Software Testing

Jesper Andersson

Linnæus University

Cost of Defects

- ✓ US National Institute of Standards and Technology
  - 2002 estimae costs of software failure in US alone at $60 billion per year
  - 80% of software development costs are finding and fixing defects
- ✓ Royal Academy of Engineering (UK) reported
  - Only 16% of projects in the UK were considered successful
  - This suggests that around GBP 17 billion will be wasted in 2003/2004 alone.
- ✓ Swedish numbers something li 80-100 Billion SEK per year – 10 Billions?

“Smelly” Projects

- ✓ Air-traffic control problems
  - The new UK National Air Traffic System (NATS) stopped three times in 5 weeks in April-May 2002 causing huge delays. Each time was due to a different kind of software failure.
  - This system cost around $600 million and was 6 years late.
- ✓ Trains …
  - 23/Jan/2004. The new 1.2 billion pound London Underground Tube smartcard was reporting a ‘code 24’, (negative pre-paid balance !), condition stopping many travelers from getting on their trains.

About the security content of iOS 7.0.6

Languages Englis

This document describes the security content of iOS 7.0.6.

For the protection of our customers, Apple does not disclose, discuss, or confirm security issues until a full investigation has occurred and any necessary patches or releases are available. To learn more about Apple Product Security, see the Apple Product Security website.

For information about the Apple Product Security PGP Key, see "How to use the Apple Product Security PGP Key."

Where possible, CVE IDs are used to reference the vulnerabilities for further information.

To learn about other Security Updates, see "Apple Security Updates".

iOS 7.0.6

- **Data Security**

  Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

  Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

  Description: Secure Transport failed to validate the authenticity of the connection. This issue was addressed by restoring missing validation steps.

The Code

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedP
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.upd
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.fin
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

The code will always jump to the end from that second goto, err will contain a successful value because the SHA1 update operation was successful and so the signature verification will never fail.

Basic Definitions: Testing

- ✓ What is software testing?
  - – Running a program
  - – Finding faults
    - Defects
    - Errors
    - Flaws
    - Faults
    - BUGS

Faults, Errors, and Failures

- ✓ Fault: a static flaw in a program
  - What we usually think of as "a bug"
- ✓ Error: a bad program state that results from a fault
  - Not every fault always produces an error
- ✓ Failure: an observable incorrect behavior of a program as a result of an error
  - Not every error ever surfaces

“Must haves” if you should find faults with tests

- ✓ Reachability: A test much execute (reach) the location of the fault in the source code
- ✓ Infection: The fault must alter the program state into an illegal one, that is result in an error
- ✓ Propagation: The error a some point produce an incorrect output, that is an observable failure

- ✓ Not all faults are reachable.
- ✓ Not all faults alter the state into an illegal state
- ✓ Not all faults surface

More “Must haves”: Controllability and Observability

- ✓ Goals for a test case:
  - Reach a fault
  - Produce an error
  - Make the error visible as a failure
- ✓ **Controllability**
  - Steer the program to a point that we would like to test
- ✓ **Observability**
  - Ability to see what the application is actually doing.

Contributors to Testability from: Robert V Binder

- ✓ **System perspective**
  - **Representation**: How do we know what to test? Are requirements and specifications available?
  - **Implementation**: What is the structure of the as-built system
  - **Built-in Test**. Does the system under test have any built-in features that help us set up and evaluate tests?

Contributors to Testability cont.

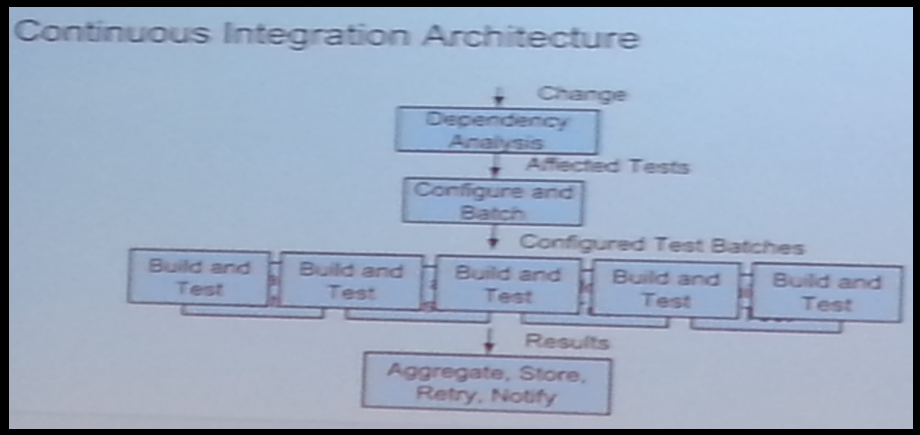
- ✓ **Testing perspective**
  - **Test Suite**. Is the test suite organized to make it extensible or is an all-or-nothing monolith?
  - Test Tools. Are tools available that can automate regression testing? To what extent can we easily change test suites?
  - Test Process. Does the testing work flow complement the development work flow? Is testing an equal partner in development of requirements and features or an afterthought?

Testing requires Development Resources!

- ✓ Include this in your planning!
- ✓ Setup test environment
  - – Test management tools
    - • Manage test cases, specification and documentation
    - • Manage test reports
  - – Drivers - Stubs
  - – Emulators/simulators/hardware in the loop
- ✓ Develop test-scripts
  - – Test-data and expected output

Example. “the big search engine company”

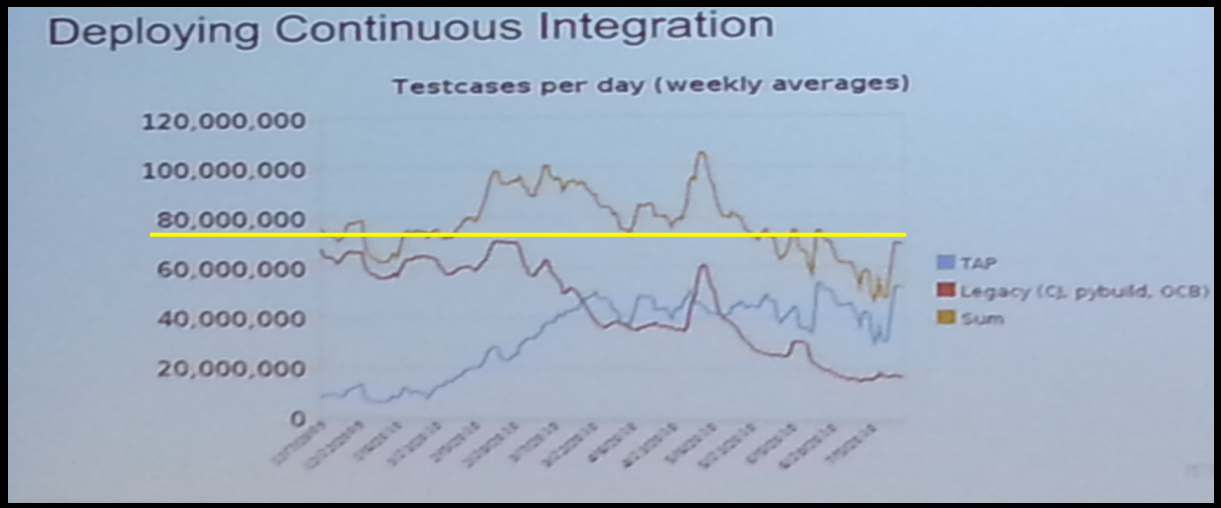
Continuous integration (CI), a strategy in software technology where working copies from several developers are integrated in a shared “mainline” several times everyday.



