

# Testing Retrospective

© 2009 by Jonathan Aldrich  
Portions © 2007 by William L Scherlis  
used by permission

No part may be copied or used  
without written permission.

Primary source: Kaner, Falk, Nguyen.  
Testing Computer Software (2nd Edition).

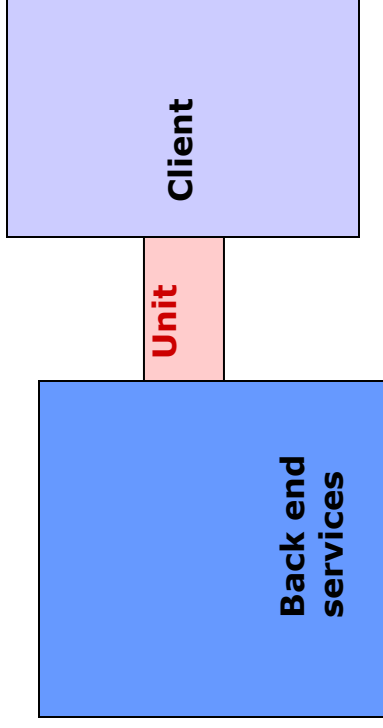
**Jonathan Aldrich**  
Assistant Professor  
Institute for Software Research

School of Computer Science  
Carnegie Mellon University  
jonathan.aldrich@cs.cmu.edu  
+1 412 268 7278

