Software quality

What is Software Quality?

- Conformance to requirements:
  - Lack of bugs
    - Low defect rate (# of defects/size unit)
  - High reliability (number of failures per n hours of operation)
    - Measured as mean time to failure (MTTF), i.e., the probability of failure-free operation in a specified time

What is Software Quality?

according to the IEEE:

software quality = (1) the degree to which a system, component, or process meets specified requirements; (2) the degree to which a system, component, or process meets customer or user needs or expectations

Software Quality Assurance

- Verification
  - Are we building the product right?
- Validation
  - Are we building the right product?

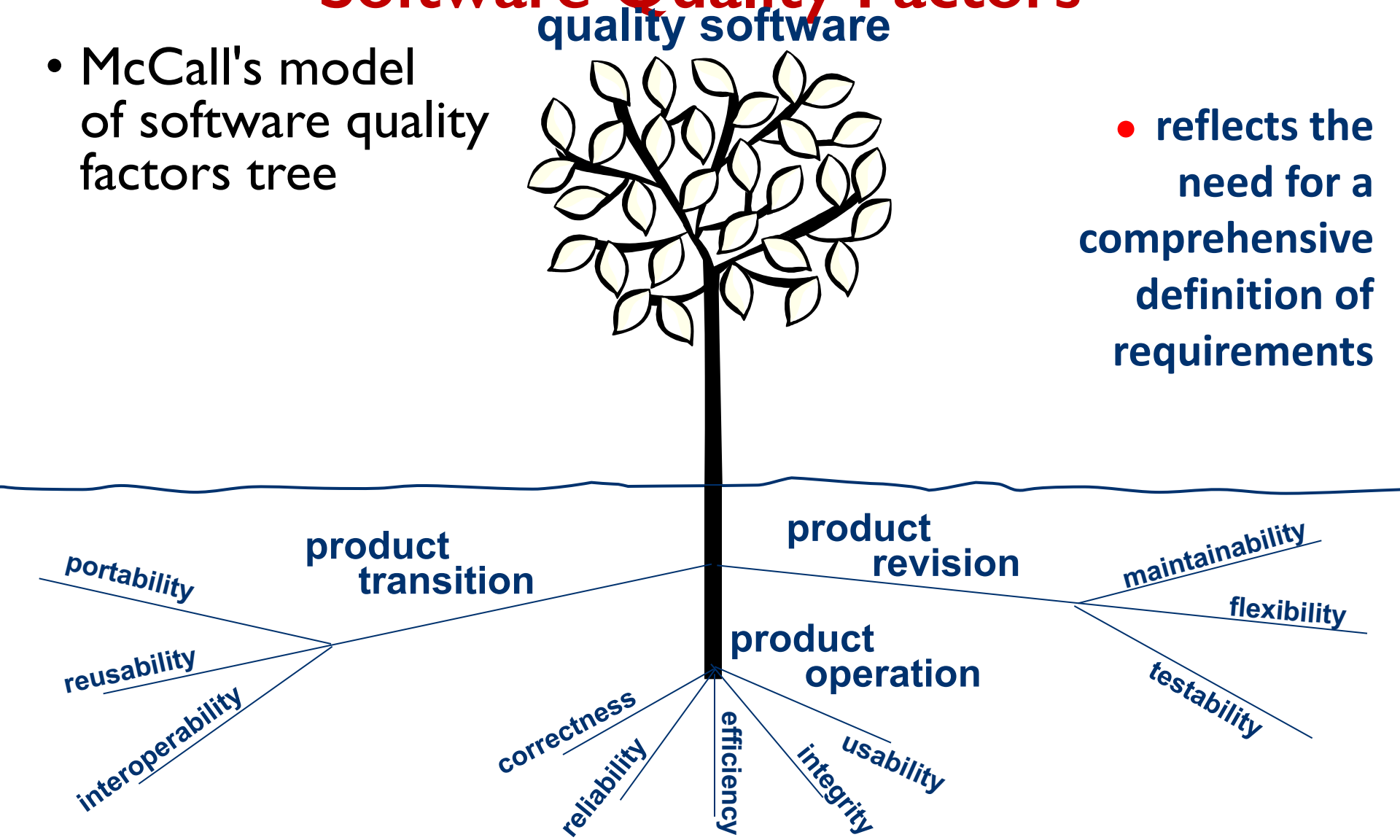
Importance of Software Quality

- Software is a major component of computer systems (about 80% of the cost) used for
  - Communication (e.g., phone system, email system)
  - Health monitoring
  - Transportation (e.g., automobile, aeronautics),
  - Economic exchanges (e.g., e-commerce),
  - Entertainment,
  - etc.
- Software defects may be extremely costly in terms of
  - Money
  - Reputation
  - Loss of life

With all the examples that we talked about before.

Software Quality Factors

- McCall's model of software quality factors tree



- **reflects the need for a comprehensive definition of requirements**

Software Quality Factors

- Correctness
  - Accuracy and completeness of required output
  - Up-to-dateness and availability of the information
- Reliability
  - Maximum failure rate
- Efficiency
  - Hardware resources needed to perform software function (processing capabilities, data storage, bandwidth, power usage)
- Integrity
  - Software system security, access rights

Software Quality Factors

- Usability
  - Training required, ability to learn and perform required task
- Maintainability
  - Effort to identify and fix software failures (modularity, documentation, etc)
- Flexibility
  - Degree of adaptability (to new customers, tasks, etc)
- Testability
  - Support for testing (e.g., log files, automatic diagnostics, etc), traceability

Software Quality Factors

- Portability
  - Adaptation to other environments (hardware, software)
- Reusability
  - Use of software components for other projects
- Interoperability
  - Ability to interface with other components/systems

- Other factors: robustness, performance, user friendliness, verifiability, repairability, evolvability, understandability, safety, manageability

What is Software Quality Assurance?

according to the IEEE:

software quality assurance = (1) a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements; (2) a set of activities designed to evaluate the process by which the products are developed or manufactured – contrast with: quality control

quality control: set of activities designed to evaluate the quality of a developed or manufactured product – after development before shipment

quality assurance aims to minimize the cost of guaranteeing quality

What is Software Quality Assurance?

according to D. Galin:

software quality assurance = a systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines

Objectives of Software Quality Assurance (SQA)

1) Assuring an acceptable level of confidence that the software will conform to functional technical requirements

2) Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements

3) Initiation and management of activities for the improvement and greater efficiency of software development, software maintenance, and software quality assurance activities

Three General Principles of Software Quality Assurance

- Know what you are doing
- Know what you should be doing
- Know how to measure the difference