Map-Reduce Architecture

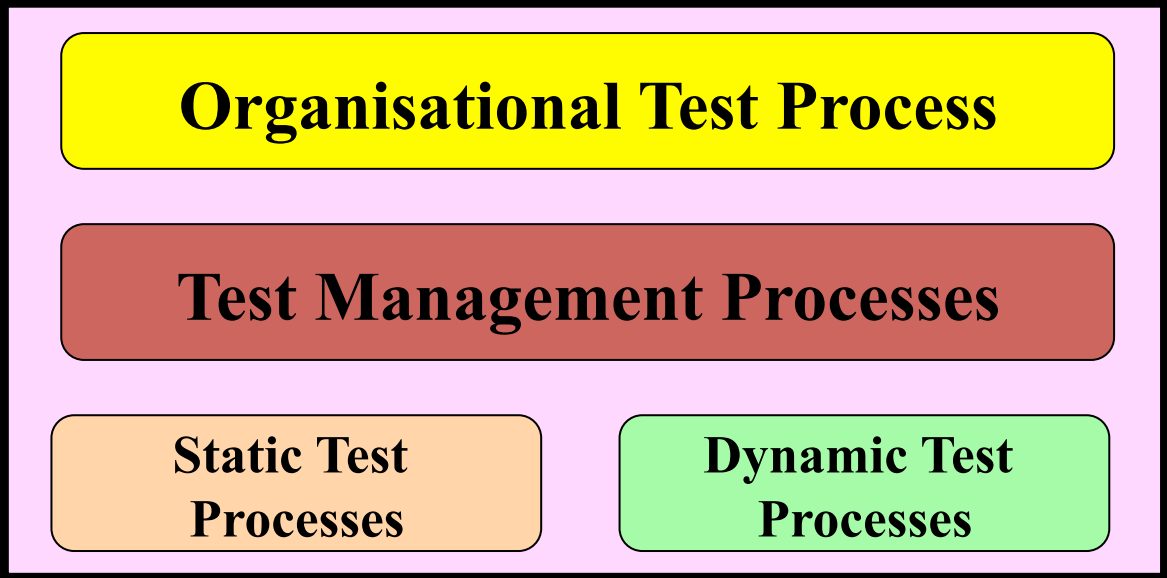
Continued.

Sum of executed test cases per day (average) yellowish graph.



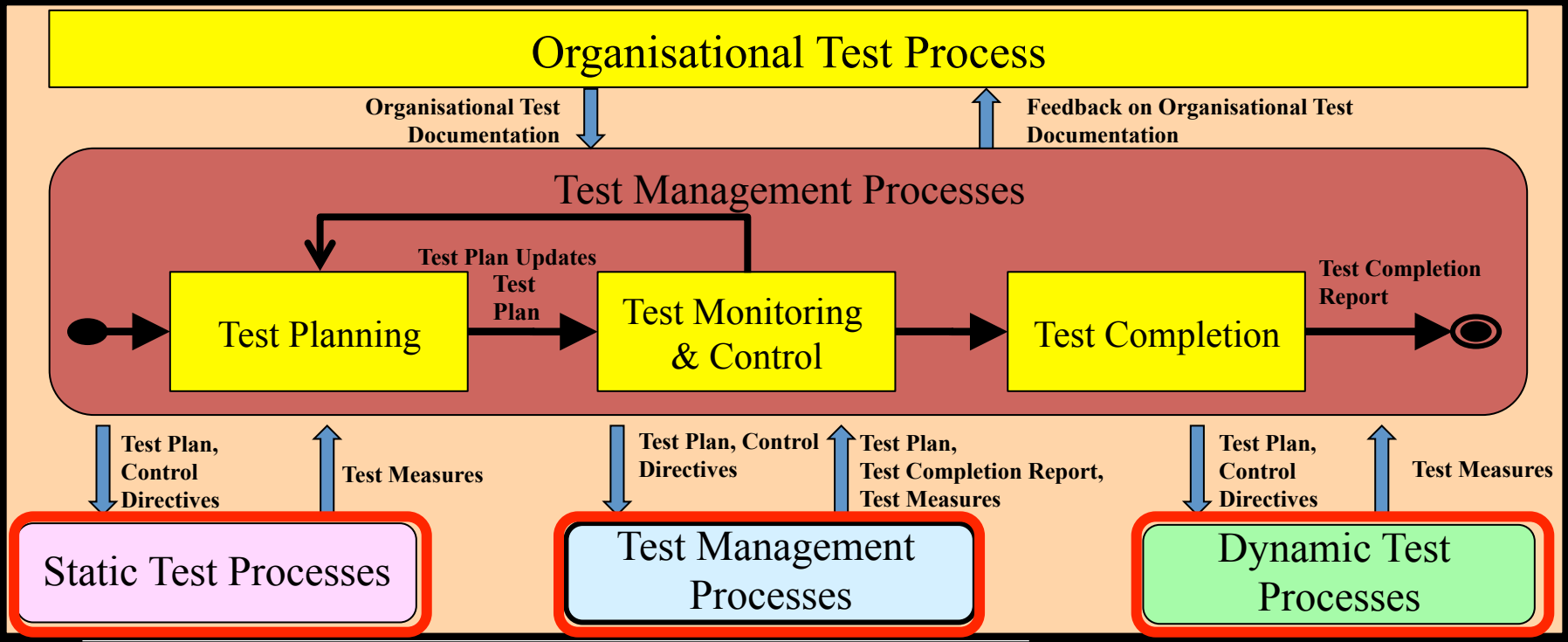
Testing Integrity???

Testing processes – ISO29119

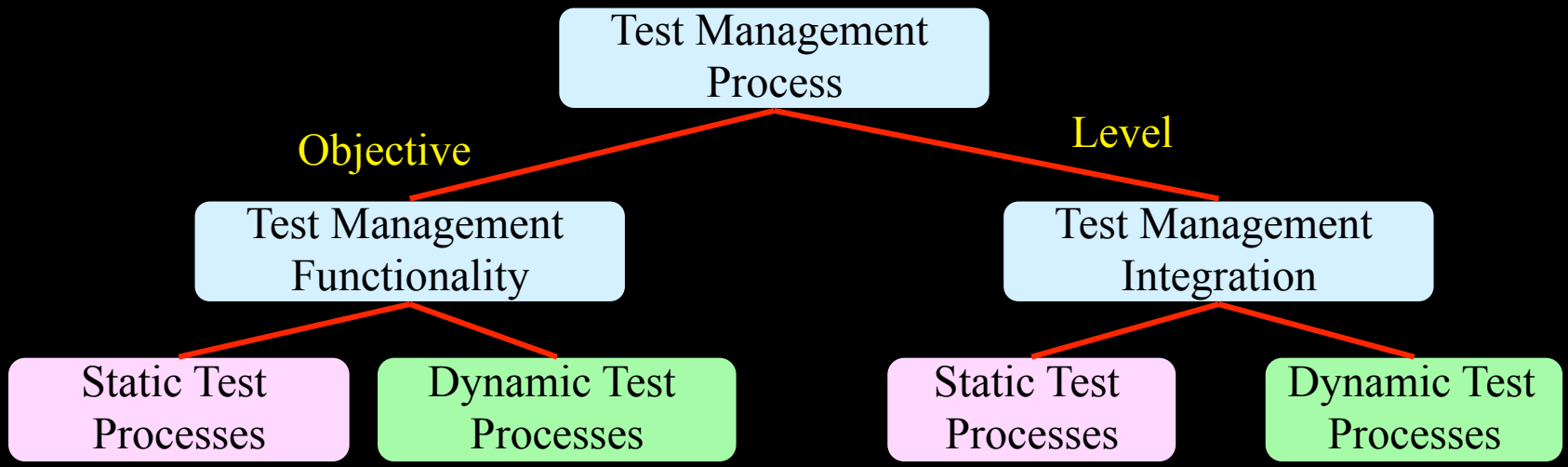


Testing Integrity!!!