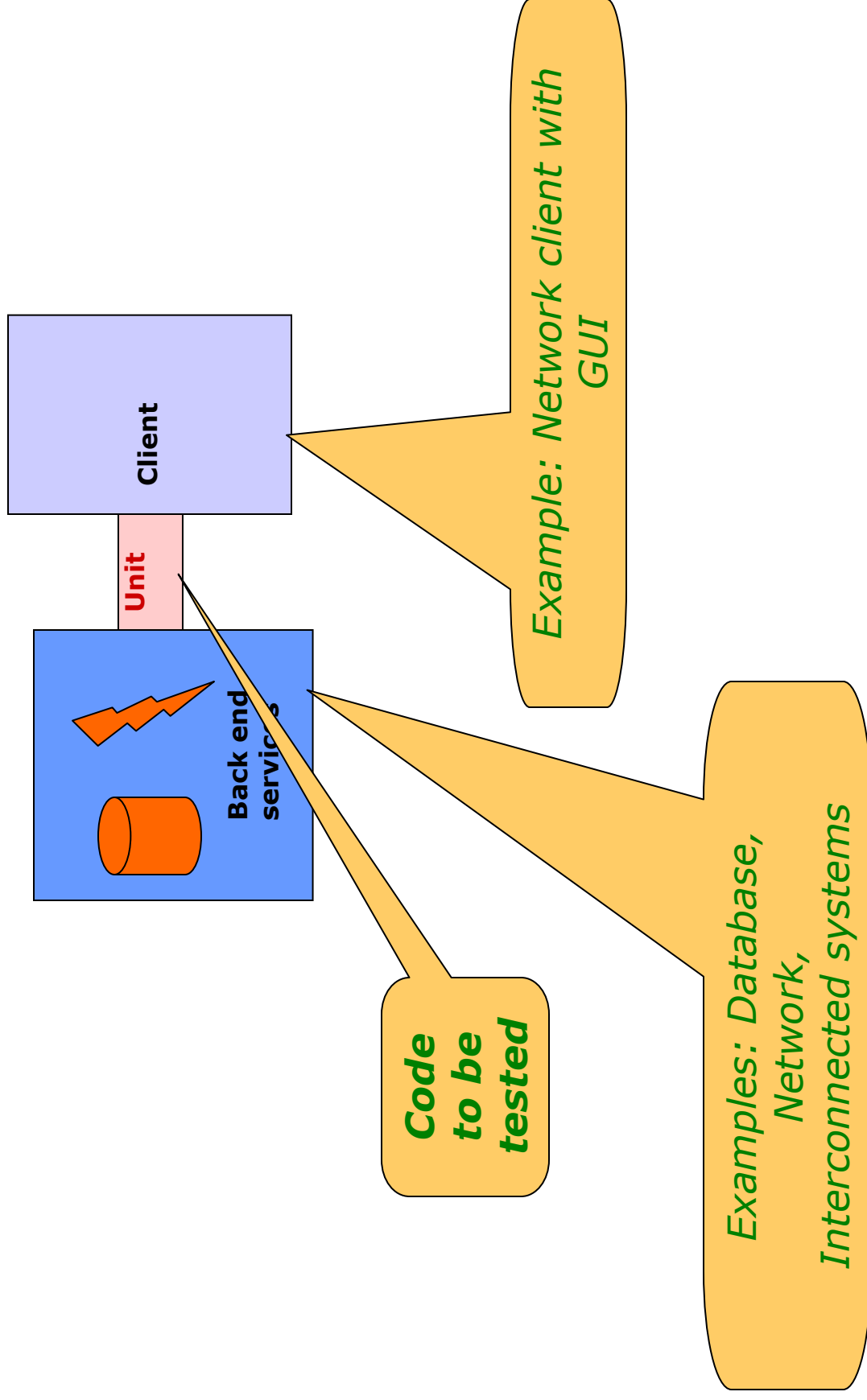
# Testing – The Big Questions

- 1. What is testing?**
  - And why do we test?
- 2. To what standard do we test?**
  - Specification of behavior and quality attributes
- 3. How do we select a set of good tests?**
  - Functional (black-box) testing
  - Structural (white-box) testing
- 4. How do we assess our test suites?**
  - Coverage, Mutation, Capture/Recapture...
- 5. What are effective testing practices?**
  - Levels of structure: unit, integration, system...
  - **Design for testing: scaffolding**
  - Effective testing practices
  - How does testing integrate into lifecycle and metrics?
- 6. What are the limits of testing?**
  - What are complementary approaches?
    - *Inspections*
    - *Static and dynamic analysis*

