Announcements

- Change of office hours: M,W 5-6pm
- Please submit your CV through the course webpage (non-graded assignment)
- Ice Cream Social tomorrow 3pm-5pm in ETRL Courtyard
- Interested in participating in an experiment?
  - What is expected from you: Read and understand Java code
  - What do you get in return: A Starbucks giftcard ($15)
  - Fill the eligibility survey: tinyurl.com/n2mcwjy

Cpt S 422: Software Engineering Principles II

Testing Fundamentals

Dr. Venera Arnaoudova

WASHINGTON STATE UNIVERSITY

Nature of Software Development

- Development, not production
- Human intensive
- Engineering, but also social process
- More and more complex software systems
- Pervasive in an increasing number of industries

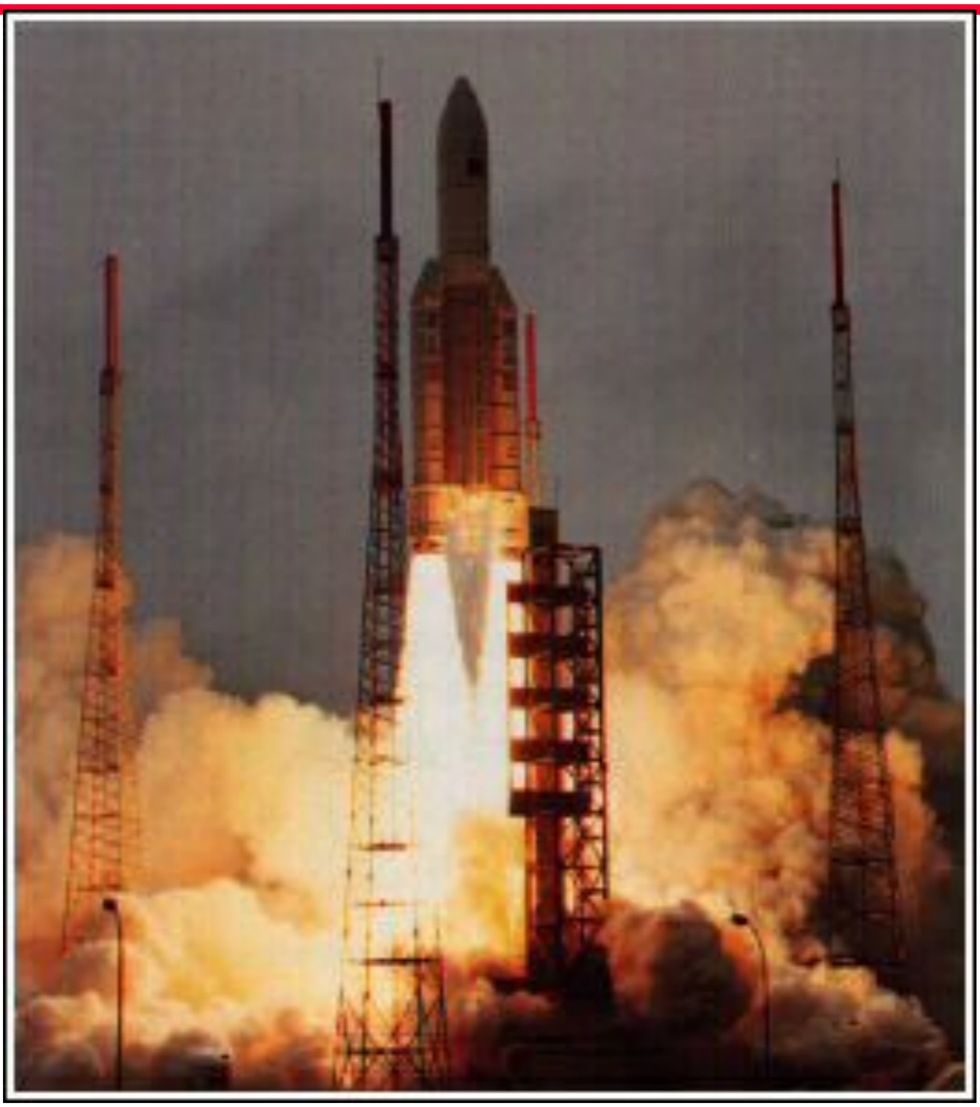
Qualities of Software Products

- Correctness
- Reliability
- Robustness
- Performance
- User Friendliness
- Verifiability
- Maintainability
- Repairability
- Evolvability
- Reusability
- Portability
- Understandability
- Interoperability

Pervasive Problems

- Software is commonly delivered late, way over budget, and of unsatisfactory quality
- Software validation and verification are rarely systematic and are usually not based on sound, well-defined techniques
- Software development processes are commonly unstable and uncontrolled
- Software quality is poorly measured, monitored, and controlled.