3 General Principles of SQA

- Know what you are doing:
  - Understand what is being built, how it is being built and what it currently does
  - Implies a software development process with
    - Management structure (milestones, scheduling)
    - Reporting policies
    - Tracking

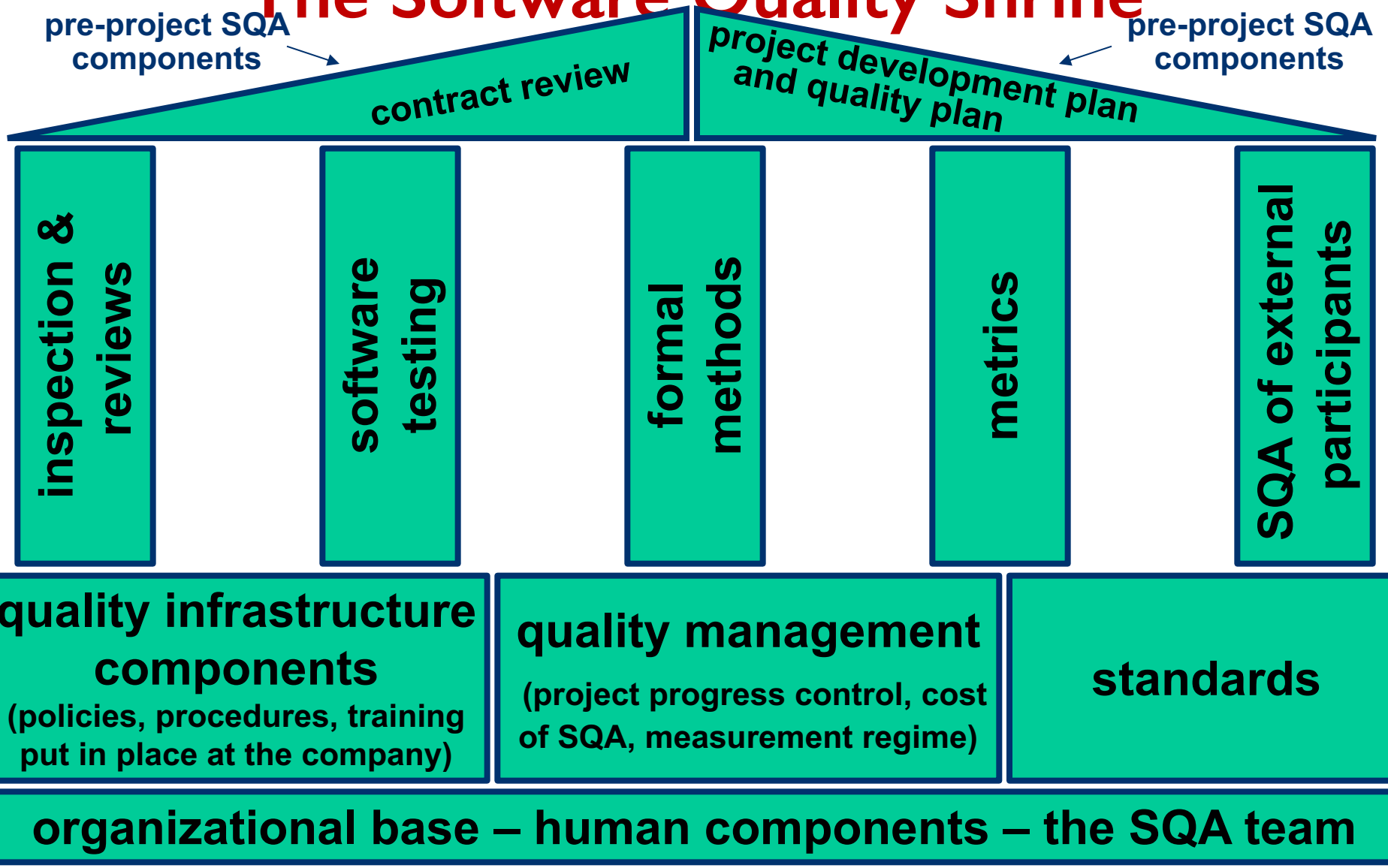
3 General Principles of SQA

- Know what you should be doing:
  - Having explicit requirements and specifications
  - Implies a software development process with
    - Requirements analysis
    - Acceptance tests
    - Frequent user feedback

3 General Principles of SQA

- Know how to measure the difference:
  - Having explicit measures comparing what is being done with what should be done
  - Four complementary methods:
    1) **Formal methods – verify mathematically specified properties**
    2) **Testing – explicit input to exercise software and check for expected output**
    3) **Inspections – human examination of requirements, design, code, … based on checklists**
    4) **Metrics – measure a known set of properties related to quality**

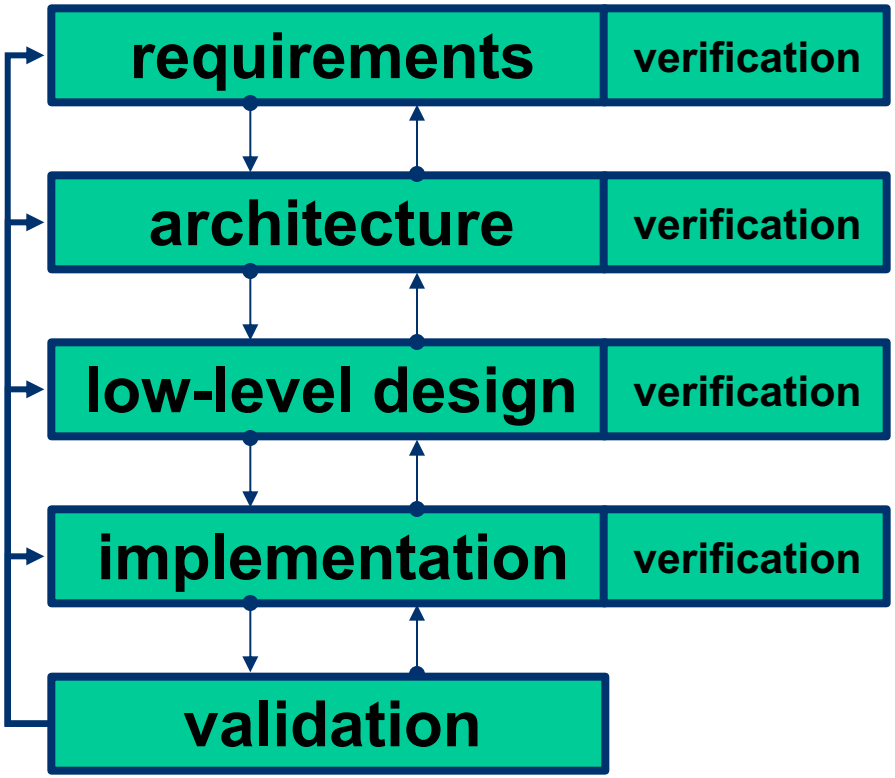
"The Software Quality Shrine"