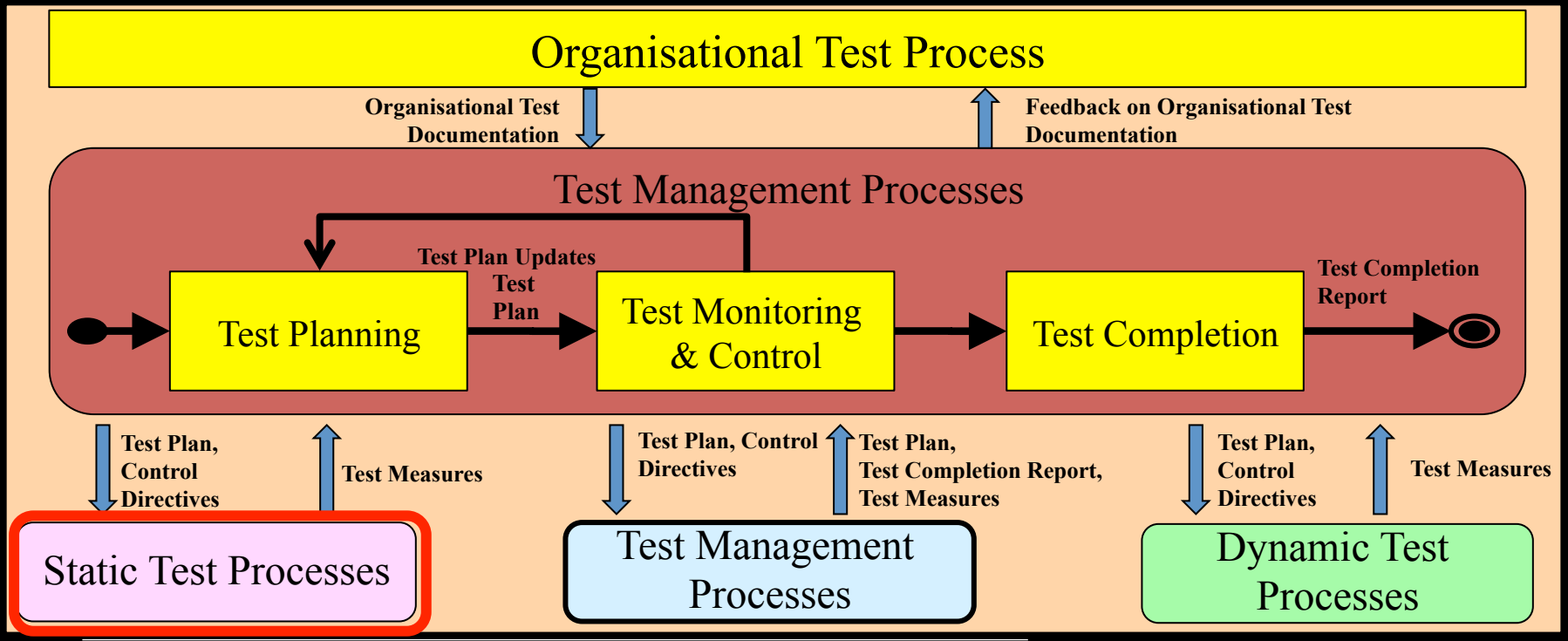
Test Management Processes – Objective and Type



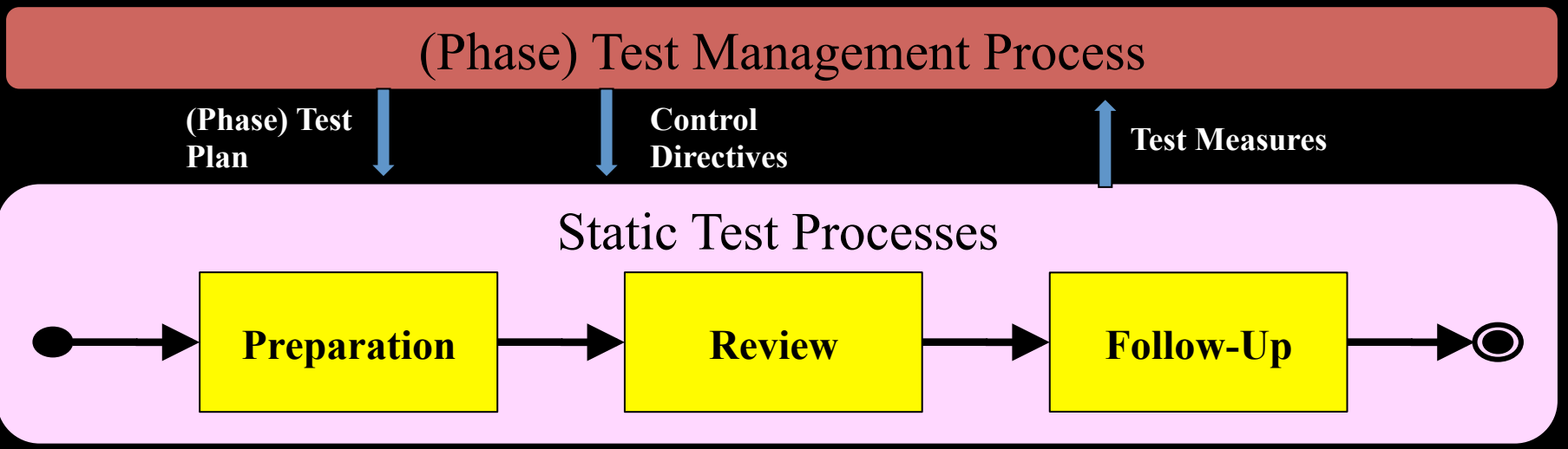
Processes for Test Objectives or Test Level



Static Test Processes



Static Test Processes



Static Testing can Remove errors in early phases

- ✓ Common misconceptions
  - Quality can only be tested when the code is available
  - Quality can only be improved by removing errors from the code
- ✓ Non-execution-based testing, Reviews: Inspections & walkthroughs
- ✓ Benefits
  - Not all errors are found in testing
  - Early detection of faults will reduce the numbers left when testing

Design Reviews

- ✓ A review is when you or someone else examine your own work.
- ✓ Objective - find product defects early in the development process.

Design Inspections (Reviews)

- ✓ Inspections were introduced by Fagan at IBM in 1976.
- ✓ Inspections follow a structured process
  - Done by peers
  - No managers attend
  - Each participant has a defined role
  - Preparation is required
  - Objective – find problems!

Why Do Reviews?

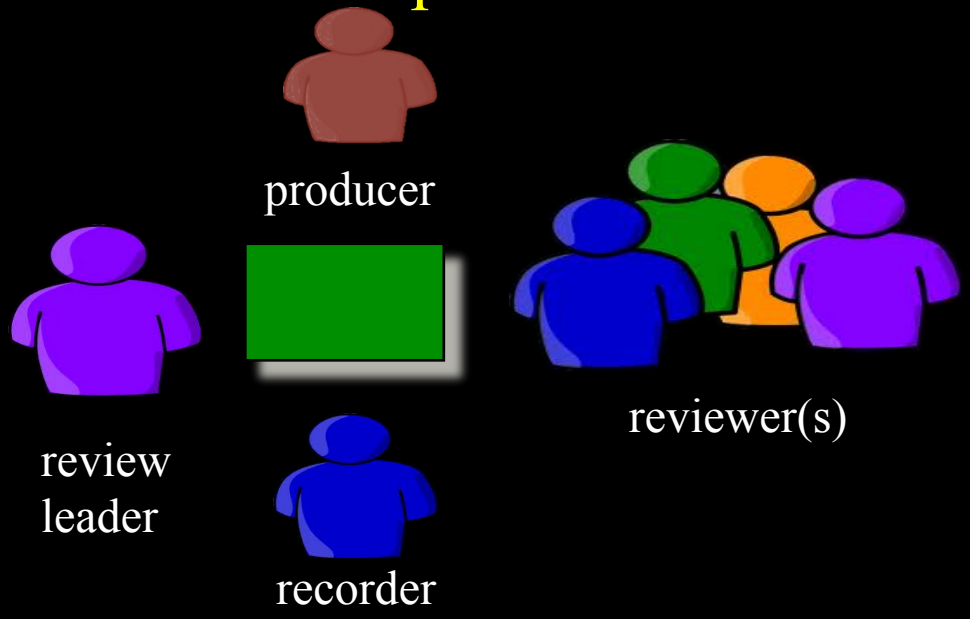
- ✓ Data show that it is much more efficient to find defects in reviews than in testing
  - in unit test, typically only about 2 to 4 defects are found per hour
  - code reviews typically find about 6 to 10 defects per hour
  - experienced reviewers can find 70% or more of the defects in a product
- ✓ Defect removal after testing is much more expensive