Ariane 5, v501 – ESA Launcher



Ariane 5 Press Release, June 96

- **Paris, 4 June 1996, ESA/CNES Joint Press Release, ARIANE 501**

  The first launch of Ariane 5 did not result in validation of Europe's new launcher. It was the first flight test of an entirely new vehicle each of whose elements had been tested on the ground in the course of the past years and months.

  Of an entirely new design, the launcher uses engines ten times as powerful as those of the Ariane-4 series. Its electronic brain is a hundred times more powerful than that used on previous Ariane launchers. The very many qualification reviews and ground tests imposed extremely tough checks on the correctness of all the choices made. There are, however, no absolute guarantees. **A launcher's capability can be demonstrated only in flight under actual launch conditions.**

  A second test already scheduled under the development plan will take place in a few months' time. Before that, everything will have to be done to establish the reasons for this setback and make the corrections necessary for a successful second test. An inquiry board will be set up in the next few days. It will be required to submit, by mid-July, an entirely independent report identifying the causes of the incident and proposing modifications designed to prevent any further incidents.

  Ariane 5 is a major challenge for space activities in Europe. The skills of all teams involved in the programme, coupled with the determination and solidarity of all the political, technical and industrial authorities, make us confident of a successful outcome.

Ariane 5 – Root Cause

- Source: ARIANE 5 Flight 501 Failure, Report by the Inquiry Board

  A program segment for converting a floating point number to a signed 16 bit integer was executed with an input data value outside the range representable by a signed 16 bit integer. This run time error (out of range, overflow), which arose in both the active and the backup computers at about the same time, was detected and both computers shut themselves down. This resulted in the total loss of attitude control. The Ariane 5 turned uncontrollably and aerodynamic forces broke the vehicle apart. This breakup was detected by an on-board monitor which ignited the explosive charges to destroy the vehicle in the air. Ironically, the result of this format conversion was no longer needed after lift off.

Ariane 5 – Lessons Learned

- Rigorous reuse procedures, including *usage-based testing (based on operational profiles)*

- Adequate exception handling strategies (backup, degraded procedures?)

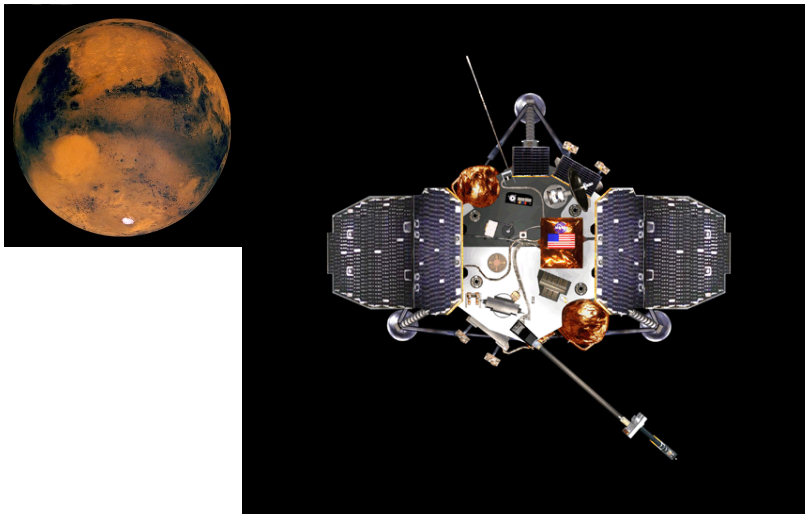
- Clear, complete, documented specifications (e.g., preconditions, post-conditions)

- Note this was not a complex, computing problem, but a deficiency of the software engineering practices in place …

Other examples of SE Failures

- Therac-25: radiation therapy and X-ray machine killed several patients. Cause: unanticipated, non-standard user inputs.