Software Quality Assurance

- SQA includes V&V:

- Verification
  - Performed at the end of a phase to ensure that requirements established during the previous phase have been met



- Validation
  - Performed at the end of the development process to ensure compliance with user expectations

Software Quality Assurance

- SQA includes:

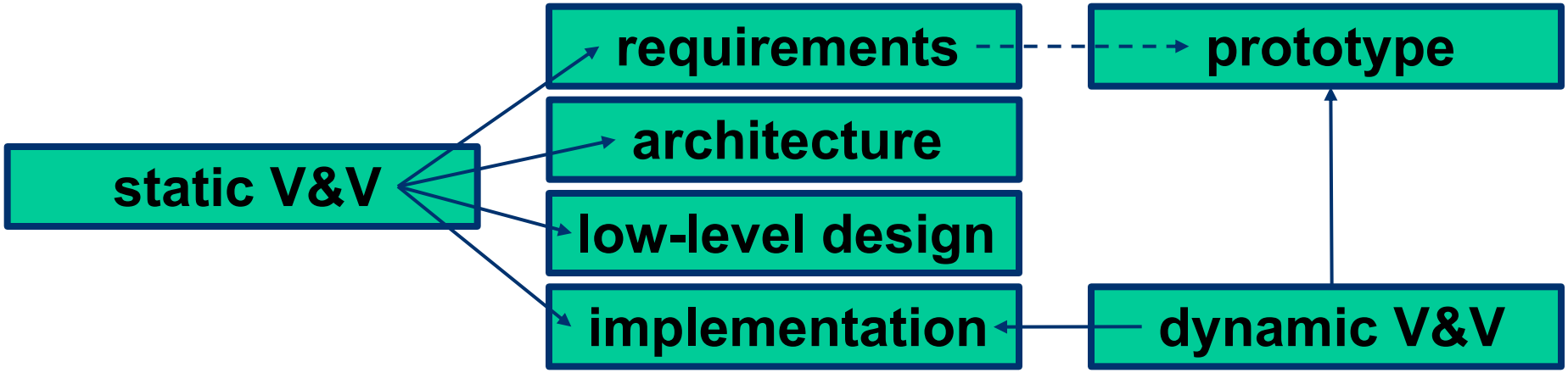
- Defect prevention
  - Prevents defects from occurring in the first place
  - Activities: training, planning, and simulation

- Defects detection
  - Finds defects in a software artifact
  - Activities: inspections, testing, or measuring

- Defects removal
  - Isolation, correction, verification of fixes
  - Activities: fault isolation, fault analysis, regression testing

Software Quality Assurance

- Typical activities of an SQA process:
  - Requirements validation
  - Design verification
  - Static code checking (inspection/reviews)
  - Dynamic testing
  - Process engineering and standards
  - Metrics and continuous improvement



Key SQA Capabilities

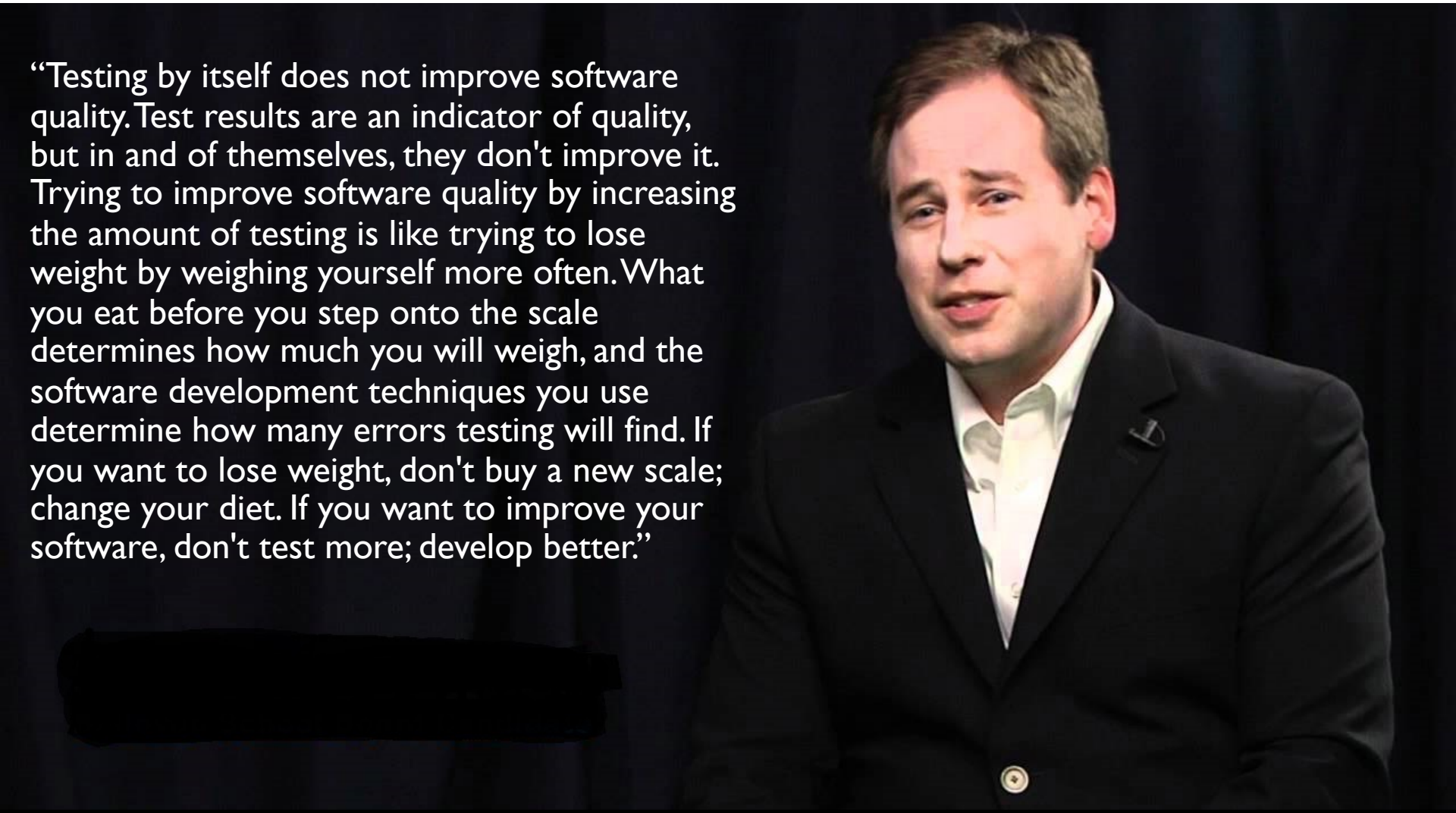
- Uncover faults in the documents where they are introduced, in a systematic way, in order to avoid ripple effects – systematic, structured reviews of software documents are referred to as inspections
- Monitor and control quality, e.g., reliability, maintainability, safety, across all project phases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

