ECE444: Software Engineering

Intro to Quality Assurance

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Learning Goals

- Understand process aspects of QA
- Describe the tradeoffs of QA techniques
- Select an appropriate QA technique for a given project and quality attribute
- Apply testing and test automation for functional correctness
- Understand opportunities and challenges for testing quality attributes; enumerate testing strategies to help evaluate the following quality attributes: usability, reliability, security, robustness (both general and architectural), performance, integration.
- Discuss the limitations of testing

QA is Hard

“We had initially scheduled time to write tests for both front and back end systems, although this never happened.”

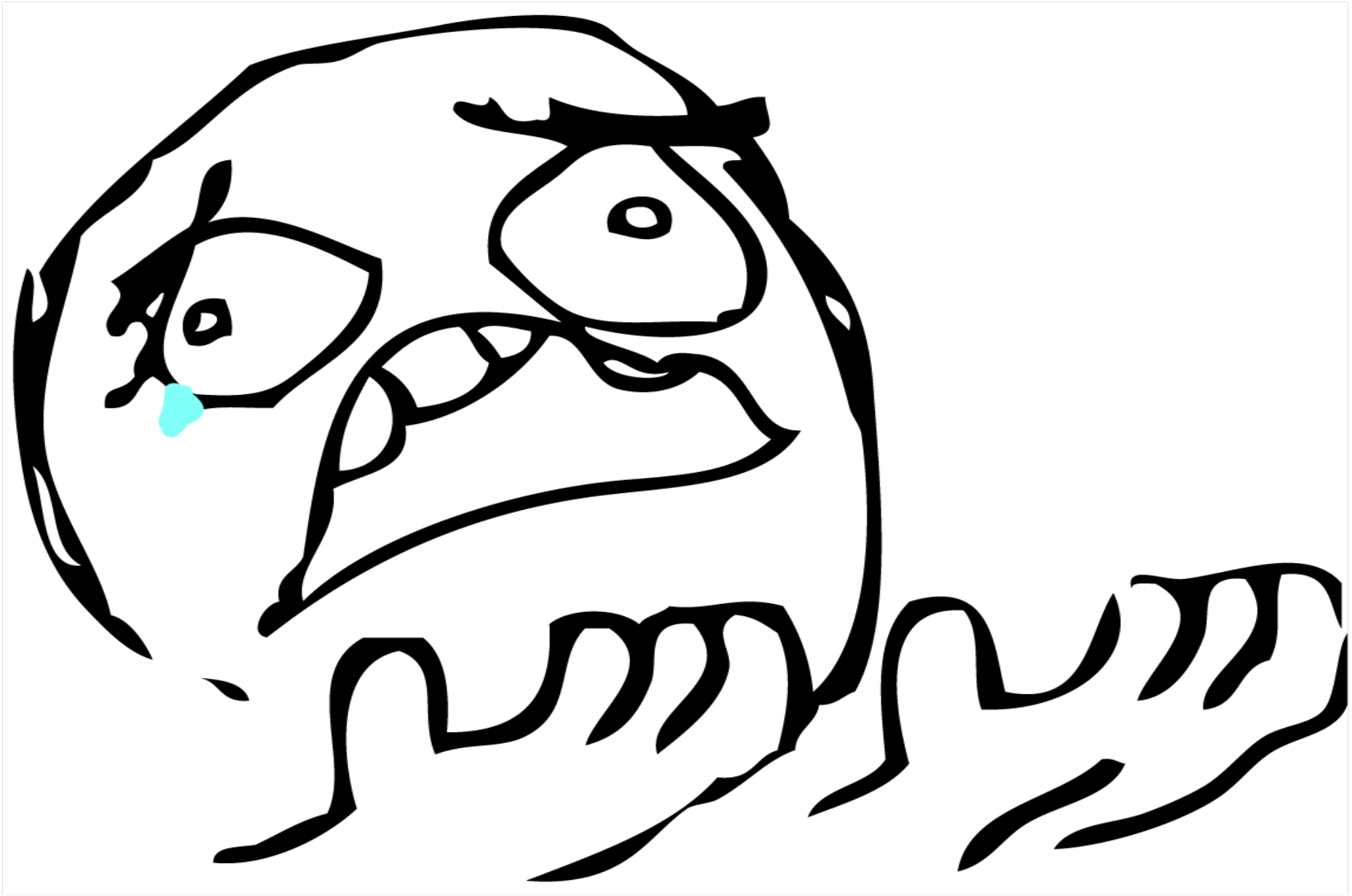
“Due to the lack of time, we could only conduct individual pages’ unit testing. Limited testing was done using use cases. Our team felt that this testing process was rushed and more time and effort should be allocated.”