Why Reviews are Efficient

- ✓ Tests generates failures
  - detect that it was unusual
  - figure out what the test program was doing
  - find where it was in your program
  - figure out what defect could cause such behavior
- ✓ You are searching for the unplanned and unexpected.

- ✓ With reviews and inspections
  - you follow your own logic
  - when you find a defect, you know exactly where you are
  - you know what the program should do and did not do
  - you thus know why this is a defect
  - you are therefore in a better position to devise a complete and effective fix

The review/inspection team



Result of a review

- ✓ Decision about the product
  - accept without further modification
  - reject the work due to severe errors (review must be repeated)
  - accept with minor modifications (incorporated into the document by the producer)
- ✓ All participants have to sign-off
  - shows participation and responsibility
  - show that they comply with findings

Conducting the Review

- ✓ Be prepared - evaluate product before review
- ✓ Use product specific **check lists**
- ✓ Review the product, not the people
- ✓ Keep your tone, ask questions do not accuse
- ✓ Stick to the agenda
- ✓ Raise issues, don't resolve them!
- ✓ Discussions rules, time, its technical correctness not style
- ✓ Schedule reviews in project plan
- ✓ Record and report all results

Static Testing of Designs – Examples from checklists

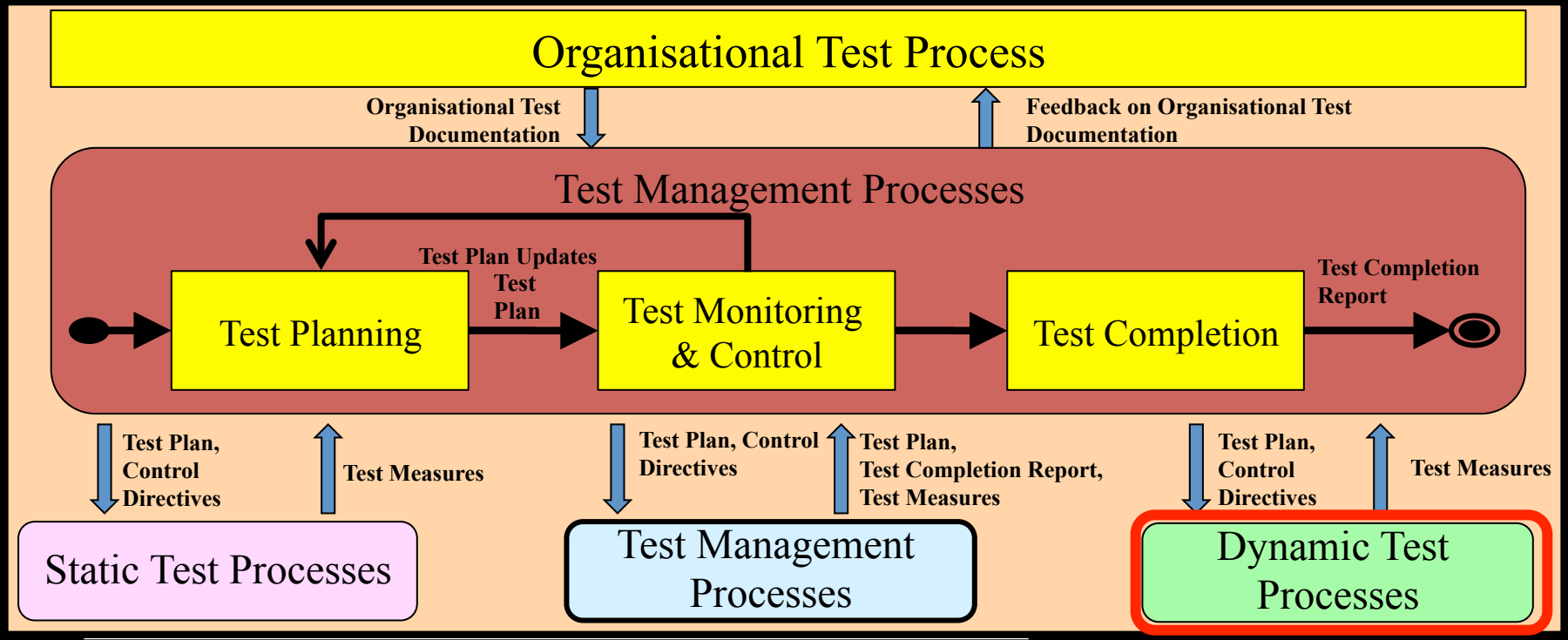
Is it clear where each requirement is going to be implemented using this architecture?	For each subsystem/component consider: • Do the components in the subsystem make sense in relation to the functional requirements for this subsystem? • Do the classes in the component make sense in relation to the functional requirements for this component? For each requirement, is it clear which subsystems and classes can implement that requirement? ***If usage scenarios are available:*** For each usage scenario, is it clear which subsystems and classes will participate in implementing the relevant functionality?
Are all specified system outputs and inputs covered with this architecture?	Is it clear which component(s) is(are) responsible for input and/or output?