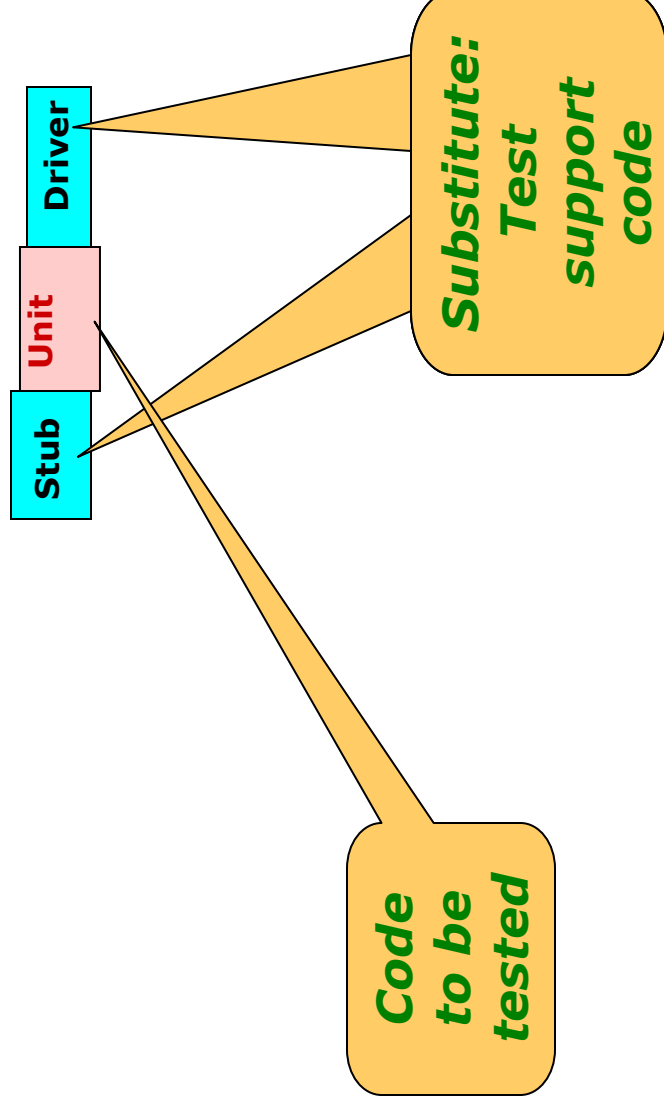
# Unit Test and Scaffolding



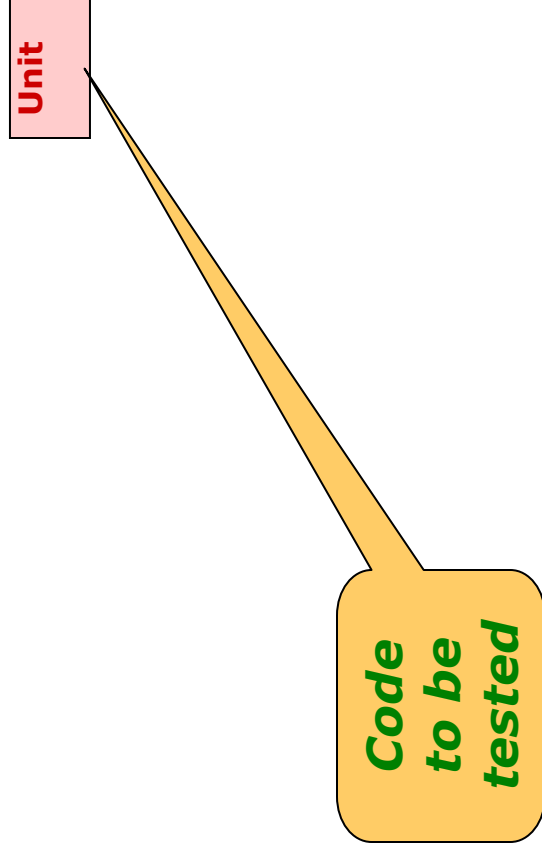
# Unit Test and Scaffolding



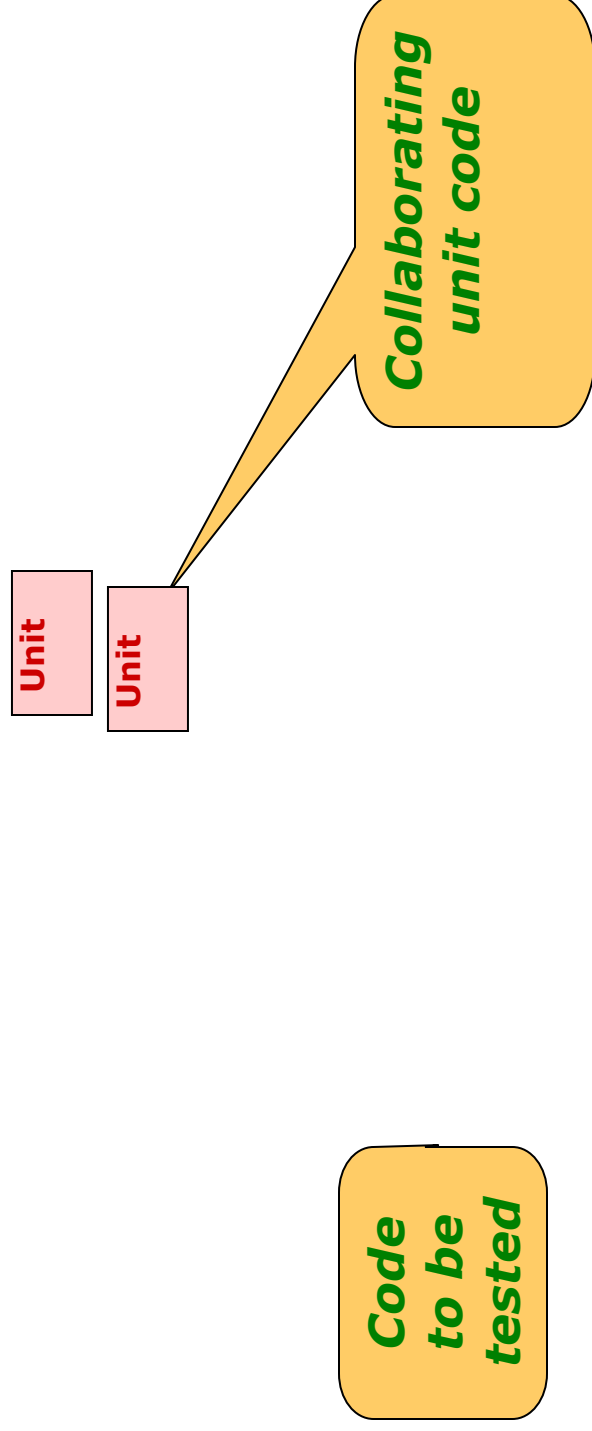
# Unit Test and Scaffolding



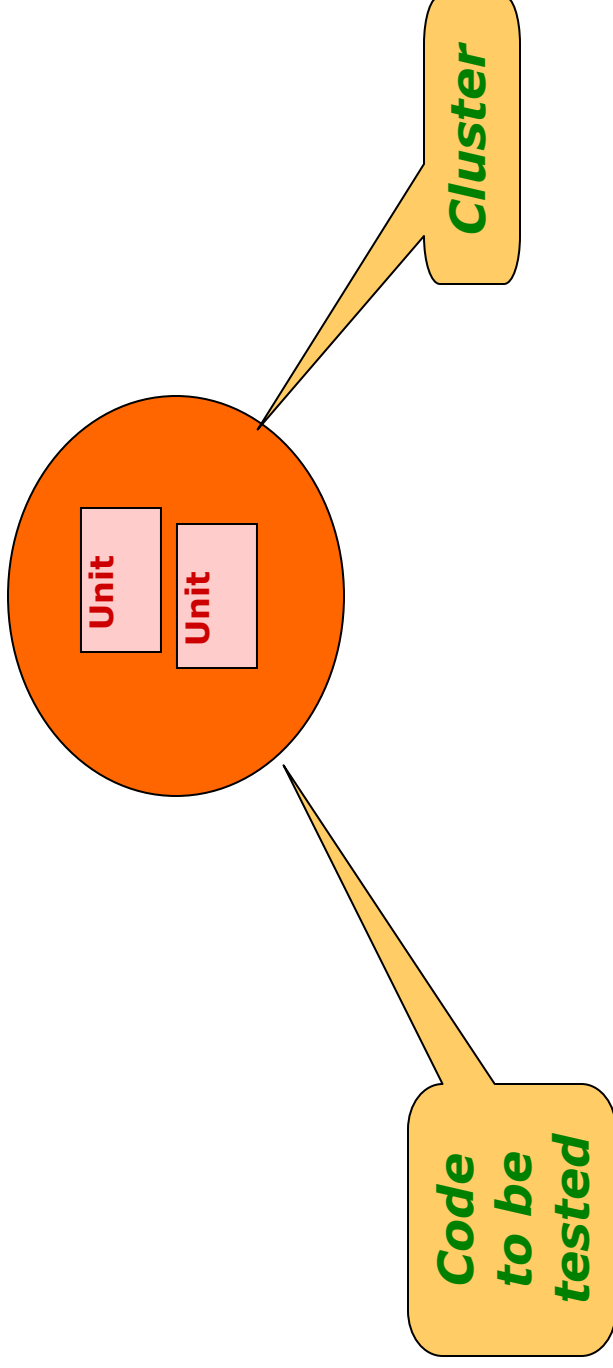
# Unit Test and Scaffolding



# Unit Test and Scaffolding

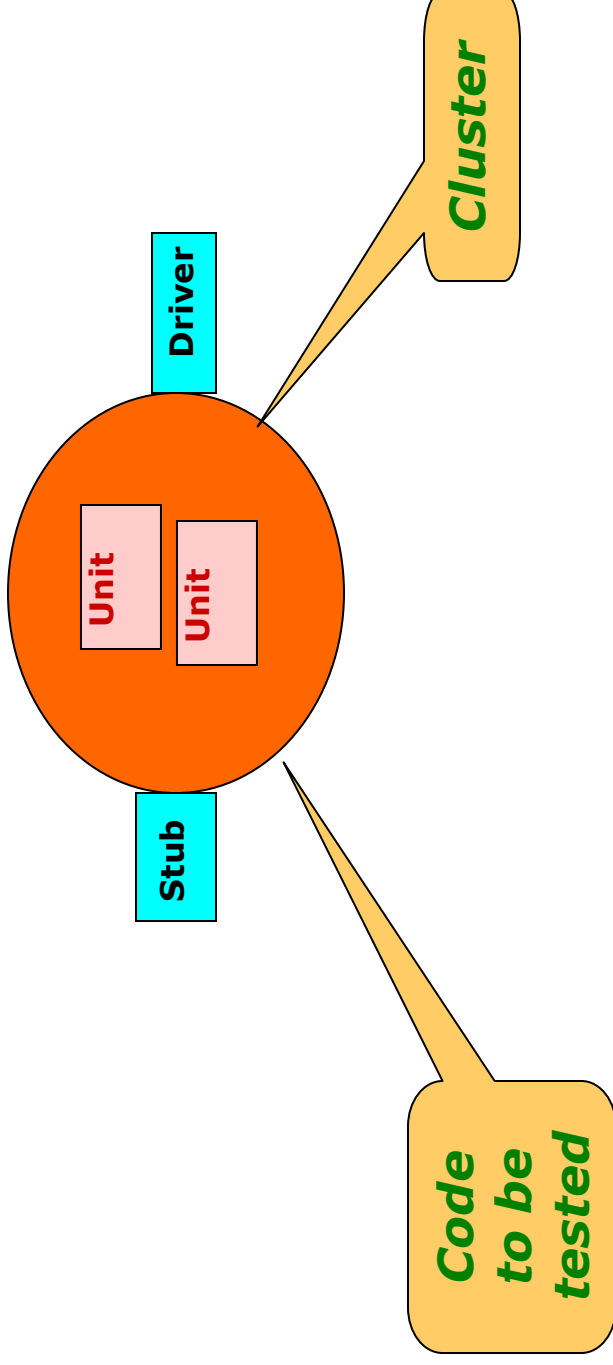


# Unit Test and Scaffolding





# Unit Test and Scaffolding

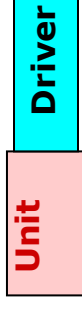


# Techniques for Unit Testing 1: Scaffolding

- Use “scaffold” to simulate external code
- External code – scaffold points
  1. Client code
  2. Underlying service code
- 1. Client API
  - Model the software client for the service being tested
  - Create a **test driver**
  - Object-oriented approach:
    - Test individual calls and sequences of calls



Testers write  
driver code

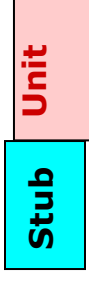


# Techniques for Unit Testing 1: Scaffolding

- Use “scaffold” to simulate external code
- External code – scaffold points
  1. Client code
  2. Underlying service code
- **2. Service code**
  - Underlying services
    - Communication services
    - Model behavior through a communications interface