“We failed completely to adhere to the initial [testing] plan. From the onset of the development process, we were more concerned with implementing the necessary features than the quality of our implementation, and as a result, we delayed, and eventually, failed to write any tests.”

Time estimates (in hours):

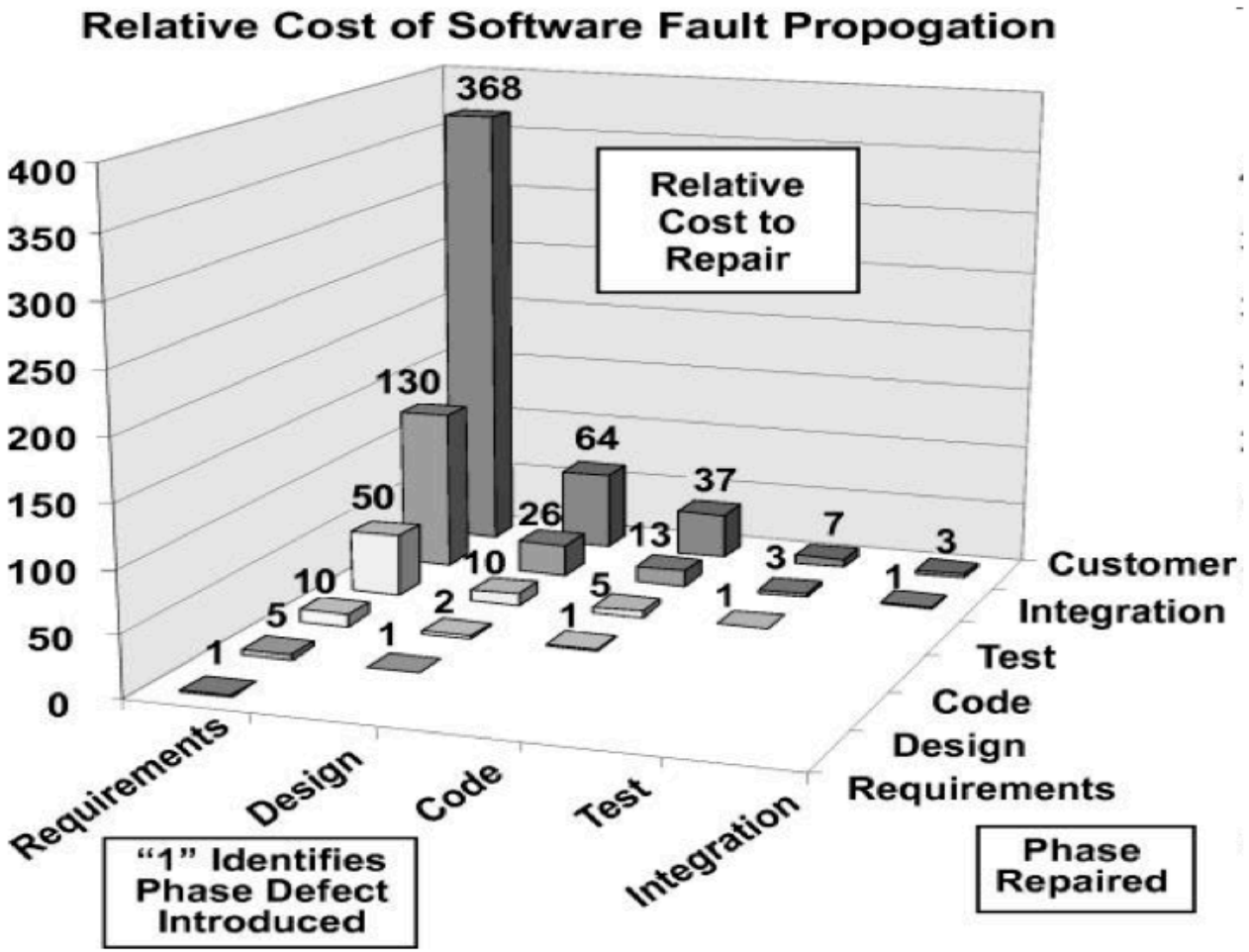
Activity	Estimated	Actual
testing plans	3	0
unit testing	3	1
validation testing	4	2
test data	1	1

“One portion we planned for but were not able to complete to our satisfaction was testing.”