Continuous … and Testing

- Continuous Integration (CI)
  - A software development process where a continuous integration server rebuilds a branch of source code every time code is committed to the source control system
  - The process is often extended to include deployment, installation, and testing of applications in production environments
- Continuous Deployment
  - A software production process where changes are automatically deployed to production without any manual intervention
- Continuous Delivery
  - A software production process where the software can be released to production at any time with as much automation as possible for each step

Thoughts

"Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often. What you eat before you step onto the scale determines how much you will weigh, and the software development techniques you use determine how many errors testing will find. If you want to lose weight, don't buy a new scale; change your diet. If you want to improve your software, don't test more; develop better."



Steve McConnell, Code Complete

Testing basics

Today, QA is mostly testing

"50% of my company employees are testers and the rest spends 50% of their time testing"

Bill Gates, Microsoft

“legacy code is simply code without tests,” Feathers

- Code without tests is bad code.
- It doesn’t matter how well written it is; it doesn’t matter how pretty or object-oriented or well-encapsulated it is.
- With tests, we can change the behavior of our code quickly and verifiably.
- Without them [tests], we really don’t know if our code is getting better or worse.

What is Software Testing?

according to D. Galin:

software testing = formal process carried out by a specialized testing team in which a software unit, several integrated software units, or an entire software package are examined by running the programs on a computer; all the associated tests are performed according to approved test procedures on approved test cases

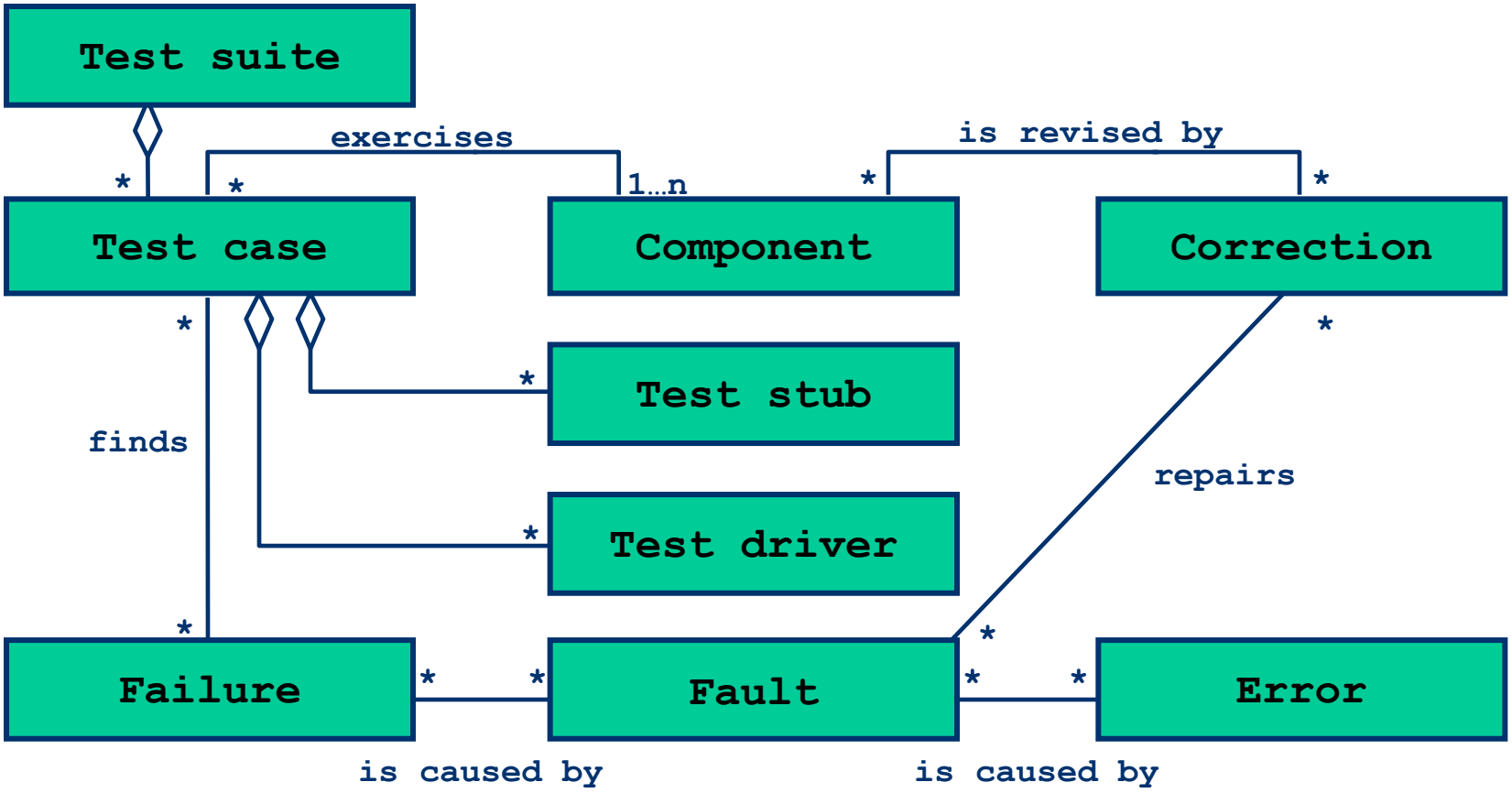
Basic Testing Definitions

- Errors: people commit errors
- Fault: a fault is the result of an error in the software documentation, code, etc.
- Failure: a failure occurs when a fault executes
- Software testing: exercise the software with test cases to gain (or reduce) confidence in the system (execution based on test cases)
  - Expectation → reveal faults with failures incidences

Basic Testing Definitions

- Test cases: set of inputs and a list of expected outputs (sometimes left out)
- Test stub: partial implementation of a component on which the tested component depends
- Test driver: partial implementation of a component that exercises and 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



Source: Bruegge and Dutoit, 2000, Object-Oriented Software Engineering, Prentice Hall

Content of a Test Case

- "Boilerplate": author, date, propose (summary), test case ID, reference to specification, version
- Pre-conditions (including environment)
- Inputs
- Expected Outputs
- Observed Outputs
- Pass/Fail (is this one always obvious?)