Static Testing of Designs – Examples from checklists

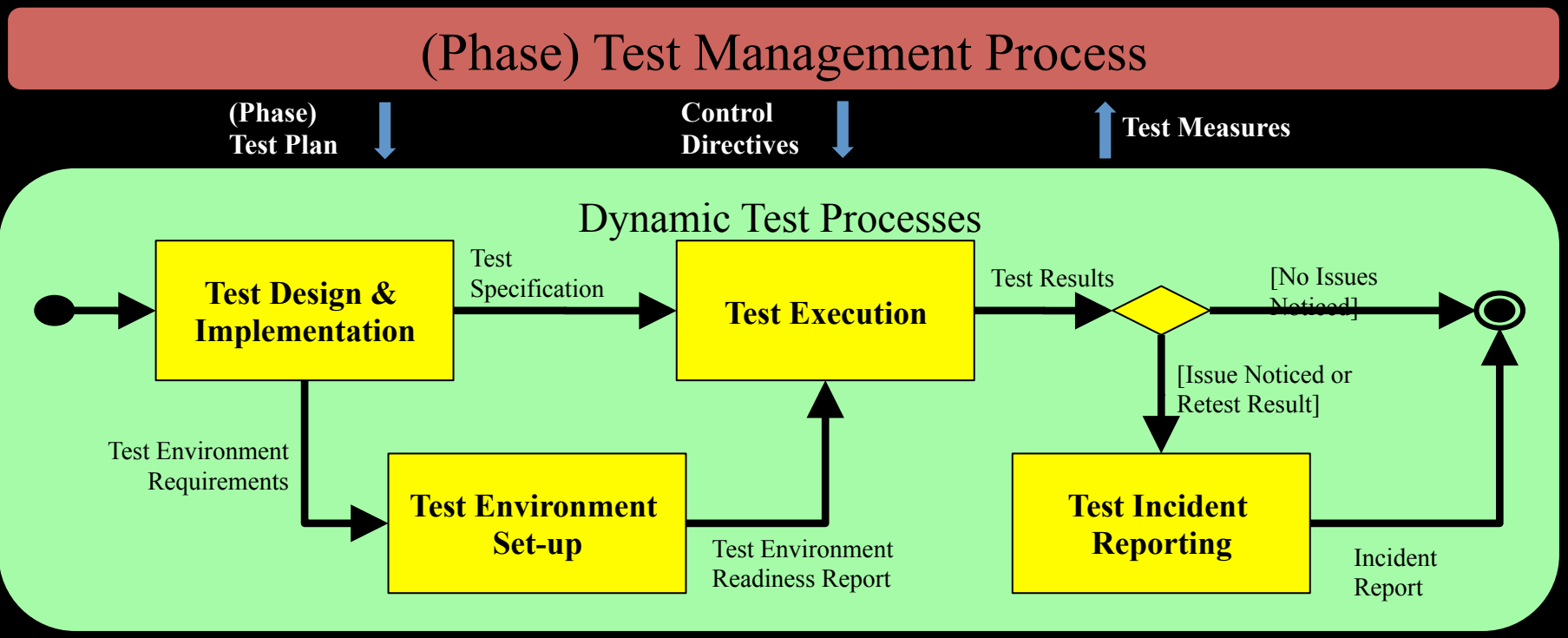
Does the structure of the system represent more functionality than specified by requirements?	For each subsystem/component consider: • Is the functionality the component provides necessary in order to implement the specified requirements?

Interfaces:	
Does any unnecessary coupling exist?	Examples: • Extra coupling between components that should not be communicating with each other o components that directly access each other rather than going through an intermediate component o components not capable of communicating with each other with direct interfaces described • Components accessing more than the number of required classes in a component

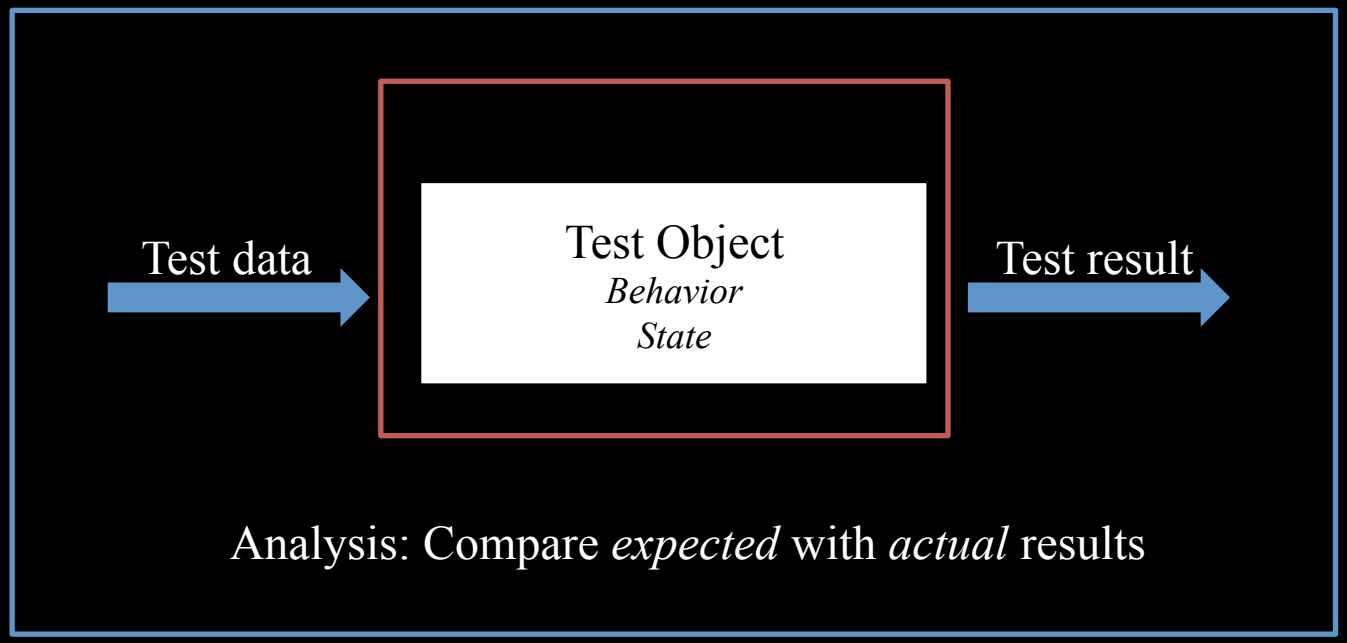
Dynamic Test Processes



Dynamic Test Processes



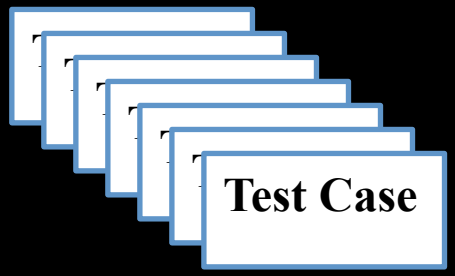
What is a test?