    - Database queries and transactions
  - Network/web transactions
  - Device interfaces
    - Simulate device behavior and failure modes
  - File system
    - Create file data sets
    - Simulate file system corruption
  - Etc
- Create a set of **stub** services or **mock** objects
  - *Minimal* representations of APIs for these services

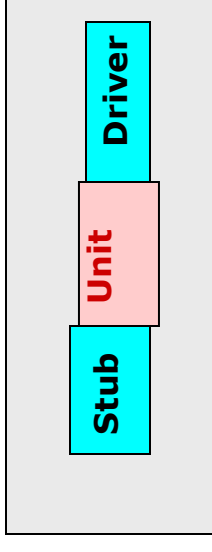


Testers write  
stub code



# Scaffolding

- **Purposes**
  - Catch bugs early
    - Before client code or services are available
  - Limit the scope of debugging
    - Localize errors
  - Improve coverage
    - System-level tests may only cover 70% of code [Massol]
    - Simulate unusual error conditions – test internal robustness
  - Validate internal interface/API designs
    - Simulate clients in advance of their development
    - Simulate services in advance of their development
  - Capture developer intent (in the absence of specification documentation)
    - A test suite formally captures elements of design intent
    - Developer documentation
  - Enable division of effort
    - Separate development / testing of service and client
  - Improve low-level design
    - Early attention to ability to test – “testability”