- NASA mission to Mars (Mars Climate Orbiter Spacecraft, 1999): Incorrect conversion from imperial→metric leads to crash



Consequences of Poor Quality

- Standish Group surveyed 350 companies, over 8000 projects, in 1994
    - 31% cancelled before completed, 9-16% were delivered within cost and budget
- US study (1995): 81 billion dollars are spent per year for failing software development projects

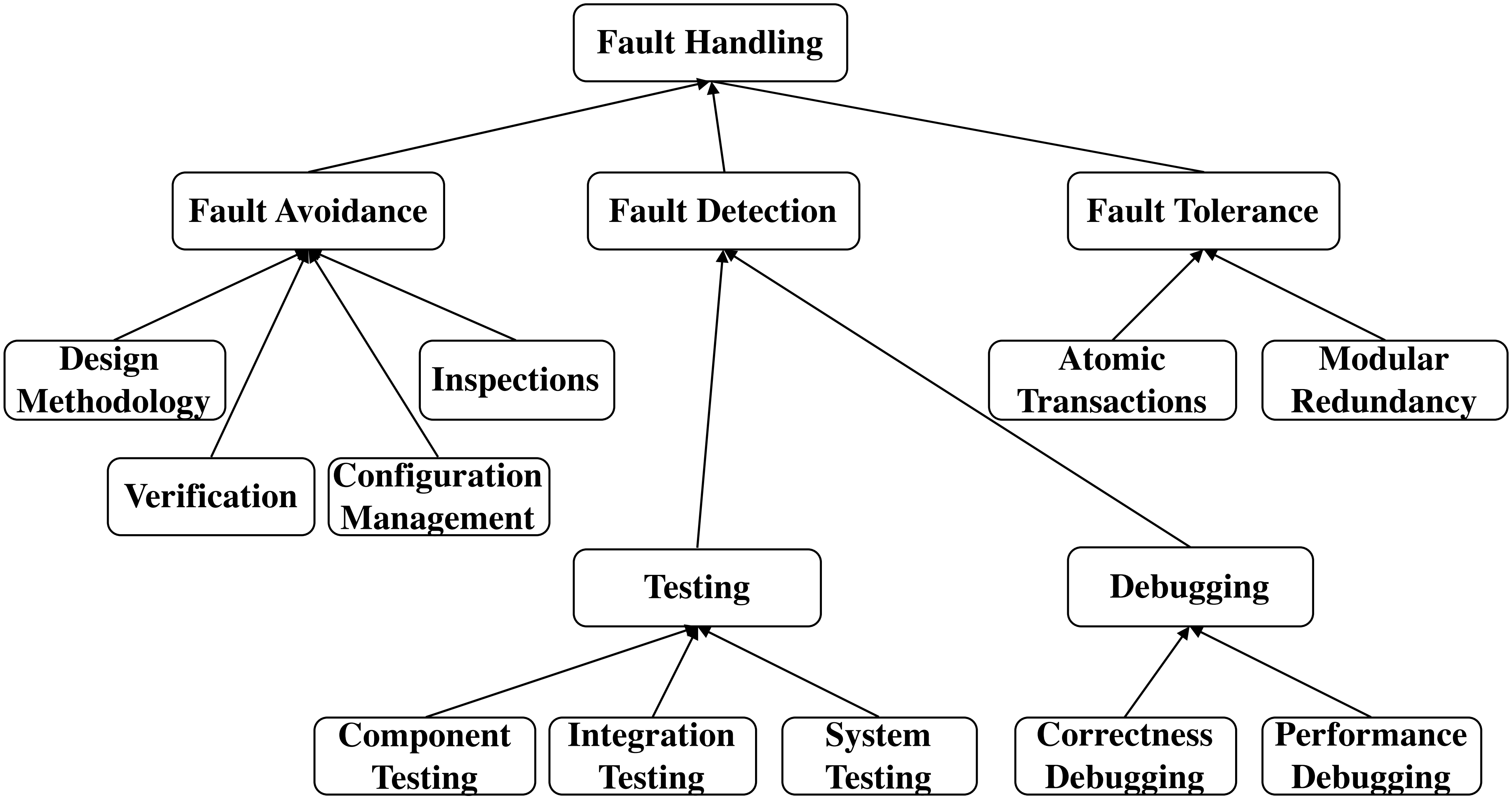
NIST Study

- According to a study from NIST (May 2002), software bugs and glitches cost the U.S. economy about $59.5 billion a year.
- The study also found that better testing could expose the bugs and remove bugs at the early development stage and this could reduce the cost by about $22.2 billion.
- See the report online …