QA is Important (Duh!)

Cost


Relative Cost of Software Fault Propogation
400
350
300
250
200
150
100
50
0
368
130
64
50
37
26
10
7
10
5
3
3
2
1
1
1
1
Relative Cost to Repair
Customer
Integration
Test
Code
Design
Requirements
Phase Repaired
Requirements
Design
Code
Test
Integration
"1" Identifies Phase Defect Introduced

Cost

theguardian

News | US | World | Sports | Comment | Culture | Business | Money | Environment | Science

News › Technology › Heartbleed

Heartbleed: developer who introduced the error regrets 'oversight'

Submitted just seconds before new year in 2012, the bug 'slipped through' – but discovery 'validates' open source

Alex Hern

Follow @alexhern Follow @guardiantech

theguardian.com, Friday 11 April 2014 03.05 EDT

Jump to comments (108)

Share 430

Tweet 269

+1 27

Share 103

Email

Technology

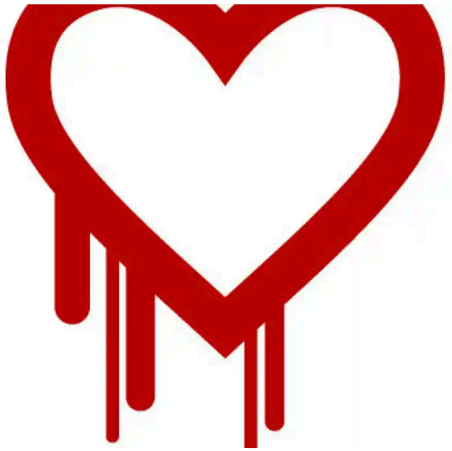
Heartbleed · Open source · Programming · Software · Internet · Hacking · Data and computer security

More news

More on this story

Heartbleed bug 'will cost millions'

Revoking all SSL certificates leaked by Heartbleed will cost millions of dollars, according to Cloudflare, which provides services to website hosts



▲ Image: Codenomicon

QA has many facets

How do you know that your Program works?

Questions

- How can we ensure that the specifications are correct?
- How can we ensure a system meets its specification?
- How can we ensure a system meets the needs of its users?
- How can we ensure a system does not behave badly?

Two kinds of analysis questions

- **Verification:** Does the system meet its specification?
  - i.e. did we build the system correctly?
- **Verification:** are there flaws in design or code?
  - i.e. are there incorrect design or implementation decisions?
- **Validation**: Does the system meet the needs of users?
  - i.e. did we build the right system?
- **Validation**: are there flaws in the specification?
  - i.e., did we do requirements capture incorrectly?

Software Errors

- Functional errors
- Performance errors
- Deadlock
- Race conditions
- Boundary errors
- Buffer overflow
- Integration errors
- Usability errors
- Robustness errors
- Load errors
- Design defects
- Versioning and configuration errors
- Hardware errors
- State management errors
- Metadata errors
- Error-handling errors
- User interface errors
- API usage errors
- …

Definition: software analysis

The systematic examination of a software artifact to determine its properties.

Definition: software analysis

The **systematic** examination of a software artifact to determine its properties.

- Attempting to be comprehensive, as measured by, as examples:

  Test coverage, inspection checklists, exhaustive model checking.

Definition: software analysis

The systematic **examination** of a software artifact to determine its properties.

- **Automated:** Regression testing, static analysis, dynamic analysis
- **Manual:** Manual testing, inspection, modeling

Definition: software analysis

The systematic examination of a **software artifact** to determine its properties.

- Code, system, module, execution trace, test case, design or requirements document.

Definition: software analysis

The systematic examination of a software artifact to determine its **properties**.

- **Functional:** code correctness
- **Non-functional:** evolvability, safety, maintainability, security, reliability, performance, …

VERY IMPORTANT

- *There is no one analysis technique that can perfectly address all quality concerns.*
- Which techniques are appropriate depends on many factors, such as the system in question (and its size/complexity), quality goals, available resources, safety/security requirements, etc etc…

Principle techniques

- **Dynamic:**
  - **Testing:** Direct execution of code on test data in a controlled environment.
  - **Analysis:** Tools extracting data from test runs.
- **Static:**
  - **Inspection:** Human evaluation of code, design documents (specs and models), modifications.
  - **Analysis:** Tools reasoning about the program without executing it.