## Barriers to Scaffolding

- For some applications scaffolding is difficult
  - Wide interface between components
    - Must replicate entire interface
    - Automated tools can help
  - Complex behavior exercised in tests
    - Actual implementation may be simpler than scaffolding
    - Scaffolding may not be worthwhile here!
- May be difficult to set up data structure for tests
  - Design principle - create special constructors for testing

# Testing – The Big Questions

- 1. What is testing?**
  - And why do we test?
- 2. To what standard do we test?**
  - Specification of behavior and quality attributes
- 3. How do we select a set of good tests?**
  - Functional (black-box) testing
  - Structural (white-box) testing
- 4. How do we assess our test suites?**
  - Coverage, Mutation, Capture/Recapture...
- 5. What are effective testing practices?**
  - Levels of structure: unit, integration, system...
  - Design for testing
  - **Effective testing practices**
  - How does testing integrate into lifecycle and metrics?
- 6. What are the limits of testing?**
  - What are complementary approaches?
    - *Inspections*
    - *Static and dynamic analysis*

## 5c. Integration/System Testing

1. Do incremental integration testing
  - Test several modules together
  - Still need scaffolding for modules not under test
- Avoid “big bang” integrations
  - Going directly from unit tests to whole program tests
  - Likely to have many big issues
  - Hard to identify which component causes each
- Test interactions between modules
  - Ultimately leads to end-to-end system test
- Used focused tests
  - Set up subsystem for test
  - Test specific subsystem- or system-level features
    - no “random input” sequence
  - Verify expected output



## 5c. Frequent (Nightly) Builds

- 2. Build a release of a large project every night
  - Catches integration problems where a change “breaks the build”
    - Breaking the build is a BIG deal—may result in midnight calls to the responsible engineer
  - Use test automation
    - Upfront cost, amortized benefit
    - Not all tests are easily automated – manually code the others
- Run simplified “smoke test” on build
  - Tests basic functionality and stability
  - Often: run by programmers before check-in
  - Provides rough guidance prior to full integration testing