What do testing and debugging do?

- Testing: Evaluating software by observing its execution.

- Test Failure: Execution that results in a failure.

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

Not all tests will trigger failure. Not all failure can be associated to fault.

- Three conditions must be present for a failure to be observed.
  - The location or locations in the program that contain the fault must be reached (**R**eachability).
  - After executing the location, the state of the program must be incorrect (**I**nfection).
  - The infected state must propagate to cause some output of the program to be incorrect (**P**ropagation).

Objectives of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence"

Edsger Dijkstra, 1972

- To find as many defects as possible before they cause a production system to fail
  - Absolute certainty cannot be gained from testing → testing should be integrated with other verification activities, e.g., inspections

Objectives of Testing

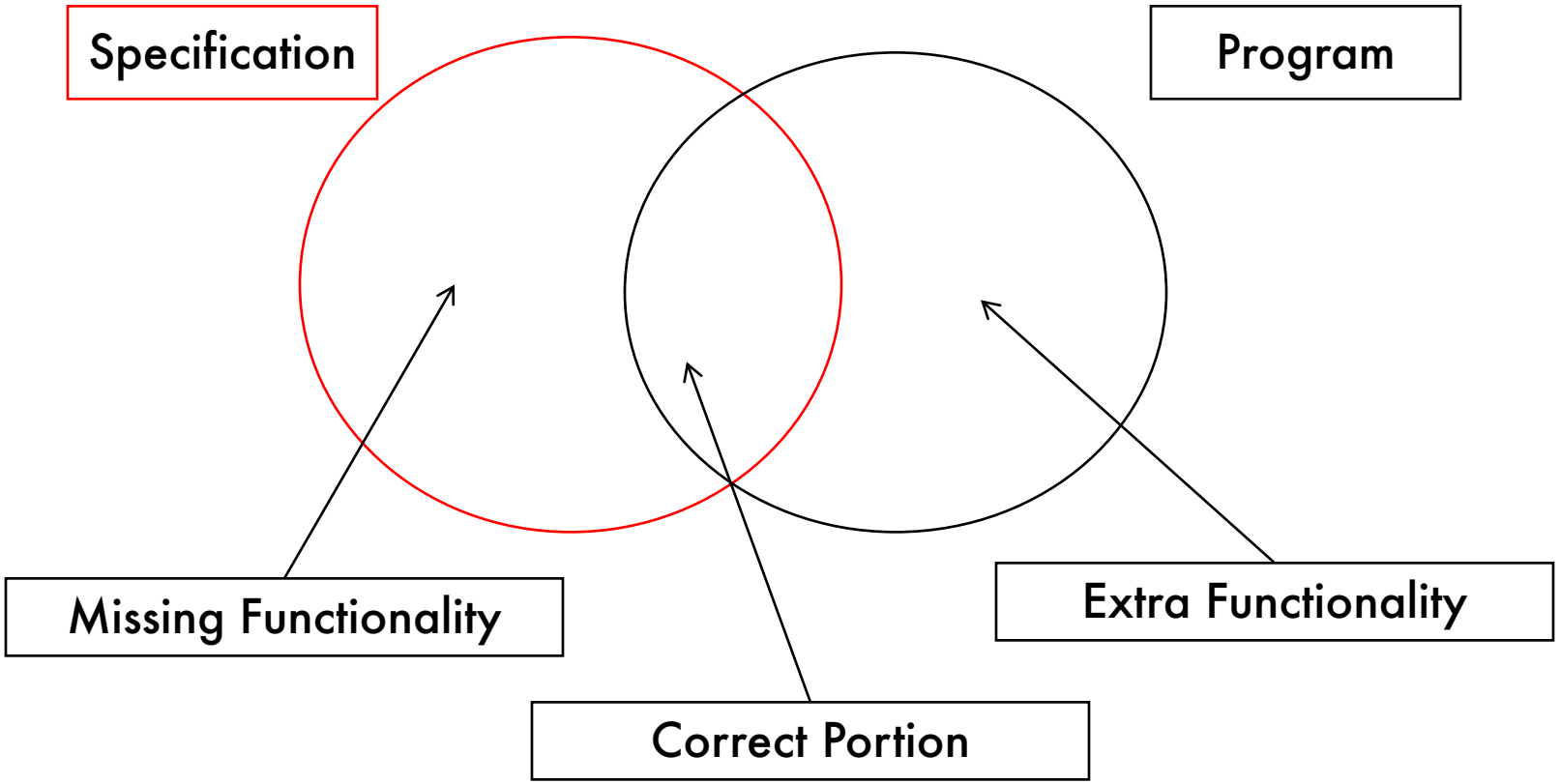
- Main goal: demonstrate that the software can be depended upon

- To bring the tested software, after correction of the identified defects and retesting, to an acceptable level of quality

- To perform the required tests efficiently and effectively, within budgetary and scheduling limitation

- To compile a record of software errors for use in error prevention (by corrective and preventive actions)