	Error exists	No error exists
Error Reported	True positive (correct analysis result)	False positive
No Error Reported	False negative	True negative (correct analysis result)

Sound Analysis:

reports all defects
-> no false negatives
typically overapproximated

Complete Analysis:

every reported defect is an actual defect
-> no false positives
typically underapproximated

Classic Testing (Functional Correctness)

Testing

- Executing the program with selected inputs in a controlled environment (dynamic analysis)
- Goals:
  - Reveal bugs (main goal)
  - Assess quality (hard to quantify)
  - Clarify the specification, documentation
  - Verify contracts

"Testing shows the presence, not the absence of bugs

Edsger W. Dijkstra 1969

Specifications

- Textual
- Assertions
- Formal specifications

```
Algorithms.shortestDistance(g, "Tom", "Anne");

> ArrayOutOfBoundsException
```

```
Algorithms.shortestDistance(g, "Tom", "Anne");

> -1
```

```
class Algorithms {
    /**
     * This method finds the shortest distance between to
     * verticies. It returns -1 if the two nodes are not
     * connected.
    */
    int shortestDistance(…) {…}
}
```

```
class Algorithms {
    /**
     * This method finds the shortest distance between to
     * verticies. Method is only supported
     * for connected verticies.
    */
    int shortestDistance(…) {…}
}
```

```
/*@ requires amount >= 0;
    ensures balance == \old(balance)-amount &&
            \result == balance;
@*/
public int debit(int amount) {
    ...
}
```

- JML (Java modeling language specification)

```
/**
  * Calls the <code>read(byte[], int, int)</code> overloaded [..
  * @param buf The buffer to read bytes into
  * @return The value retured from <code>in.read(byte[], int, in
  * @exception IOException If an error occurs
  */
public int read(byte[] buf) throws IOException
{
    return read(buf, 0, buf.length);
}
```

- Textual specification with JavaDoc

Benefits of Specification

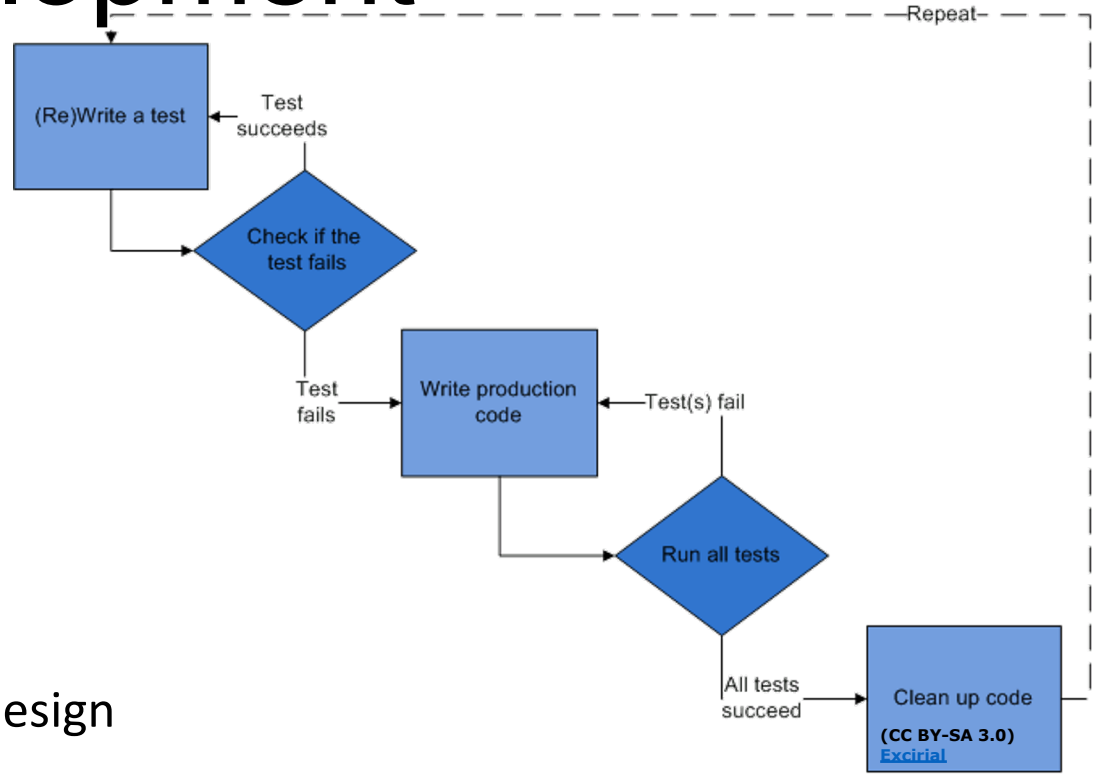
- Exact specification of what should be implemented
- Decompose a system into its parts, develop and test parts independently
- Accurate blame assignments and identification of buggy behavior
- Useful for test generation and as test oracle