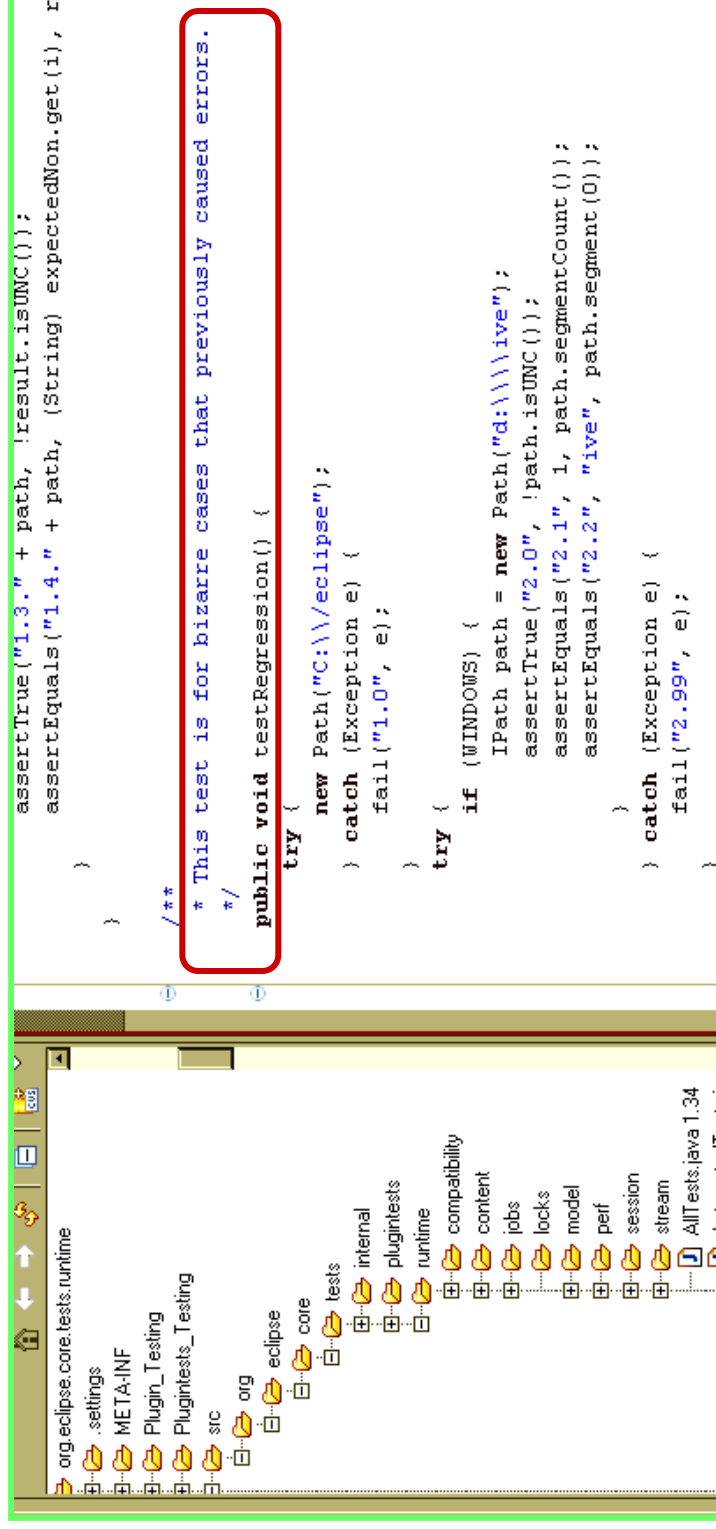




# Practices – Regressions

## 3. Use regression tests

- Regression tests: run every time the system changes
- **Goal: catch new bugs introduced by code changes**
  - Check to ensure fixed bugs stay fixed
    - New bug fixes often introduce new issues/bugs
  - Incrementally add tests for new functionality



```
assertTrue("1.3." + path, !result.isUNC());
assertEquals("1.4." + path, (String) expectedNon.get(i), r
)
}
}
/**
 * This test is for bizarre cases that previously caused errors.
 */
public void testRegression() {
    try {
        new Path("C:\\eclipse");
    } catch (Exception e) {
        fail("1.0", e);
    }
    try {
        if (WINDOWS) {
            IPath path = new Path("d:\\\\ive");
            assertTrue("2.0", !path.isUNC());
            assertEquals("2.1", 1, path.segmentCount());
            assertEquals("2.2", "ive", path.segment(0));
        } catch (Exception e) {
            fail("2.99", e);
        }
    }
}
```

## Practices – Acceptance, Release, Integrity Tests

4. Acceptance tests (by customer)
  - Tests used by customer to evaluate quality of a system
  - Typically subject to up-front negotiation
5. Release Test (by provider, vendor)
  - Test release CD
    - Before manufacturing!
  - Includes configuration tests, virus scan, etc
  - Carry out entire install-and-run use case
6. Integrity Test (by vendor or third party)
  - Independent evaluation before release
  - Validate quality-related claims
  - Anticipate product reviews, consumer complaints
  - Not really focused on bug-finding

# Practices: Reporting Defects

## 7. Develop good defect reporting practices

- Reproducible defects
  - Easier to find and fix
  - Easier to validate
    - Built-in regression test
  - Increased confidence
- Simple and general
  - More value doing the fix
  - Helps root-cause analysis
- Non-antagonistic
  - State the problem
  - Don't blame

The screenshot shows the Eclipse bug reporting interface for Bugzilla Bug 141261. The page title is "Eclipse bugs" and the bug title is "Bugzilla Bug 141261 crash - Shell create, RepositionWindow() - Unexpected Eclipse crash.RC3 (JavaNativeCrash)". The bug is reported by Igor Goldenberg. The interface includes fields for Hardware (Macintosh), OS (Mac OS), Version (3.2), Priority (P3), Severity (major), and Target (Milestone). The bug is assigned to Silenio Quarti. The QA Contact, URL, Summary, Whiteboard, and Keywords fields are empty. The bug depends on Bug 141261. The interface also includes a table for Attachments and a section for Additional Comments.

**Eclipse** bugzilla 2.20.3

crash - Shell create, RepositionWindow() - Unexpected Eclipse crash.RC3 (JavaNativeCrash) Last modified: 2006-11-14 17:46:58

Bug List: (31 of 200) [First](#) [Last](#) [Prev](#) [Next](#) [Show last search results](#) [Search page](#) [Enter new bug](#)

**[Eclipse] 141261** **Reporter:** Igor Goldenberg [Add CC:](#)

**Bug#:** 141261 **Hardware:** Macintosh **OS:** Mac OS **Version:** 3.2 **Priority:** P3 **Severity:** major **Target:** — **Milestone:** —

**Product:** Platform **Component:** SWT **Status:** NEW **Assigned To:** Silenio Quarti <Silenio\_Quarti@ca.ibm.com>

**QA Contact:**   
**URL:**   
**Summary:** crash - Shell create, RepositionWindow() - Unexpected Eclipse c  
**Status:**   
**Whiteboard:**   
**Keywords:**

Remove selected CCs

| Attachment   | Type | Created | Size | Actions                  |
|--|------|---------|------|--------------------------|
| <a href="#">Create a New Attachment</a> (proposed patch, testcase, etc.) |      |         |      | <a href="#">View All</a> |

**Bug 141261 depends on:**  [Show dependency tree](#)

**Bug 141261 blocks:**

**Votes:** 0 [Show votes for this bug](#) [Vote for this bug](#)

**Additional Comments:**

# Practices: Social Issues

- 8. Respect social issues of testing
  - There are differences between developer and tester culture
  - Acknowledge that testers often deliver bad news
  - Avoid using defects in performance evaluations
    - Is the defect real?
    - Bad will within team
  - Work hard to detect defects before integration testing
    - Easier to narrow scope and responsibility
    - Less adversarial
  - Issues vs. defects

[Reassign bug to pde-ui-inbox@eclipse.org](#)  
[Reassign bug to default assignee and QA contact of selected component](#)  
[Commit](#)

[View Bug Activity](#) | [Format For Printing](#) | [Clone This Bug](#)

**Description:** [\[reply\]](#)      **Opened:** 2005-07-25 07:03

I didn't even know that there was an undo feature inside the GUI editor, but today I accidentally pressed CTRL-Z instead of CTRL-S and the undo started... crashed.