Key Quality Assurance (QA) Capabilities

- Uncover faults in the documents *where they are introduced*, in a systematic way, in order to avoid ripple effects. *Systematic, structured reviews* of software documents are referred to as *inspections*.
- Derive, in a *systematic* way, effective test cases to uncover faults
- Automate *testing* and *inspection* activities, to the maximum extent possible
- Monitor and control *quality,* e.g., reliability, maintainability, safety, across *all* project phases and activities
- All this implies the *measurement* of SW products and processes and the *empirical evaluation* of testing and inspection technologies

Dealing with software faults

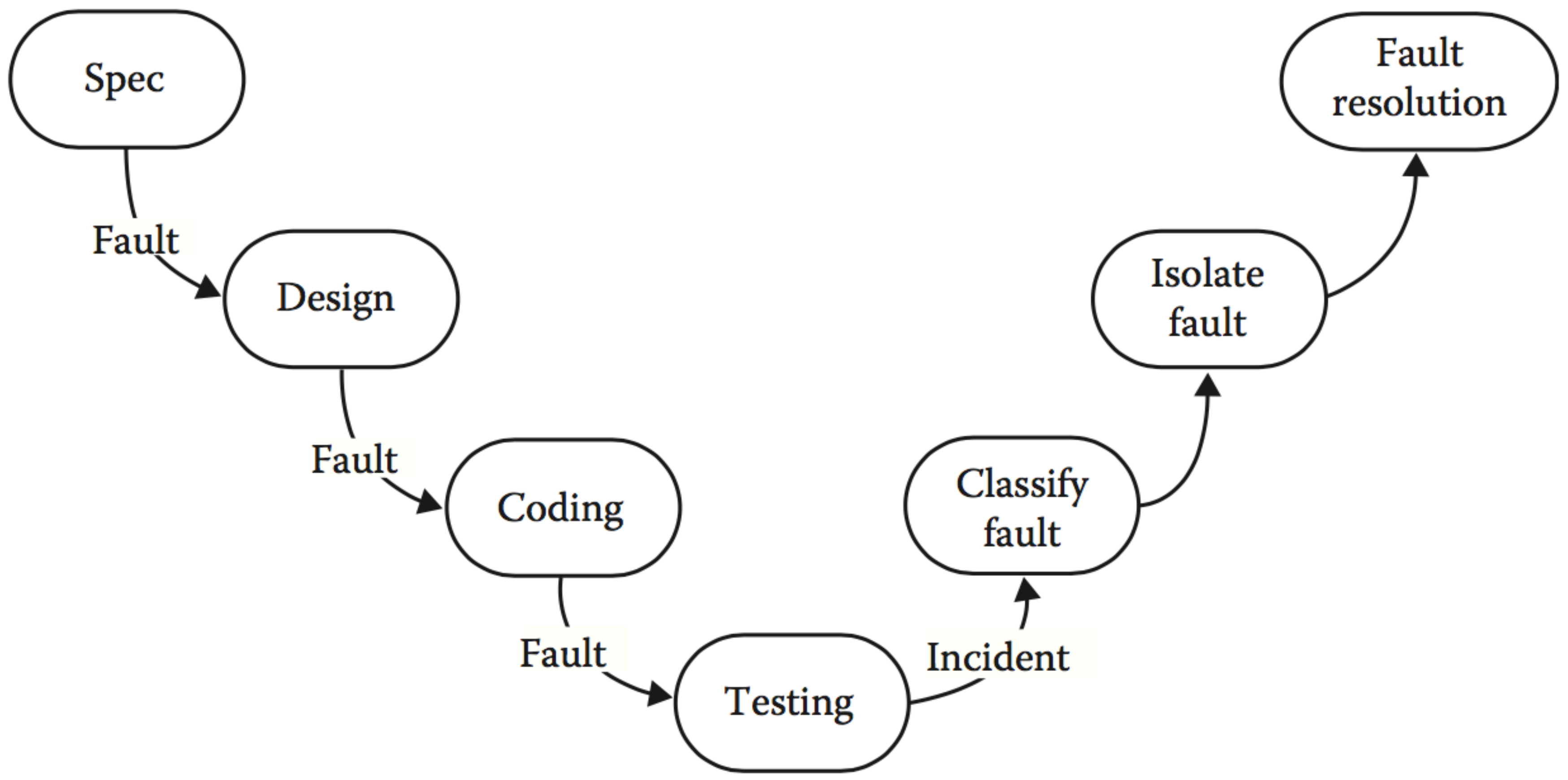


Testing Definitions & Objectives

Basic Testing Definitions

- **Errors:** People commit errors
- **Fault:** A fault is the result of an error in the software documentation, code, etc. Fault of commission versus fault of omission.
- **Failure:** A failure occurs when a fault executes
- **Incident:** Consequences of failures
- **Testing :** Exercise the software with test cases to find faults or gain confidence in the system
- **Test cases:** Set of inputs and a list of expected outputs

Testing life cycle



Fault Taxonomies

- Input/Output
- Logic
- Computation
- Interface
- Data
- … but also difficulty to resolve, priority, etc.