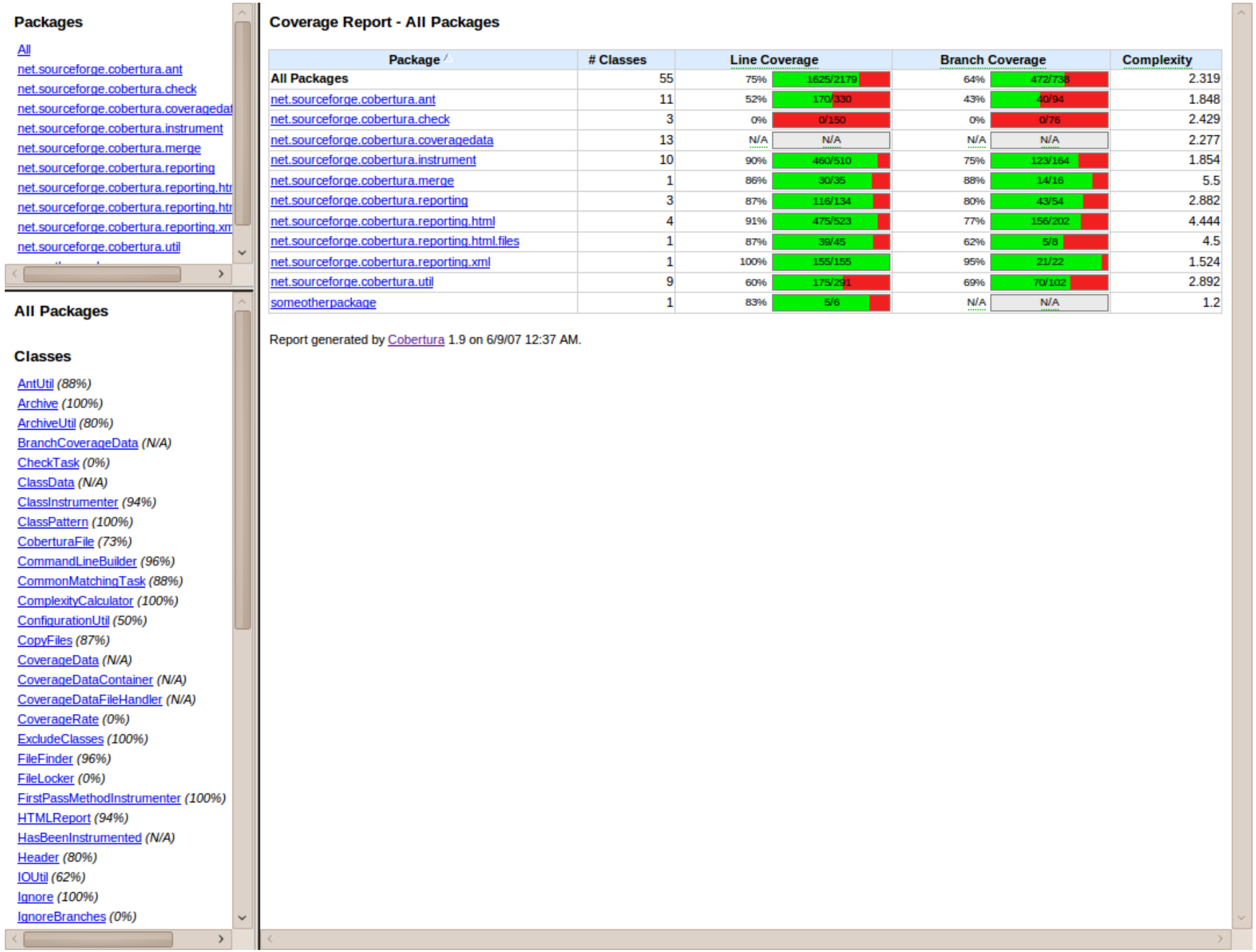
Testing Levels

- Unit testing
- Integration testing
- System testing

Test Driven Development

- Tests first!
- Popular agile technique
- Write tests as specifications before code
- Never write code without a failing test
- Claims:
  - Design approach toward testable design
  - Think about interfaces first
  - Avoid writing unneeded code
  - Higher product quality (e.g. better code, less defects)
  - Higher test suite quality
  - Higher overall productivity