Try adding some extension in the extensions page and then press CTRL-Z.

This is actually two bugs imho;

1. The details is very very poor so I really don't know what happened. A stacktrace would be great for debugging.
2. The undo obviously does not work correctly.

here is a screenshot of the crash:  
[http://memo.minimum.se/eclipse\\_crashes/eclipse\\_undo\\_crash.png](http://memo.minimum.se/eclipse_crashes/eclipse_undo_crash.png)

I don't have time for extensive repro testing atm, maybe someone else can assist with this and see if they can get the plugin.xml editor to crash using weird combinations of editing and CTRL-Z undoing.

## Practices: Root cause analysis

- 9. How can defect analysis help prevent later defects?
  - Identify the “root causes” of frequent defect types, locations
    - Requirements and specifications?
    - Architecture? Design? Coding style? Inspection?
  - Try to find all the paths to a problem
    - If one path is common, defect is higher priority
    - Each path provides more info on likely cause
  - Try to find related bugs
    - Helps identify underlying root cause of the defect
    - Can use to get simpler path to problem
      - This can mean easier to fix
  - Identify the most serious consequences of a defect

# Testing – The Big Questions

- 1. What is testing?**
  - And why do we test?
- 2. To what standard do we test?**
  - Specification of behavior and quality attributes
- 3. How do we select a set of good tests?**
  - Functional (black-box) testing
  - Structural (white-box) testing
- 4. How do we assess our test suites?**
  - Coverage, Mutation, Capture/Recapture...
- 5. What are effective testing practices?**
  - Levels of structure: unit, integration, system...
  - Design for testing
  - Effective testing practices
  - **How does testing integrate into lifecycle and metrics?**
- 6. What are the limits of testing?**
  - What are complementary approaches?
    - *Inspections*
    - *Static and dynamic analysis*

## 5d. Testing and Lifecycle Issues

1. Testing issues should be addressed at every lifecycle phase
  - **Initial negotiation**
    - Acceptance evaluation: evidence and evaluation
    - Extent and nature of specifications
  - **Requirements**
    - Opportunities for early validation
    - Opportunities for specification-level testing and analysis
    - Which requirements are testable: functional and non-functional
  - **Design**
    - Design inspection and analysis
    - Designing for testability
      - Interface definitions to facilitate unit testing
  - **Follow both top-down and bottom-up unit testing approaches**
    - Top-down testing
      - Test full system with stubs (for undeveloped code).
      - Tests design (structural architecture), when it exists.
    - Bottom-up testing
      - Units → Integrated modules → system

## Lifecycle issues

### 2. Favor unit testing over integration and system testing

- **Unit tests find defects earlier**
  - Earlier means less cost and less risk
  - During design, make API specifications specific
    - Missing or inconsistent interface (API) specifications
    - Missing representation invariants for key data structures
    - What are the unstated assumptions?
      - Null refs ok?
      - Pass out this exception ok?
      - Integrity check responsibility?
      - Thread creation ok?
- **Over-reliance on system testing can be risky**
  - Possibility for finger pointing within the team
  - Difficulty of mapping issues back to responsible developers
  - Root cause analysis becomes blame analysis



# Test Plan

- 3. Create a QA plan document
  - Which quality techniques are used and for what purposes
  - Overall system strategy
    - Goals of testing
      - Quality targets
      - Measurements and measurement goals
    - What will be tested/what will not
      - Don't forget quality attributes!
    - Schedule and priorities for testing
      - Based on hazards, costs, risks, etc.
    - Organization and roles: division of labor and expertise
    - Criteria for completeness and deliverables
  - Make decisions regarding when to unit test
    - There are differing views
      - **CleanRoom**: Defer testing. Use separate test team
      - **Agile**: As early as possible, even before code, integrate into team

|       |  |
|-------|--|
| 1     | Scope                                      |
| 1.1   | System Overview                            |
| 2     | Reference Documents                        |
| 3     | Software Test Environment                  |
| 4     | Test Identification                        |
| 4.1   | General Information                        |
| 4.1.1 | Test Level                                 |
| 4.1.2 | Test Classes                               |
| 4.2   | Planned Tests                              |
| 4.2.1 | Test 1 – Linear Operators                  |
| 4.2.2 | Test 2 – Convergence of Multifluid Project |
| 4.2.3 | Test 3 – Fixed-boundary diffusion solver   |
| 4.2.4 | Test 4 – Upwind advection                  |
| 4.2.5 | Test 5 – Fixed-boundary projection test    |
| 4.2.6 | Test 6 – Surface Tension Test              |
| 4.2.7 | Test 7 – Multifluid system test            |
| 4.2.8 | Test 8 – Multifluid AMR test               |
| 4.2.9 | Test 9 – Multifluid system regression test |
| 5     | Test Schedules                             |
| 6     | Bug Tracking                               |
| 7     | Requirements Traceability                  |