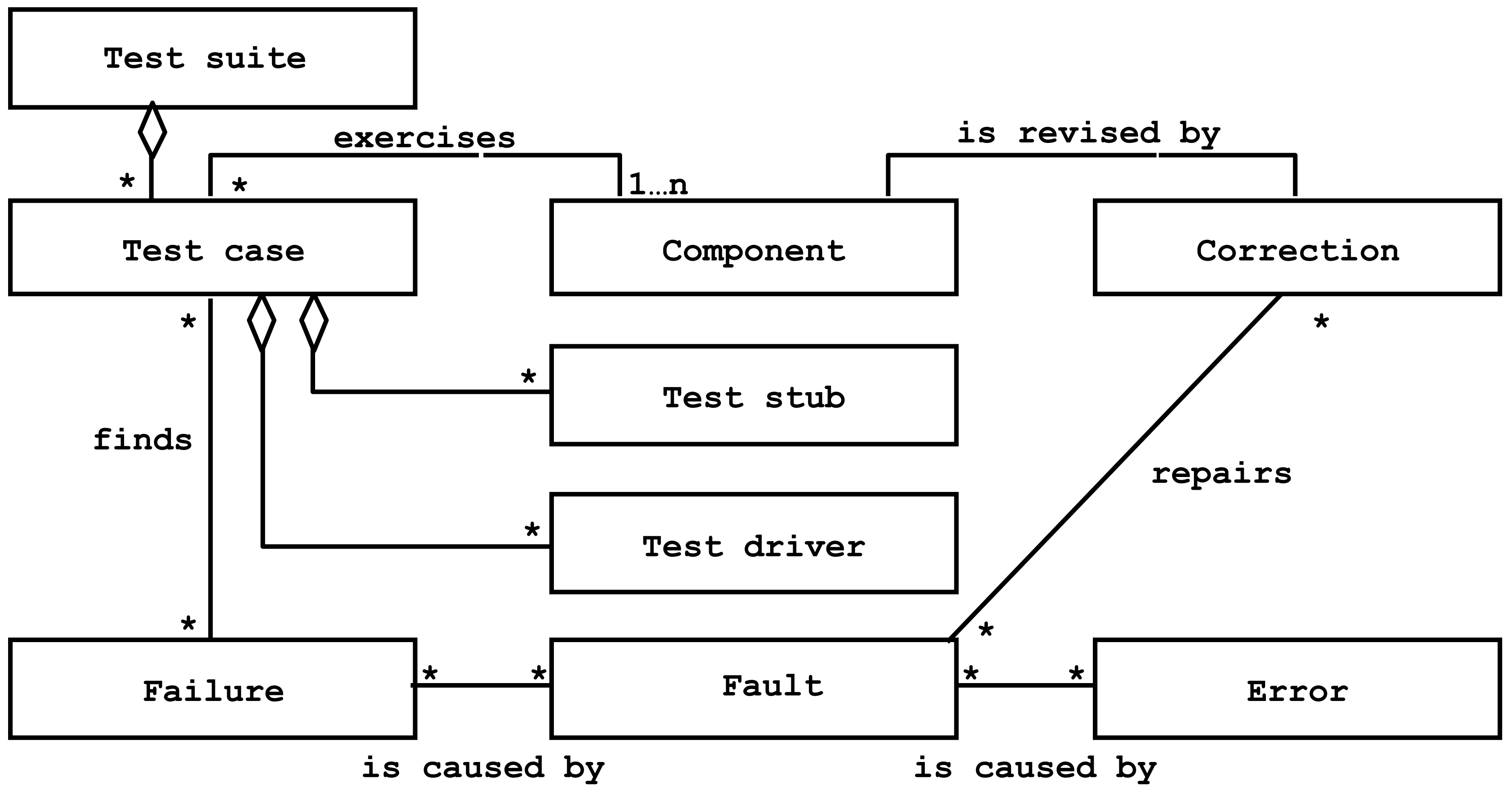
Test Stubs and Drivers

- **Test Stub**: Partial implementation of a component on which the tested component depends.