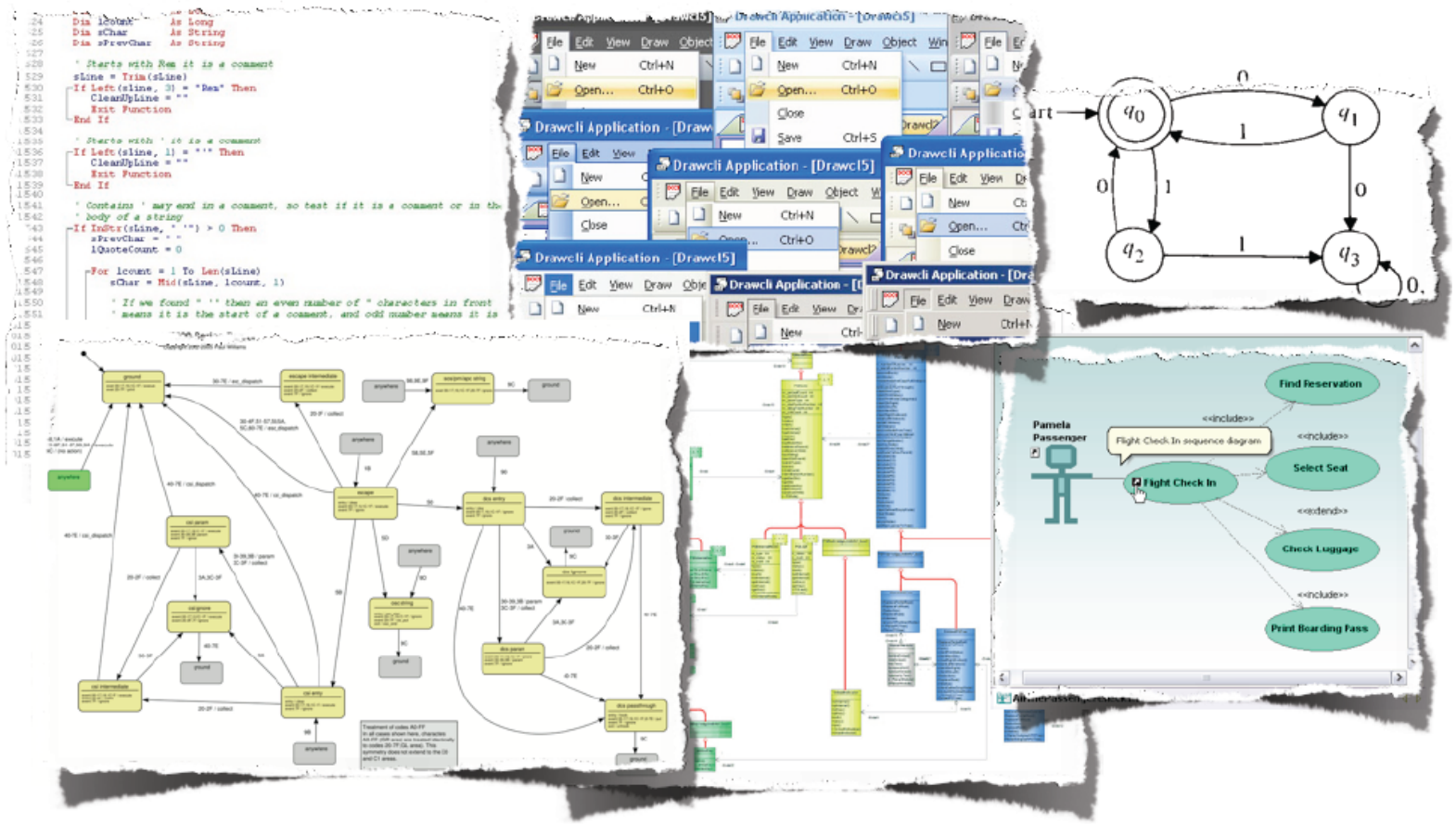



Packages
All
net.sourceforge.cobertura.ant
net.sourceforge.cobertura.check
net.sourceforge.cobertura.coveragedat
net.sourceforge.cobertura.instrument
net.sourceforge.cobertura.merge
net.sourceforge.cobertura.reporting
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.xm
net.sourceforge.cobertura.util
All Packages
Classes
AntUtil (88%)
Archive (100%)
ArchiveUtil (80%)
BranchCoverageData (N/A)
CheckTask (0%)
ClassData (N/A)
ClassInstrumenter (94%)
ClassPattern (100%)
CoberturaFile (73%)
CommandLineBuilder (96%)
CommonMatchingTask (88%)
ComplexityCalculator (100%)
ConfigurationUtil (50%)
CopyFiles (87%)
CoverageData (N/A)
CoverageDataContainer (N/A)
CoverageDataFileHandler (N/A)
CoverageRate (0%)
ExcludeClasses (100%)
FileFinder (96%)
FileLocker (0%)
FirstPassMethodInstrumenter (100%)
HTMLReport (94%)
HasBeenInstrumented (N/A)
Header (80%)
IOUtil (62%)
Ignore (100%)
IgnoreBranches (0%)
Coverage Report - All Packages
Package
Classes
Line Coverage
Branch Coverage
Complexity
All Packages 55 75% 1625/2179 64% 472/738 2.319
net.sourceforge.cobertura.ant 11 52% 170/330 43% 40/94 1.848
net.sourceforge.cobertura.check 3 0% 0/150 0% 0/76 2.429
net.sourceforge.cobertura.coveragedata 13 N/A N/A N/A N/A 2.277
net.sourceforge.cobertura.instrument 10 90% 460/510 75% 123/164 1.854
net.sourceforge.cobertura.merge 1 86% 30/35 88% 14/16 5.5
net.sourceforge.cobertura.reporting 3 87% 116/134 80% 43/54 2.882
net.sourceforge.cobertura.reporting.html 4 91% 475/523 77% 156/202 4.444
net.sourceforge.cobertura.reporting.html.files 1 87% 39/45 62% 5/8 4.5
net.sourceforge.cobertura.reporting.xml 1 100% 155/155 95% 21/22 1.524
