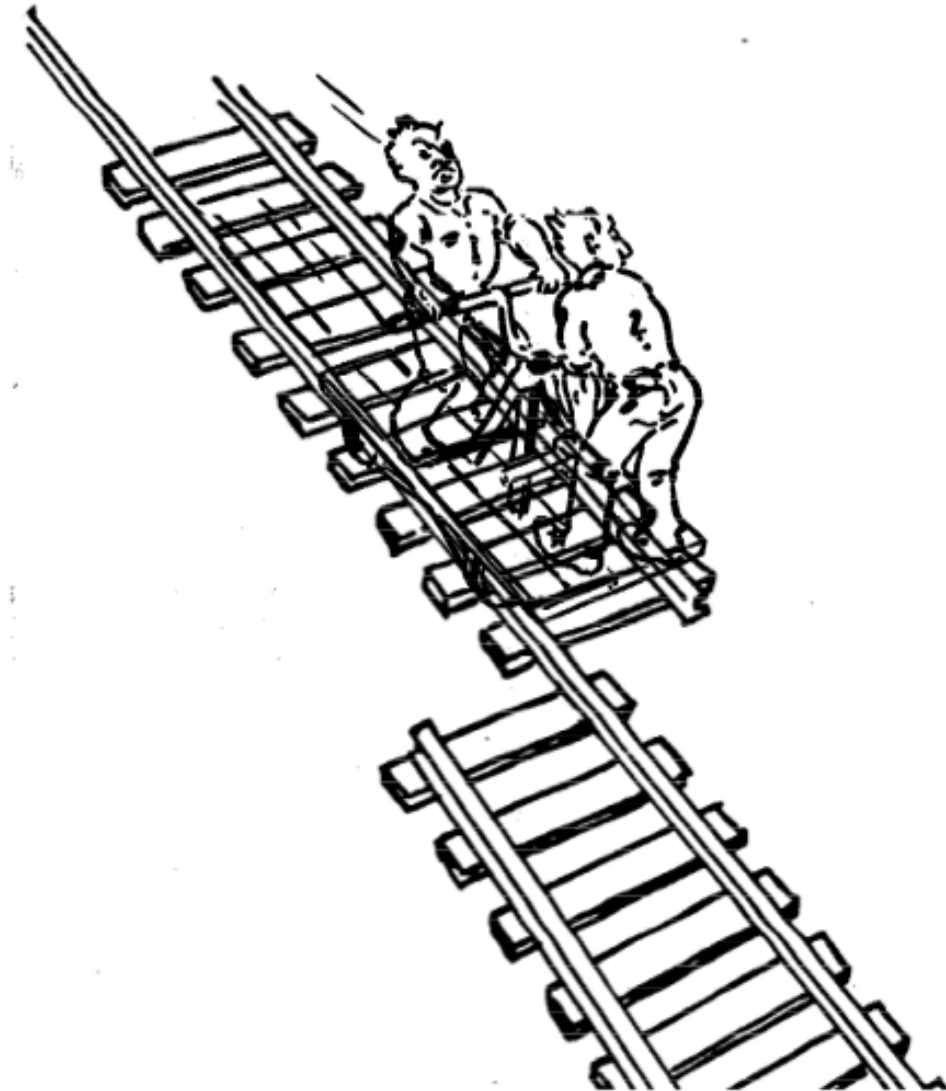
Declaring the Bug as a Feature



Patching



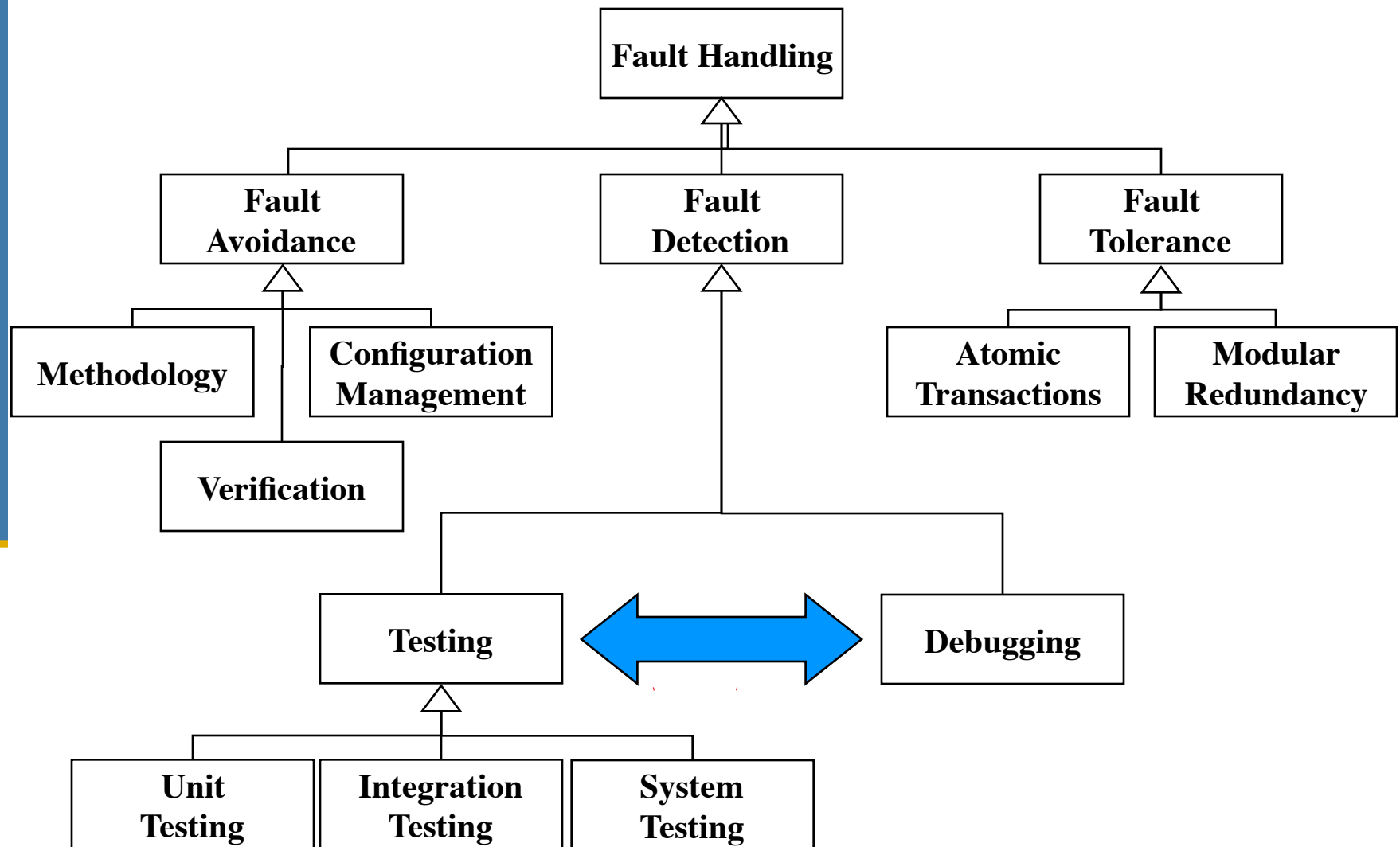
Testing



Another View on How to Deal with Faults

- Fault avoidance (before system is released)
 - Use methodology to reduce complexity
 - Use configuration management to prevent inconsistency
 - Apply verification to prevent algorithmic faults
 - Use Reviews
- Fault detection (while system is running)
 - *Testing*: Activity to provoke failures in a planned way
 - *Debugging*: Find and remove faults
 - *Monitoring*: Deliver information about state (used during debugging)
- Fault tolerance (recover from failure once the system is released)
 - Exception handling
 - Modular redundancy