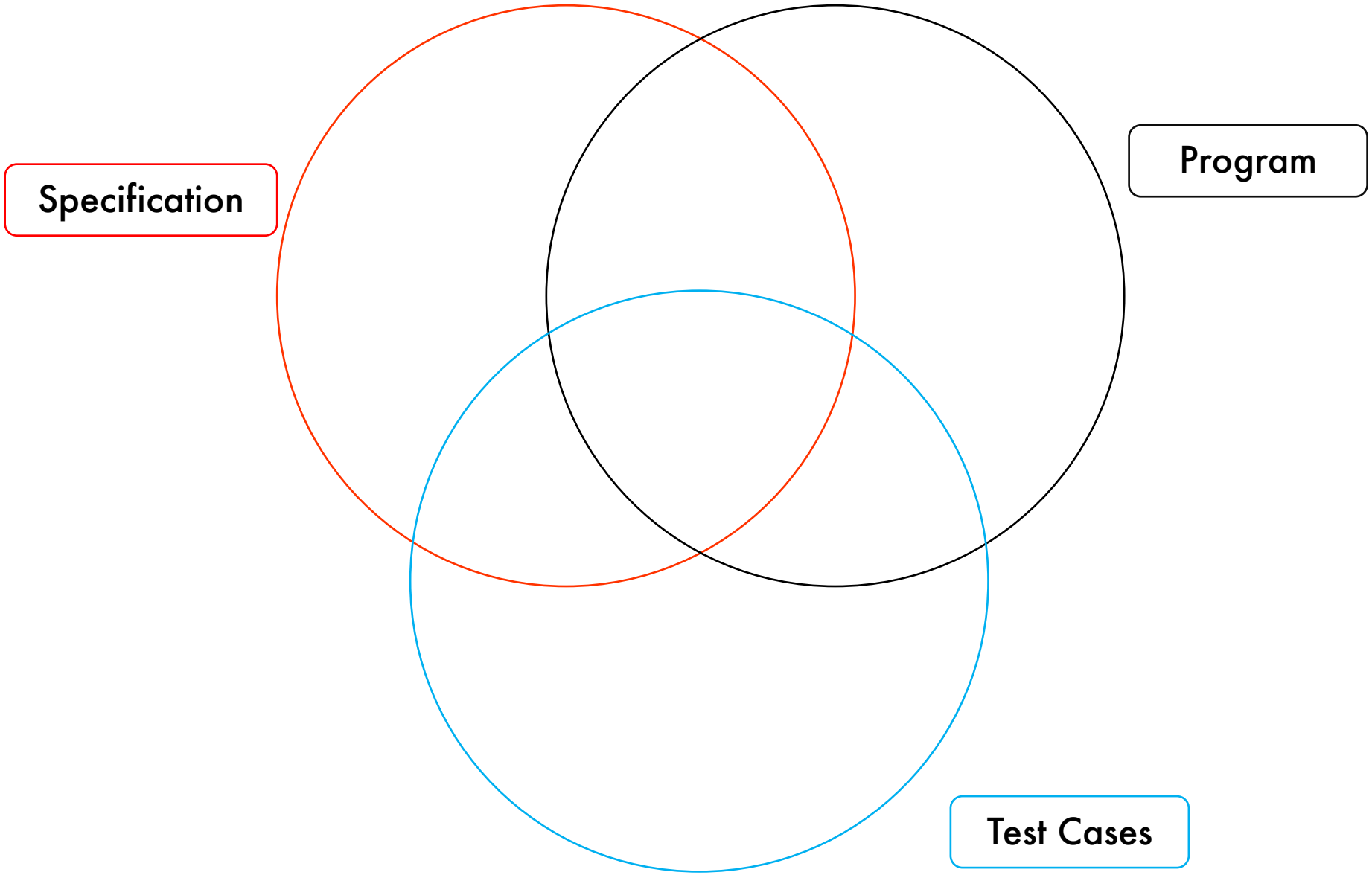
Program Behaviour



Correctness

- Impossible to demonstrate
- Better viewpoint
  - Program P is correct with respect to specification S
- Do the specification and the program meet the customer/user's expectations?
- Test can never reveal the absence of a fault

Testing Program Behaviour



Validation vs Verification

Software Quality Assurance

- Verification
  - Are we building the product right?
- Validation
  - Are we building the right product?

- Validation: The process of evaluating software at the end of software development to ensure compliance with **intended usage**. => Do the right things.

- 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. => Do the things right.

How Volkswagen Is Grappling With Its Diesel Scandal

By GUILBERT GATES, JACK EWING, KARL RUSSELL and DEREK WATKINS UPDATED Dec. 20, 2016

Volkswagen has admitted that 11 million of its vehicles were equipped with software that was used to cheat on emissions tests. The company is now contending with the fallout. RELATED ARTICLE

Writing software with the WRONG specification!

How Did the System Work?

The software sensed when the car was being tested and then activated equipment that reduced emissions, United States officials said. But the software turned the equipment down during regular driving, increasing emissions far above legal limits, most likely to save fuel or to improve the car's torque and acceleration.

The software was modified to adjust components such as catalytic converters or valves used to recycle some of the exhaust gasses. The components are meant to reduce emissions of nitrogen oxide, a pollutant that can cause emphysema, bronchitis and other respiratory diseases.

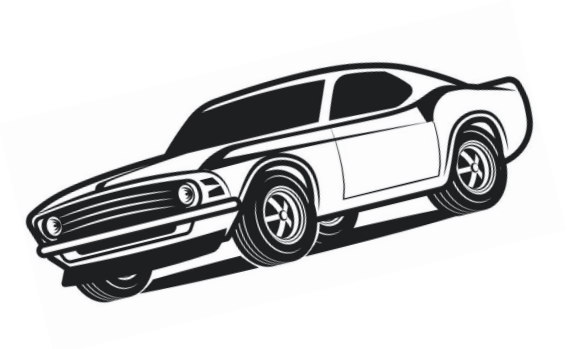
Testing techniques

Testing techniques

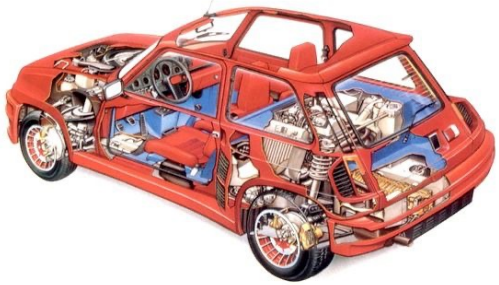
- There are a number of techniques. Each has different processes, artifacts, or approaches
- There are no perfect techniques
  - Testing is a best effort activity
- There is no best technique
  - Different contexts
  - Complementary strengths and weakness
  - Trade-offs

Basic approaches to identify test cases

- Black-Box Testing
  - based on software description (specification)
  - covers as much specified behavior as possible
  - cannot reveal errors due to implementation details



- White-Box Testing
  - based on the code
  - covers as much coded behavior as possible
  - cannot reveal errors due to missing paths



Neither approach by itself is sufficient!

Black-Box vs. White-Box Testing

Black-box

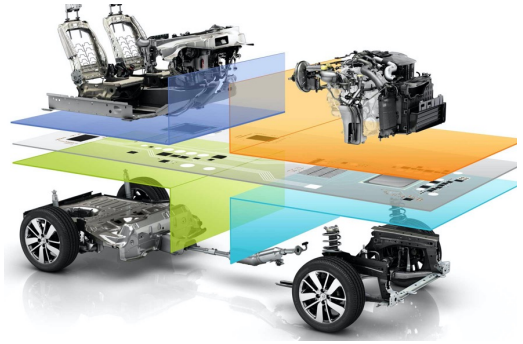
+ Checks conformance with specifications
+ Scales up (different techniques at different granularity levels)
- Depends on the specification notation and degree of detail
- Does not tell us how much of the system is being tested
- What if the software performed some unspecified, undesirable task?