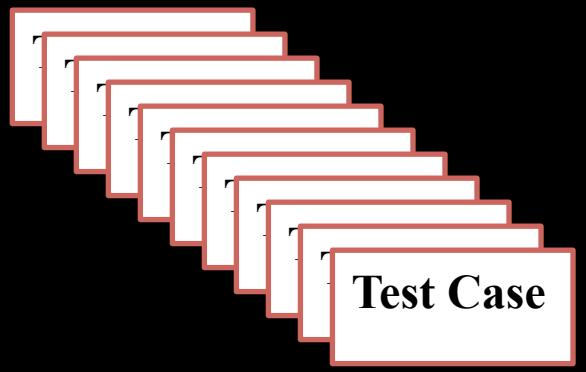


Test for *success* or test for *failure*

Test Objectives and Test Cases

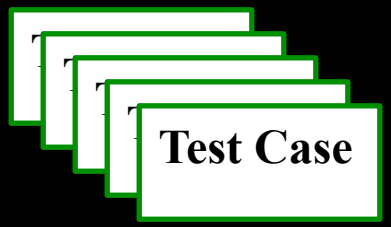
Test objective 1
functionality





Test objective 2
performance

Test objective 3
security

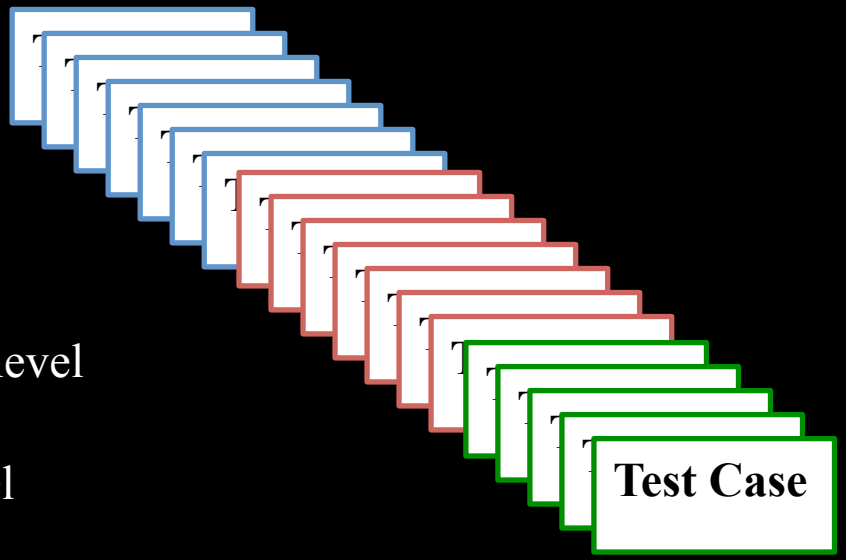


Test Objectives and Test levels

Test objective 1 – Unit level

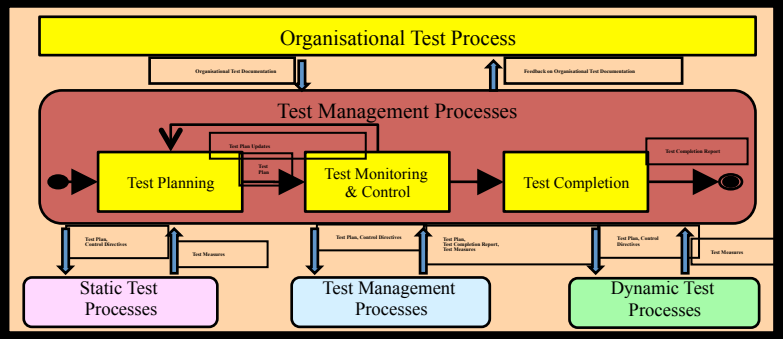
Test objective 1 – Integration level

Test objective 1 – System level



Test Suite

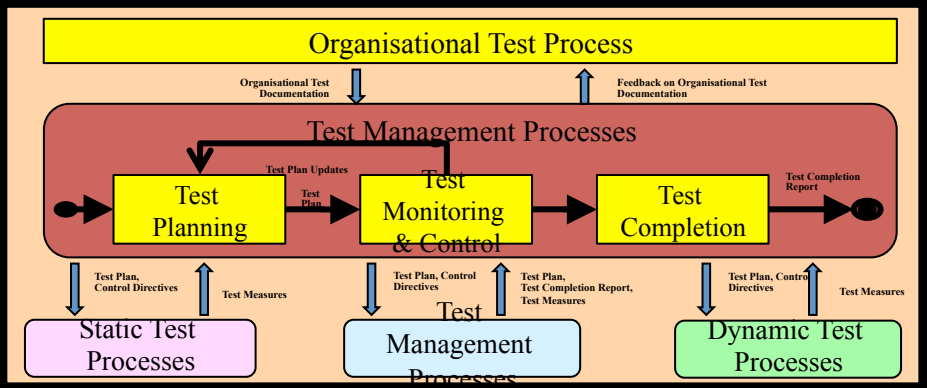
- ✓ A collection of *test cases* that tests a software system
- ✓ A test suite can contain groups of test cases for different object, objectives, and techniques, and information on the system configuration to be used during testing.
- ✓ A group of test cases also contain information on how to setup the test environment.



Test Object, Objectives & Techniques

1. Identify - Test Object ➔ What!
2. Define - Test Objective ➔Why!
3. Select - Test Technique ➔How!

A Test Suite



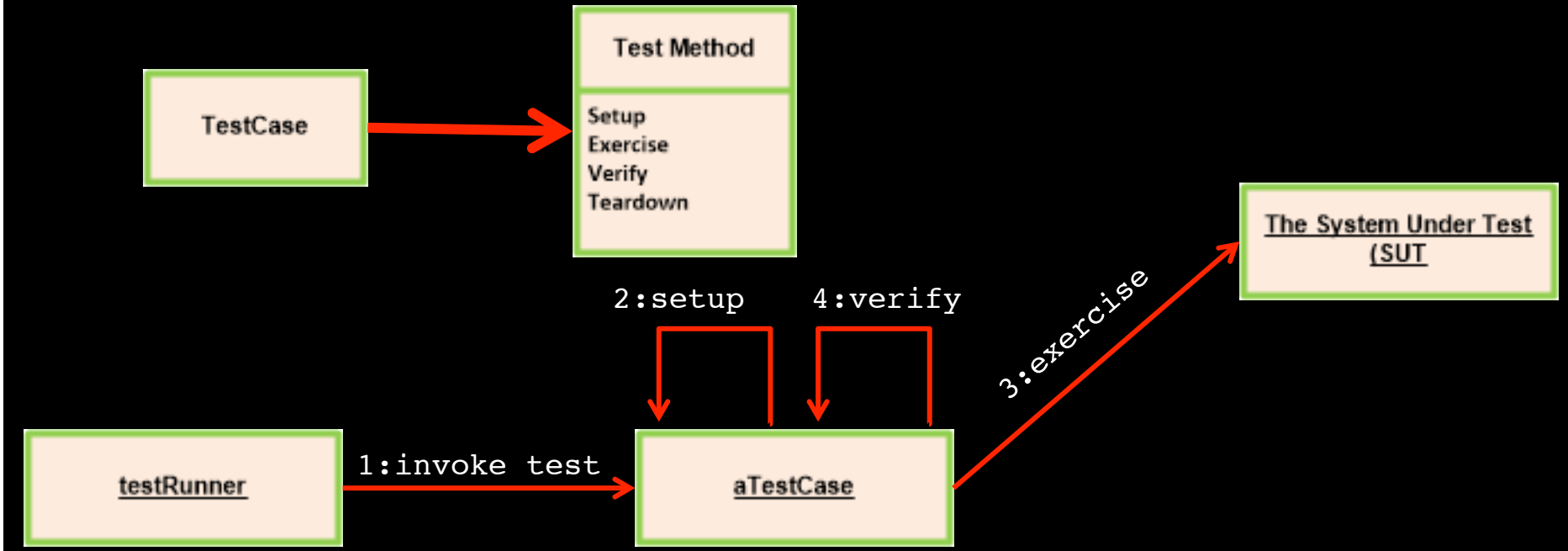
xUnit – a Testing Framework Architecture

1. Testing Frameworks are used for *repeatable automated* tests
2. Provides a structure for implementing *test suites*
3. Simplifies the work with developing "*test drivers*"

xUnit includes:

- *Assertions* for testing expected and actual results
- Support for "sharing" test environments among several test cases, test fixtures.
- Define and set up your test suites
- Graphical or text based "testrunners"

xUnit Pattern – four phase testing



xUnit Architecture – Tests

- ✓ xUnit is used for testing …
  - … a complete test object, for example a class
  - … a part of a test object, for example a method or a small set of cooperating methods
  - … interactions among several test objects (integration testing)
- ✓ A test class contains more than one test
- ✓ Each test case is represented by a method

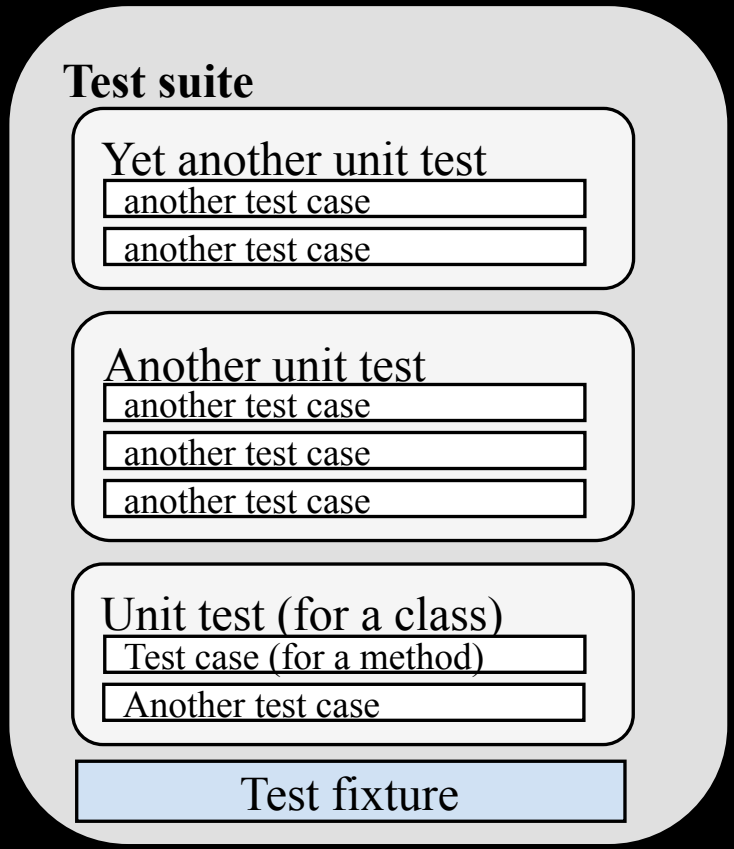
xUnit Architecture – Tests

- ✓ The test class also includes:
  - "Test runners" that execute the tests ("main")
  - A set of test methods (test cases) including assertions
  - Fixtures for
    - Set up state before test
    - Reset state after single test case
    - or after all tests