net.sourceforge.cobertura.util 9 60% 175/291 69% 70/102 2.892
someotherpackage 1 83% 5/6 N/A N/A 1.2
Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

“Traditional” coverage

- Statement: Has each statement in the program been executed?
- Branch: Has each of each control structure been executed?
- Function: Has each function in the program been called?
- Path: requires that all paths through the Control Flow Graph are covered.
- …

We can measure coverage on almost anything



A. Zeller, Testing and Debugging Advanced course, 2010

We can measure coverage on almost anything

- Common adequacy criteria for testing approximate full “coverage” of the program execution or specification space.
- Measures the extent to which a given verification activity has achieved its objectives; approximates adequacy of the activity.
  - *Can be applied to any verification activity, although most frequently applied to testing.*
- Expressed as a ratio of the measured items executed or evaluated at least once to the total number of measured items; usually expressed as a percentage.

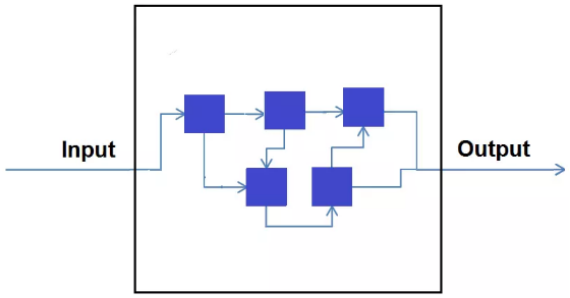
What is testing?

- *Direct execution of code on test data in a controlled environment*
- Principle goals:
  - Validation: program meets requirements, including quality attributes.
- Other goals:
  - Clarify specification: Testing can demonstrate inconsistency; either spec or program could be wrong
  - Learn about program: How does it behave under various conditions? Feedback to rest of team goes beyond bugs
  - Verify contract, including customer, legal, standards

What are we covering?