White-box

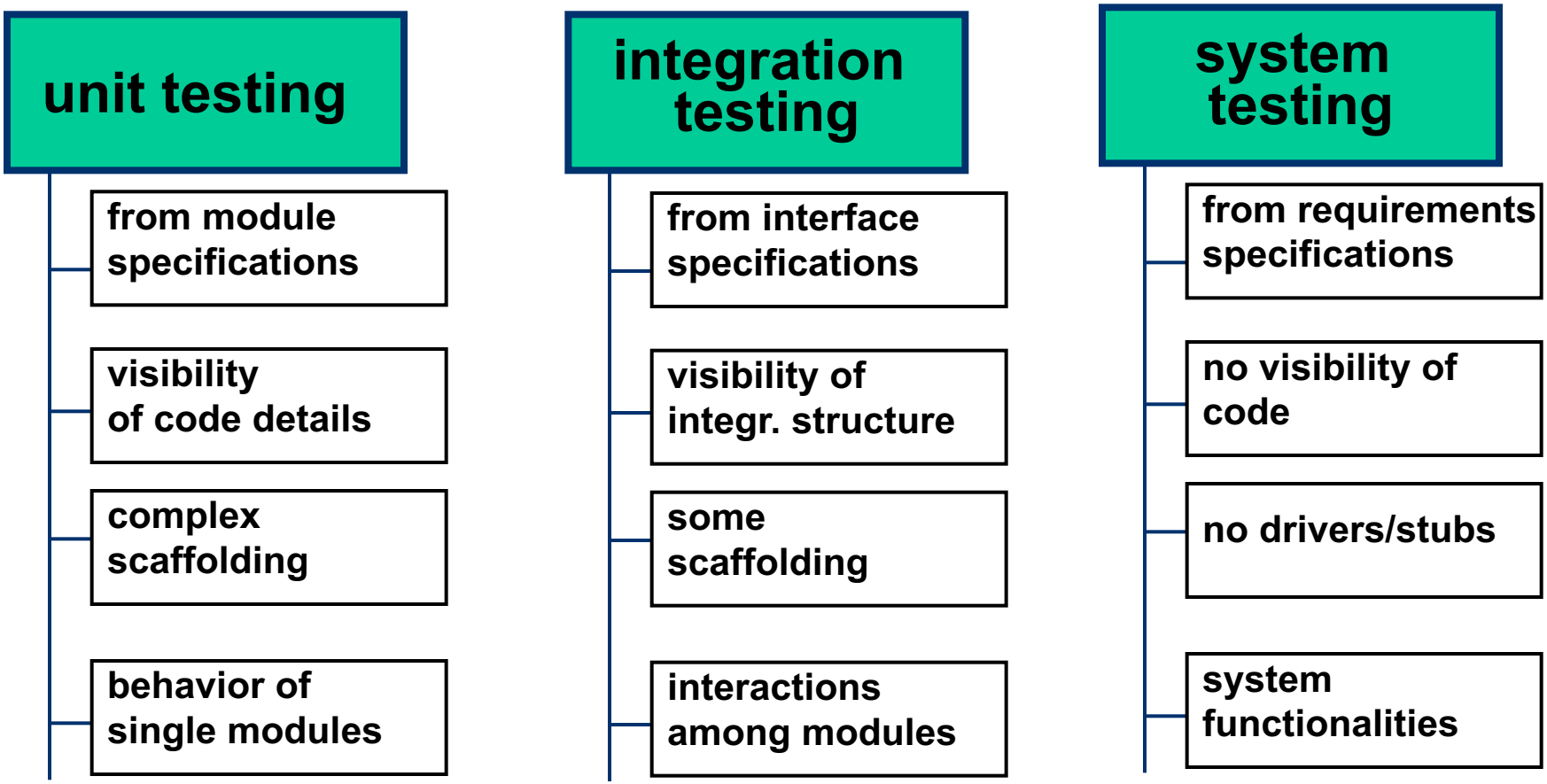
+ Allows you to be confident about test coverage
+ Is based on control or data flow coverage
- Does not scale up (mostly applicable at unit and integration testing levels)
- Unlike black-box techniques, it cannot reveal missing functionalities (part of the specification that is not implemented)

Levels of software testing

- Unit Testing
  - Individual units (e.g., a function) are tested in isolation
  - Determine whether each unit functions as designed
- Integration Testing
  - Test a group of related units together (e.g., testing database access)
  - Find interface issues between units
- System Testing
  - Test the complete software system
  - Evaluate the system's compliance with the specified requirements