    } Sequencing
- ✓ Read more junit.org

Once more, this time a picture

- ✓ A **unit test** tests the methods in a class
- ✓ A test case tests a method
- ✓ Each method is tested by several test cases
- ✓ A test suite combines several test cases
- ✓ Test fixtures, set up and reset state "test runners" execute test cases or complete test suites



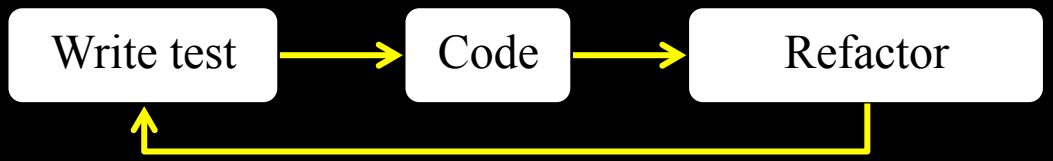
How to write tests in xUnit – Example JUnit

- ✓ @test
- ✓ Use methods in the `junit.framework.assert` class
- ✓ Each method checks the assertion and reports back to the "test runner" if a test was ok or not.
- ✓ The "Test runner" uses the result for tester feedback
- ✓ All test methods return `void`
- ✓ Some examples of methods in junit.framework.assert
  - `assertTrue (boolean)`
  - `assertTrue (String, boolean)`
  - `assertEquals (Object, Object)`
  - `assertNull (Object)`
  - `Fail (String)`

Improve Controllability – Test Driven Development

If it's worth building, it's worth testing.
If it's not worth testing, why are you wasting your time working on it?

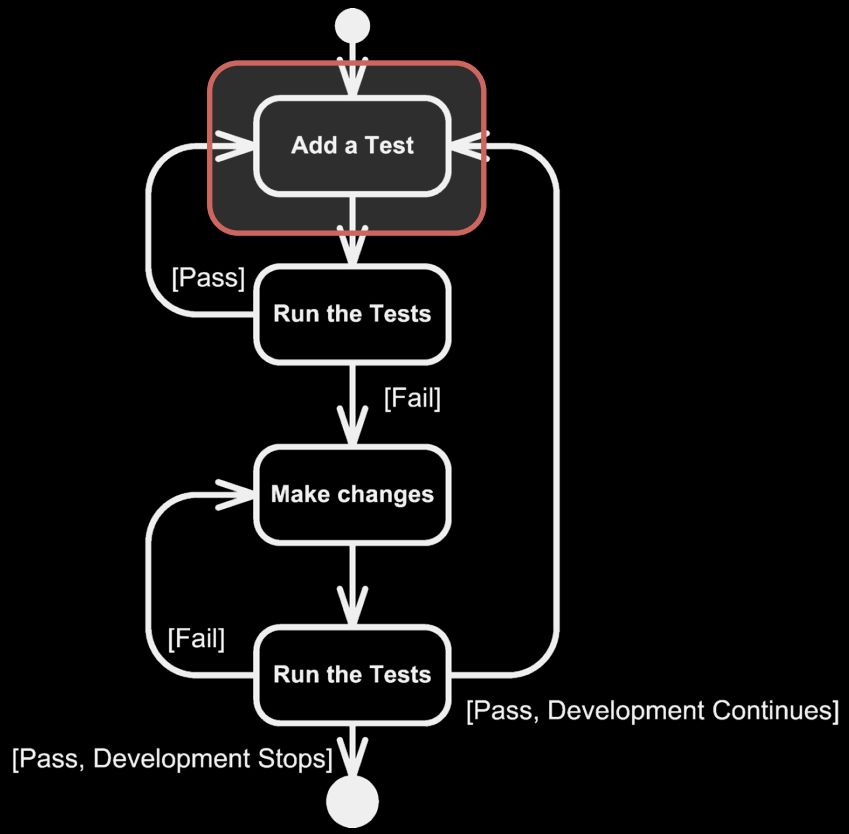
- ✓ The test is a *design specification*
  - *Pre and Post condition*
  - *Method signature*
- ✓ Forces the designers to think **"how will this be used"** before **"how should it be implemented"**



TDD is not about *Testing*, it is about *Design*!

Test First Development

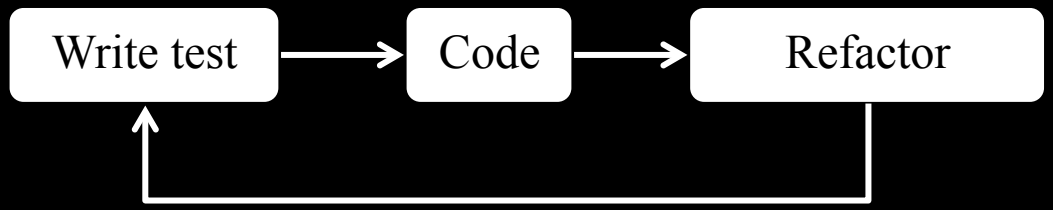
- ✓ One of the fundamental principles
- ✓ **It is not a process!**
- ✓ It is an *engineering practice* in *agile development*
- ✓ Supports test levels and objectives



Acceptance – TDD (ATDD).

- ✓ Write an acceptance test or behavioral specification
- ✓ Write exactly the amount of code needed to pass the test
- ✓ The purpose of Acceptance-TDD is executable specifications
- ✓ Often referred to as *Behavior Driven Development* (BDD).

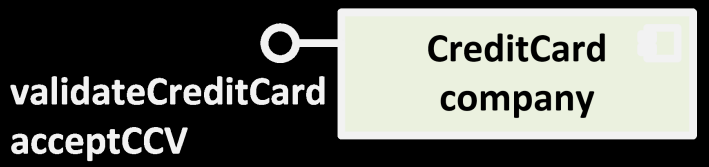
Refactoring



- ✓ Modify the code structure without changing the behavior
- ✓ Examples
  - Change name
  - Factor out methods or interfaces
  - Remove method calls (in-lining)
  - Pull up /Push down
    - Move a method from a subclass to a superclass
    - Move a method from a superclass to a subclass