Taxonomy for Fault Handling Techniques



Observations

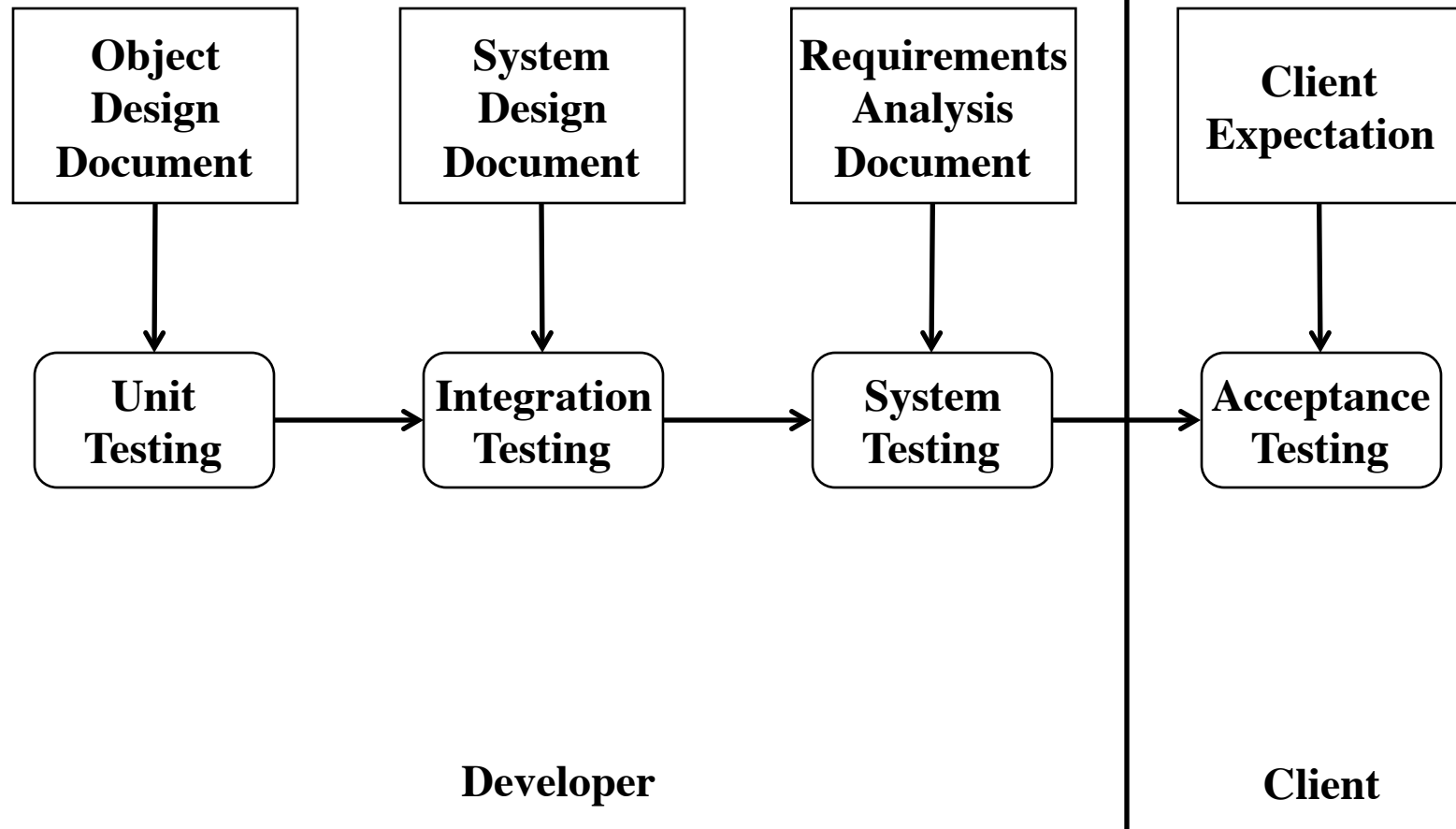
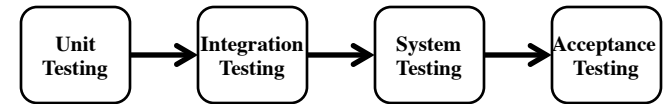
- It is impossible to completely test any nontrivial module or system
 - Practical limitations: Complete testing is prohibitive in time and cost
 - Theoretical limitations: e.g. Halting problem
- “Testing can only show the presence of bugs, not their absence” (Dijkstra).
- Testing is not for free

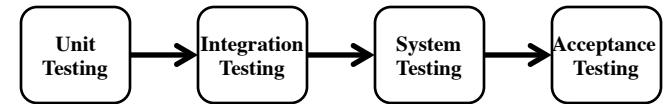
=> Define your goals and priorities

Testing takes creativity

- To develop an effective test, one must have:
 - Detailed understanding of the system
 - Application and solution domain knowledge
 - Knowledge of the testing techniques
 - Skill to apply these techniques
- Testing is done best by independent testers
 - We often develop a certain mental attitude that the program should behave in a certain way when in fact it does not
 - Programmers often stick to the data set that makes the program work
 - A program often does not work when tried by somebody else.

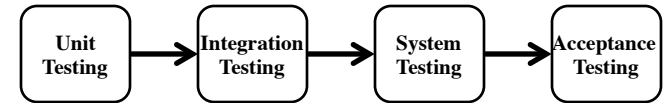
Testing Activities





Types of Testing

- **Unit Testing**
 - Individual component (class or subsystem)
 - Carried out by developers
 - Goal: Confirm that the component or subsystem is correctly coded and carries out the intended functionality
- **Integration Testing**
 - Groups of subsystems (collection of subsystems) and eventually the entire system
 - Carried out by developers
 - Goal: Test the interfaces among the subsystems.



Types of Testing continued...

- **System Testing**
 - The entire system
 - Carried out by developers
 - Goal: Determine if the system meets the requirements (functional and nonfunctional)
- **Acceptance Testing**
 - Evaluates the system delivered by developers
 - Carried out by the client. May involve executing typical transactions on site on a trial basis
 - Goal: Demonstrate that the system meets the requirements and is ready to use.