## Test Strategy Statement

- **Examples:**
  - We will release the product to friendly users after a brief internal review to find any truly glaring problems. The friendly users will put the product into service and tell us about any changes they'd like us to make.
  - We will define use cases in the form of sequences of user interactions with the product that represent ... the ways we expect normal people to use the product. We will augment that with stress testing and abnormal use testing (invalid data and error conditions). Our top priority is finding fundamental deviations from specified behavior, but we will also use exploratory testing to identify ways in which this program might violate user expectations.
  - We will perform parallel exploratory testing and automated regression test development and execution. The exploratory testing will focus on validating basic functions (capability testing) to provide an early warning system for major functional failures. We will also pursue high-volume random testing where possible in the code.

[adapted from Kaner, Bach, Pettichord, Lessons Learned in Software Testing ]

## Why Produce a Test Plan?

4. Ensure the test plan addresses the needs of stakeholders
  - **Customer: may be a required product**
    - Customer requirements for operations and support
    - Examples
      - Government systems integration
      - Safety-critical certification: avionics, health devices, etc.
  - **A separate test organization may implement part of the plan**
    - “IV&V” – Independent verification and validation
  - **May benefit development team**
    - Set priorities
      - Use planning process to identify areas of hazard, risk, cost
  - **Additional benefits – the plan is a team product**
    - Test quality
      - Improve coverage via list of features and quality attributes
      - Analysis of program (e.g. boundary values)
      - Avoid repetition and check completeness
    - Communication
      - Get feedback on strategy
      - Agree on cost, quality with management
    - Organization
      - Division of labor
      - Measurement of progress

# Defect Tracking

## 5. Track defects and issues

- **Issue: Bug, feature request, or query**
  - May not know which of these until analysis is done, so track in the same database (Issuezilla)
- **Provides a basis for measurement**
  - Defects reported: which lifecycle phase
  - Defects repaired: time lag, difficulty
  - Defect categorization
  - Root cause analysis (more difficult!)
- **Provides a basis for division of effort**
  - Track diagnosis and repair
  - Assign roles, track team involvement
- **Facilitates communication**
  - Organized record for each issue
  - Ensures problems are not forgotten
- **Provides some accountability**
  - Can identify and fix problems in process
    - Not enough detail in test reports
    - Not rapid enough response to bug reports
  - Should not be used for HR evaluation

----- [Comment #4](#) From [Clare Cary](#) 2006-10-11 15:28 [reply] -----  
(In reply to comment #3)  
> I'm sorry but we really don't have enough details to be able  
> problem. Could you try with another VM?  
>  
Problem didn't happen with another JRE - just the sun JRE.  
  
----- [Comment #5](#) From [Oleg Besedin](#) 2006-10-11 15:38 [reply] -----  
This looks like a duplicate of the [bug\\_92250](#). Could you try if  
with `-XX:MaxPermSize=256m` ?  
  
----- [Comment #6](#) From [Fascal Rapicault](#) 2006-10-12 12:57 [reply] -----  
After further investigation, setting the permenspace to 1024 r  
problem.  
\*\*\* This bug has been marked as a duplicate of [92250](#) \*\*\*  
  
----- [Comment #7](#) From [Clare Cary](#) 2006-10-12 15:18 [reply] -----  
This problem is still occurring on the dependent product with P  
to 1024M. Please investigate.  
  
----- [Comment #8](#) From [John Arthorne](#) 2006-10-12 17:24 [reply] -----

Bug List: (48 of 200) [First](#) [Last](#)

[Eclipse] 160502  
Bug#: 160502

Product: Platform  
Component: Runtime  
Status: REOPENED  
Resolution: platform-runtime-  
Assigned To: <platform-runtime-  
                  inbox@eclipse.org>

QA Contact:  
URL:  
Summary: JVM crash at random intervals on SUSE 9 with Sun JRE 1.5  
Status  
Whiteboard:  
Keywords: vm

Hardware: PC  
OS: Linux  
Version: 321  
Priority: P3  
Severity: blocker  
Target Milestone:

Reporter: [Clare Cary](#) <ccary@ca.ibm.com>  
Add CC:  
CC: ccary@ca.ibm.com  
          john\_arthorne@ca.ibm.com  
 Remove selected CCs

| Attachment  | Type       | Created          | Size      | Actions              |
|---|------------|------------------|-----------|----------------------|
| <a href="#">screenshot of crash</a>   | image/jpeg | 2006-10-11 12:14 | 131.55 KB | <a href="#">Edit</a> |
| <a href="#">Create a New Attachment</a> (proposed patch, testcase, etc.) <a href="#">View All</a> |            |                  |           |                      |

Bug 160502 depends on:  
Bug 160502 blocks:  
Votes: 0 [Show votes for this bug](#) [Vote for this bug](#) [Show dependency tree](#)

# Questions?