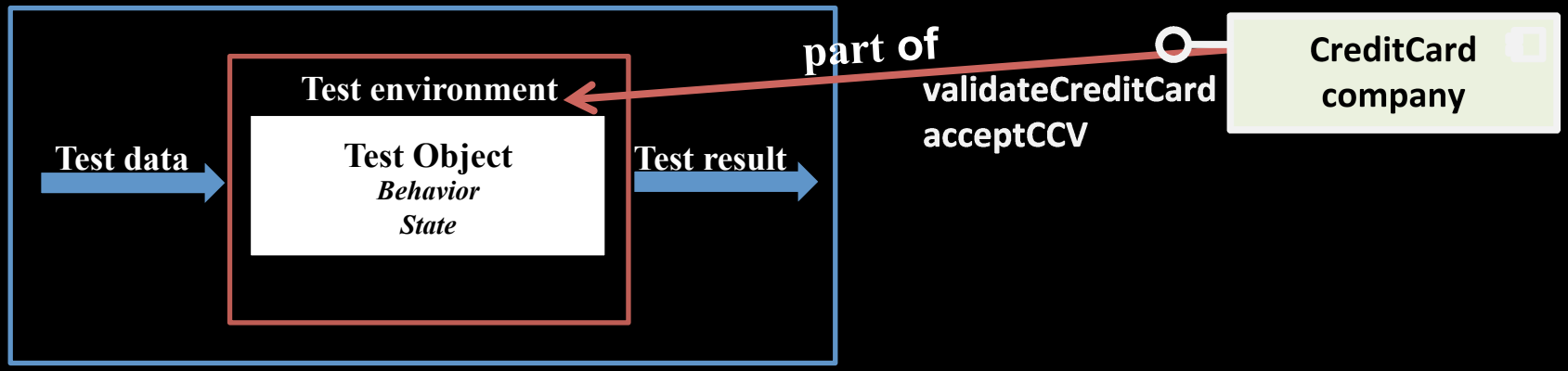
Mock Objects

- ✓ Special objects must be designed for testing
  - Drivers
  - Stubs (Mockups)
- ✓ Objects at the “edge of a system”,
  - ✓ Not “controllable”
  - ✓ Not “observable”
- ✓ Examples
  - Web Services
  - Data Access Layer
  - User Interface



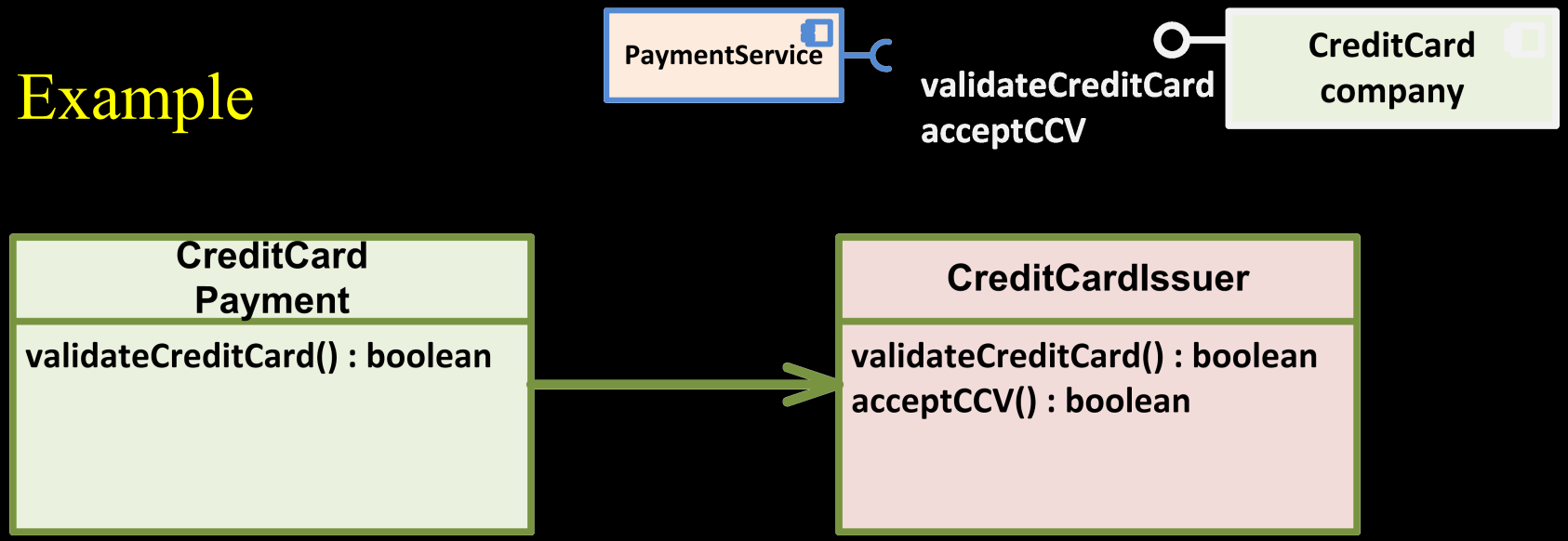
Example

- ✓ What type of behaviors may we expect from the service?
- ✓ We test code that depends on the service.



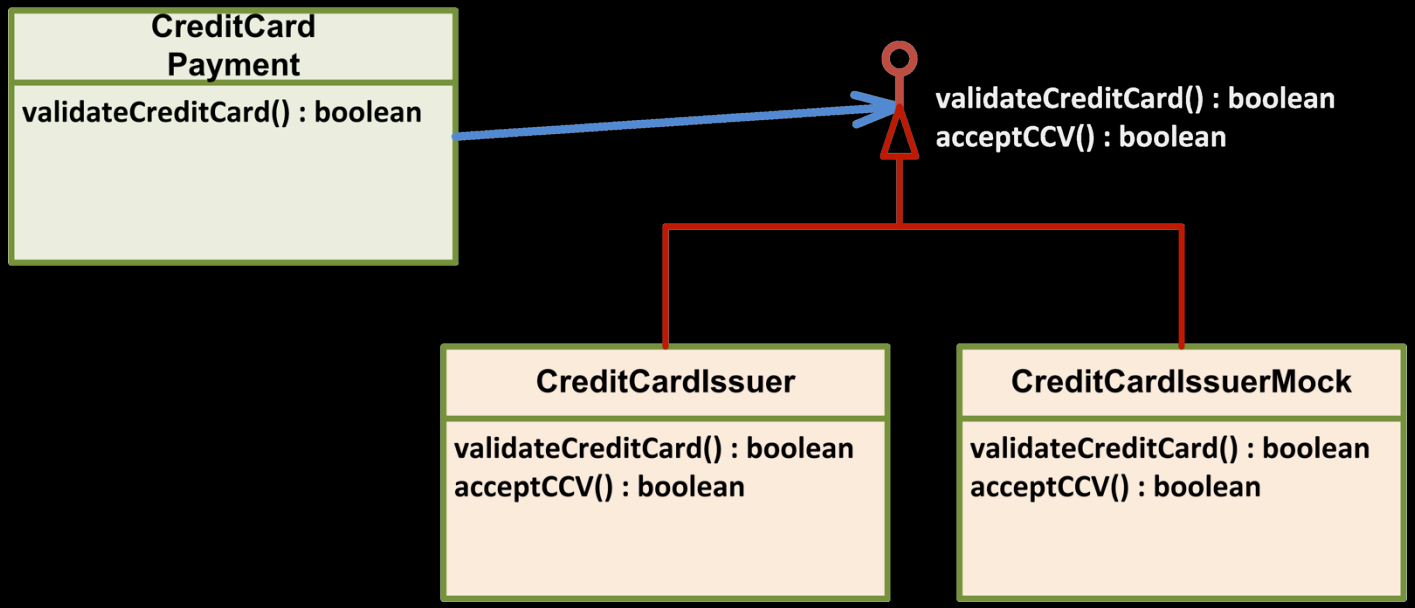
- ✓ In testing we want to control and observe!!

Example



How can you test `validateCredtitCard` if you can't control `CreditCardIssuer`?

Example contd.



Today's Takeaways

- ✓ Testing is challenging
- ✓ Testing Integrity
  - Planning
  - Design
  - Implementation
  - Execution
- ✓ Testing techniques
  - Static
  - Dynamic, more next week