- **Test Driver**: Partial implementation of a component that depends on the tested component.

- Test stubs and drivers enable components to be isolated from the rest of the system for testing.

Summary of Definitions



Goals of Testing

- Dijkstra, 1972: "Program testing can be used to show the presence of bugs, but never to show their absence"
- No absolute certainty can be gained from testing
- Testing should be integrated with other verification activities, e.g., inspections
- Main goal: demonstrate that the software can be depended upon, I.e., *sufficient dependability*

Reality Check

- No matter how rigorous we are, software is going to be faulty
- Impossible to test under all operating conditions – based on incomplete testing, we must gain confidence that the system has the desired behavior
- Testing large systems is complex – it requires strategy and technology- and is often done inefficiently in practice

Testing vs Debugging vs Quality Assurance

- **Testing**: Finding inputs that cause the software to fail. Concentrates on the product (i.e., the software).

- **Debugging**: The process of finding a fault given a failure.

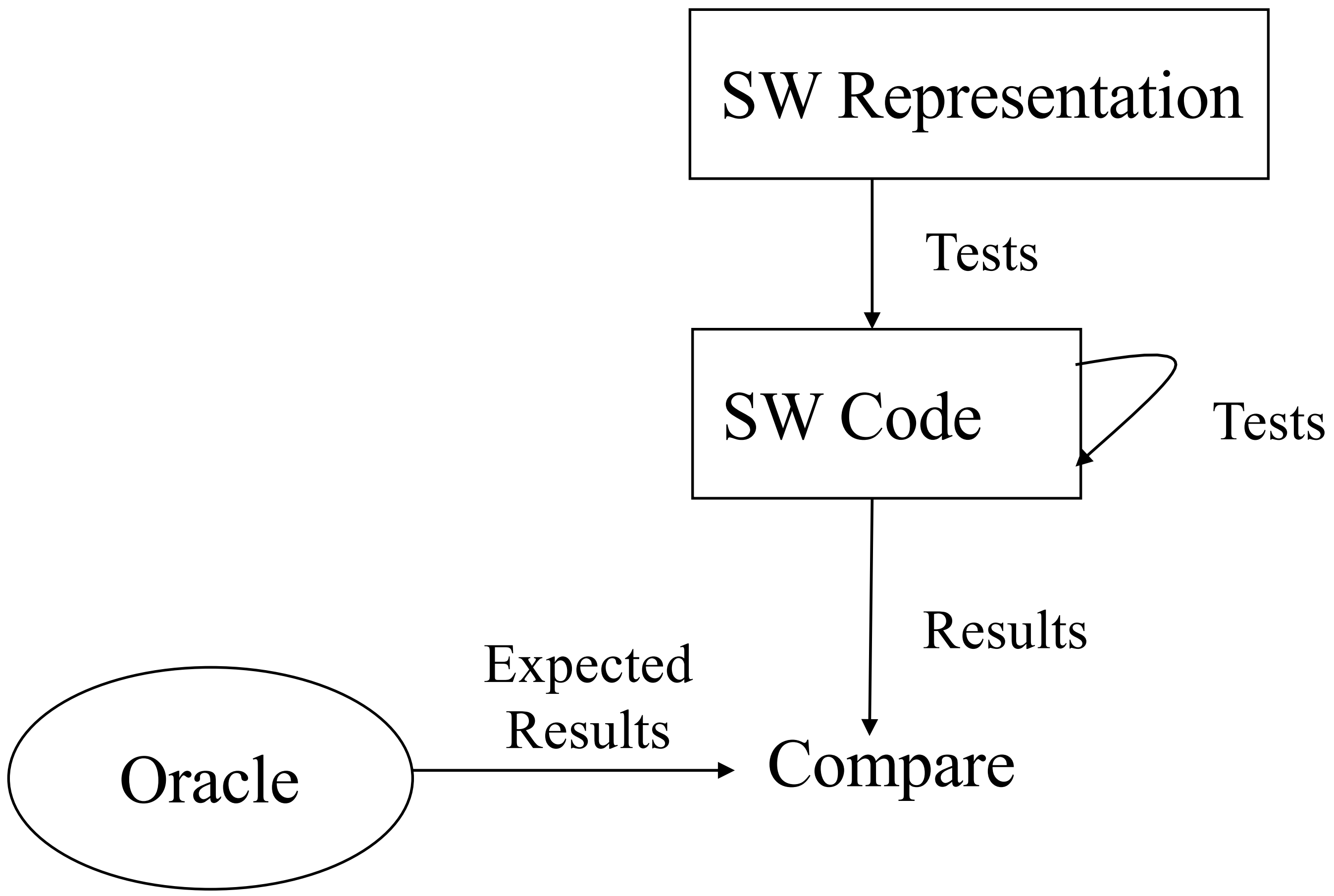
- **Quality Assurance**: Concerned with reducing the errors in the development process (i.e., improves the product by improving the process).