- Program/system functionality:
  - Execution space (white box!).
  - Input or requirements space (black box!).
- The expected user experience (usability).
  - GUI testing, A/B testing
- The expected performance envelope (performance, reliability, robustness, integration).
  - Security, robustness, fuzz, and infrastructure testing.
  - Performance and reliability: soak and stress testing.
  - Integration and reliability: API/protocol testing

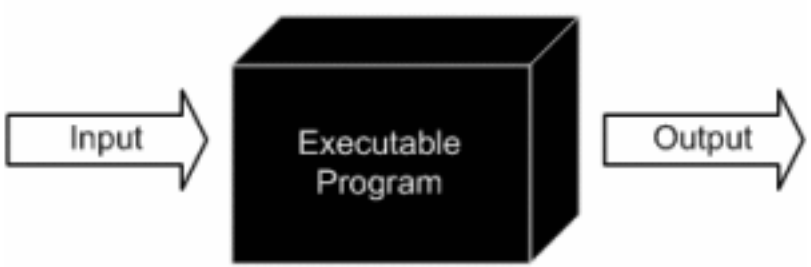
White box testing



Tests internal structures or workings of an application, as opposed to its functionality.

- Unit Test
- Testing for Memory Leaks
- Penetration Testing
  - ***"What would a cybercriminal do to harm my organization' computer systems, applications, and network?"***

Black box testing



- Functionality of application is tested without looking at the implementation details
- Types
  - **Functional Testing**
    - Smoke Testing
    - Regression Testing
    - **…**
  - **Non-Functional Testing**
    - Performance Testing
    - Compatibility Testing
    - Stress Testing

Regression testing

- Ensure that a small change in one part of the system does not break existing functionality elsewhere in the system.

Regression testing

- Ensure that a small change in one part of the system does not break existing functionality elsewhere in the system.
- Application scenario:
  - When new functionalities are added
  - In case of change requirements
  - When there is a defect fix
  - When there are performance issues
  - In case of environment changes
  - When there is a patch fix

Regression testing

- Unit Test
- Progressive Test
- Selective Test
- Retest-All Testing
- Complete Testing

What are we covering?

- Program/system functionality:
  - Execution space (white box!).
  - Input or requirements space (black box!).
- The expected user experience (usability).
  - GUI testing, A/B testing
- The expected performance envelope (performance, reliability, robustness, integration).
  - Security, robustness, fuzz, and infrastructure testing.
  - Performance and reliability: soak and stress testing.
  - Integration and reliability: API/protocol testing

Manual Testing?

GENERIC TEST CASE: USER SENDS MMS WITH PICTURE ATTACHED.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select "Create new Message"	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select "Insert Picture"	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select "Send Message"	Message is correctly sent

- Live System?
- Extra Testing System?
- Check output / assertions?
- Effort, Costs?
- Reproducible?

Automating GUI/Web Testing

- First: why is this hard?
- Capture and Replay Strategy
  - mouse actions
  - system events
- Test Scripts: (click on button labeled "Start" expect value X in field Y)
- Lots of tools and frameworks
  - e.g. JUnit + Jemmy for Java/Swing
- (Avoid load on GUI testing by separating model from GUI)

Usability: A/B testing

- Controlled randomized experiment with two variants, A and B, which are the control and treatment.
- One group of users given A (current system); another random group presented with B; outcomes compared.
- Often used in web or GUI-based applications, especially to test advertising or GUI element placement or design decisions.

Example

- A company sends an advertising email to its customer database, varying the photograph used in the ad...

Example: group A (99% of users)

- Act now! Sale ends soon!

Example: group B (1%)

- Act now! Sale ends soon!

Usability: A/B testing

- However, it cannot..
  - Tell you why
  - Let you test drastic redesigns of your website or app.
  - Tell you if you’re solving the right/wrong problem.

What are we covering?

- Program/system functionality:
  - Execution space (white box!).
  - Input or requirements space (black box!).
- The expected user experience (usability).
  - GUI testing, A/B testing
- The expected performance envelope (performance, reliability, robustness, integration).
  - Security, robustness, fuzz, and infrastructure testing.
  - Performance and reliability: soak and stress testing.
  - Integration and reliability: API/protocol testing

Performance Testing

- Specification? Oracle?
- Test harness? Environment?
- Nondeterminism?
- Unit testing?
- Automation?
- Coverage?

Unit and regression testing for performance

- Measure execution time of critical components
- Log execution times and compare over time