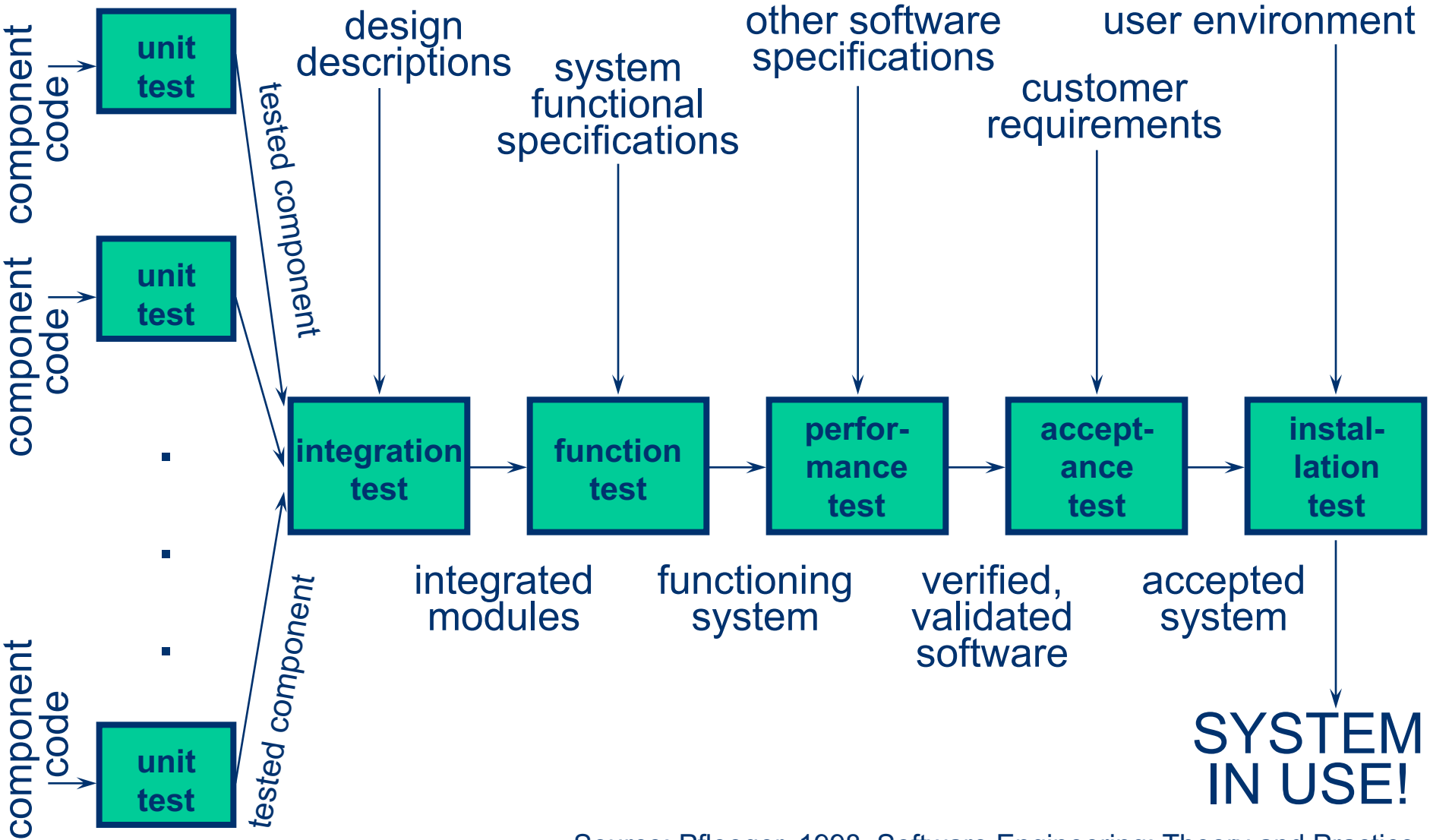


Differences among Testing Activities



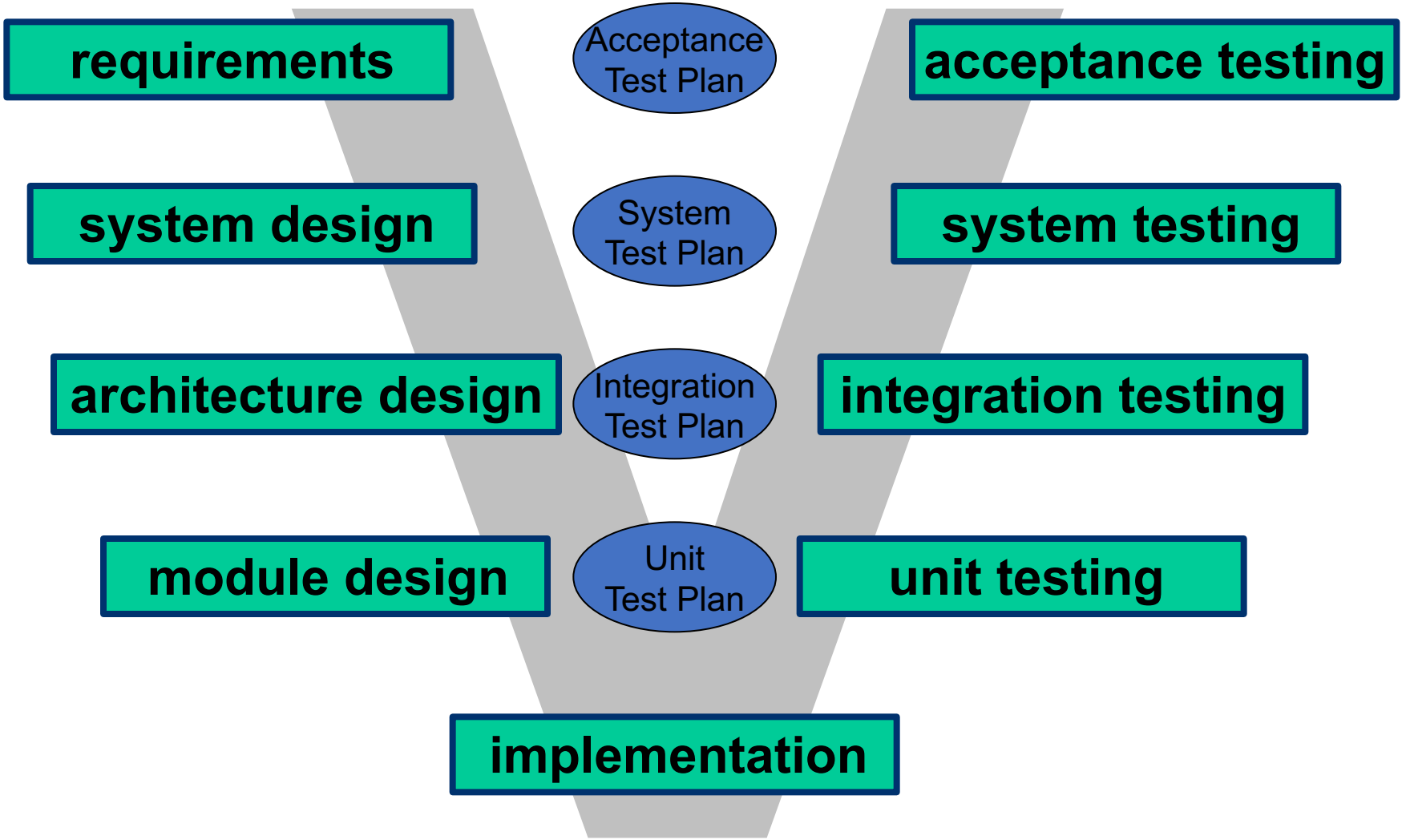
Source: Mauro Pezzè & Michal Young, 1998, Software Testing and Analysis (FSE Tutorial)

More on Types of Testing



Source: Pfleeger, 1998, Software Engineering: Theory and Practice

The V-Model of Development



Integration Testing

- Integration of well-tested components may lead to failure due to:

- Bad use of the interfaces (bad interface specifications / implementation)

- Wrong hypothesis on the behavior/state of related modules (bad functional specification / implementation), e.g., wrong assumption about return value

- Use of poor drivers/stubs: a module may behave correctly with (simple) drivers/stubs, but result in failures when integrated with actual (complex) modules

Two unit tests, zero integration tests

- Well-tested modules may still fail integration tests…

Thought and discussion

- Unit Testing vs Integration Testing
  - Are they really different?
- What about Unit testing vs System testing ?

Feathers's thought on Unit test

1. Test runs fast.
2. Test helps us localize problems.

Is it really a unit test?

It doesn't matter, so long as it is fast and it helps find problems.