Validation & Verification (V&V)

- **Validation**: The process of evaluating software at the end of software development to ensure compliance with intended usage (i.e., is the software doing what the user expects?)

- **Verification**: The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase (i.e., does it do it correctly?)

Testing Process Overview



Qualities of Testing

- *Effective* at uncovering faults
- *Repeatable* so that a precise understanding of the fault can be gained
- *Automated* to lower the cost and timescale
- *Systematic* to be predictable

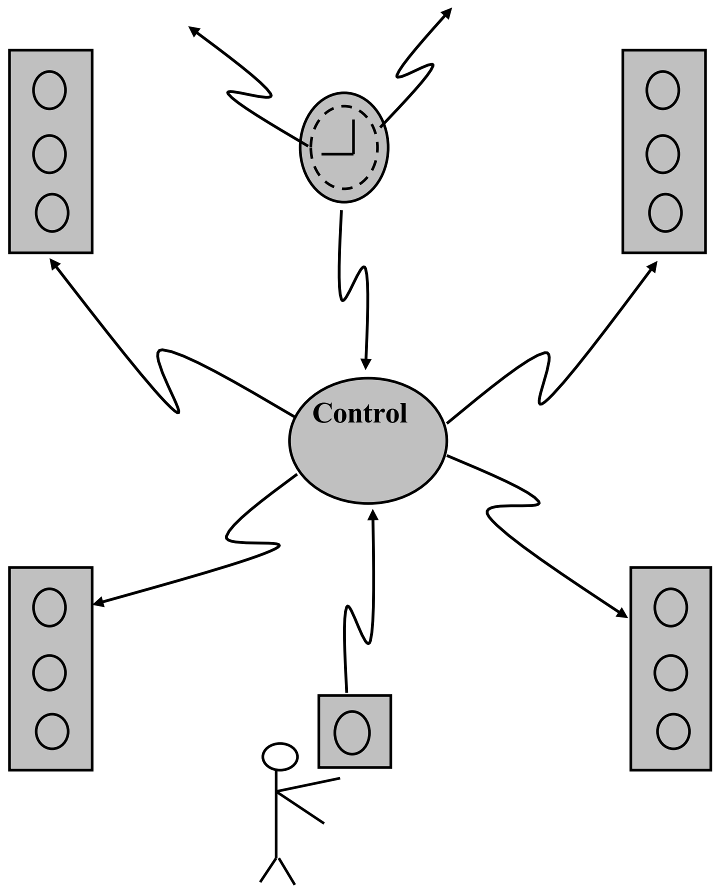
Continuity Property

- Test a bridge ability to sustain a certain weight
- If a bridge can sustain a weight equal to W1, then it will sustain any weight W2 <= W1
- The same simplifications cannot be applied to software …
- Continuity property, i.e., small differences in operating conditions will not result in dramatically different behavior -> totally lacking in software

Subtleties of Software Dependability

- Dependability: Correctness, reliability, safety, robustness
- Correct but not safe or robust: the specification is inadequate
- Reliable but not correct: failures happen rarely
- Safe but not correct: annoying failures may happen
- Robust but not safe: catastrophic failures are possible

Traffic Light Example



Correctness, Reliability:
let traffic pass according to correct pattern and central scheduling

Robustness, safety:
Provide degraded function when possible;
never signal conflicting greens

Dependability Needs Vary

- Safety-critical applications
  - flight control systems have strict safety requirements
  - telecommunication systems have strict robustness requirements
- Mass-market products
  - dependability is less important than time to market
- Can vary within the same class of products:
  - reliability and robustness are key issues for multi-user operating systems (e.g., UNIX) less important for single users operating systems (e.g., Windows)