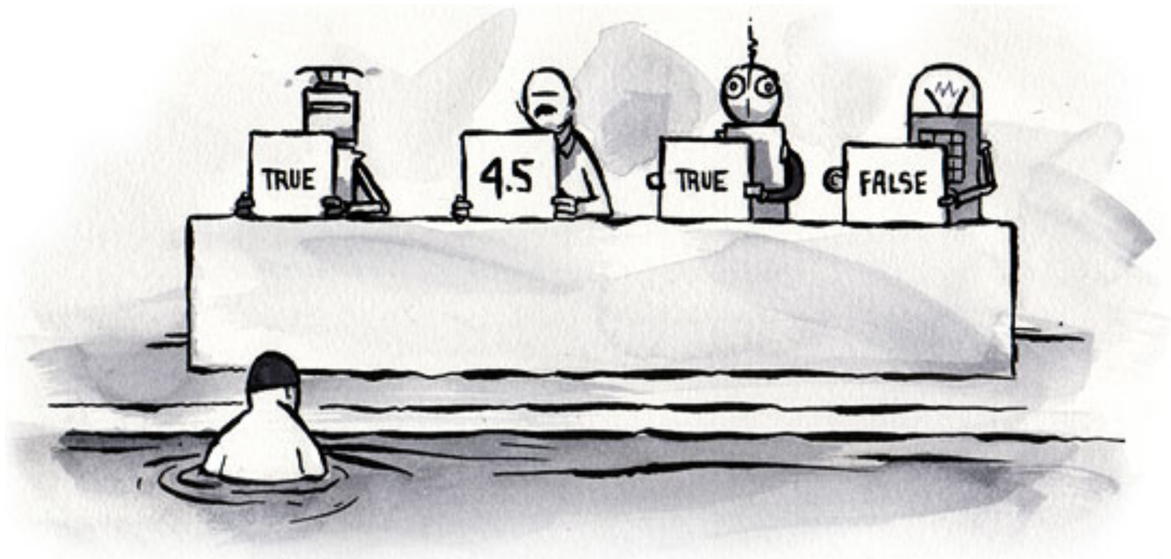


Software Testing for Continuous Delivery

Seminar 3: Some Terminology & Testability

Dr. Byron J. Williams

August 26, 2019

UF Herbert Wertheim
College of Engineering
*Department of Computer & Information
Science & Engineering*
UNIVERSITY *of* FLORIDA

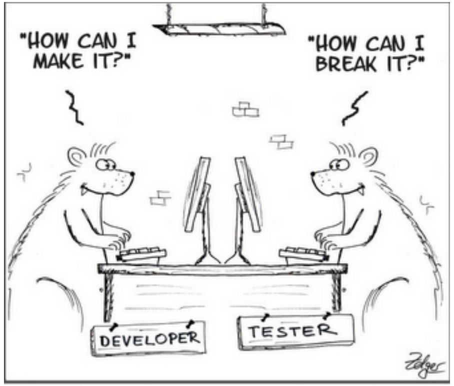
FICS Research

Content adapted from “Developer Testing” by Alexander Tarlinder 1st Edition (2016)

Testing / QA / DevOps Terms

Smoke testing	Defect	Mutation testing
Code review	Load (stability) tests	Performance testing
Software quality	Usability testing	Defect density
Penetration testing	Software inspection	Beta testing
Test runner	User interface testing	System testing
Integration testing	Exploratory testing	Error
Operational profile	Defect containment	Quality Engineering
Unit testing	Partition testing	Fault
Regression testing	Boundary value analysis	Quality Assurance
Fuzz testing	Validation	Failure
Equivalence class	Acceptance testing	Severity levels (defects)
Verification	Gray box testing	Continuous integration
Continuous delivery	Automation	Correctness

RECAP



They are not so much different,
but they have different path for the same goal,
to improve quality!!

Testing Perspectives

"developer" **-** person whose primary responsibility is to write source code - the output of the developers should be working software, not just something that compiles

Developer Testing (developer mind-set)

taking ownership of the quality of the produced code, instead of expecting that someone else will test it

Developer testing is an umbrella term for all test-related activities a developer engages in

To build - **Testing to Support**

Software Testing (tester mind-set)

investigating how the product might fail

To break - **Testing to Critique**

RECAP

Points to Remember

- Quality cannot be tested in, it has to be built in
- Tests and testers are needed because developers suffer from author bias i.e., the inability to see faults in one’s own creation
- All written code ***should*** have tests - the opposite: code that turns all attempts to change it into a mixture of one part guessing game and one part nightmare— **legacy code**
  - legacy code is code without tests

RECAP

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

Terminology

Terminology

- **Important**: you agree on the terminology in your organization (SW Testing for CD)

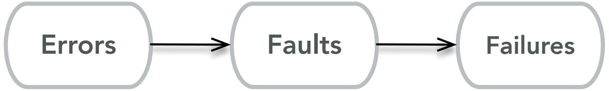
- What is a *module*? What is a software *component*? How do you define a unit (i.e., unit testing)? Smoke test vs Sanity test? What is an integration test?

Defects / Bugs

- **Failure**: external behavior
  - deviation from expected behavior
  - something goes wrong at execution
  - e.g., student cannot enroll in a course even if nobody is currently enrolled
- **Fault**: internal characteristics
  - cause for failures
  - a mistake written down in code and/or document
  - e.g., if(current_enrol=max_enrol) {// cannot enroll any more}
  - SHOULD BE if(current_enrol==max_enrol) {//cannot enroll any more}

- **Error**: incorrect/missing human action
  - conceptual mistakes
  - human misunderstanding
  - e.g., when the class is full, student can still enroll if the instructor permits
- **Bug**: abstract way of describing the above - generally a problematic term; avoid

- Relationship:

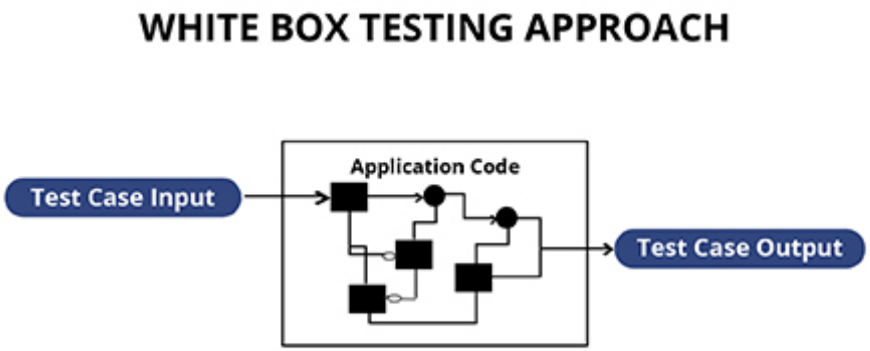


Defect

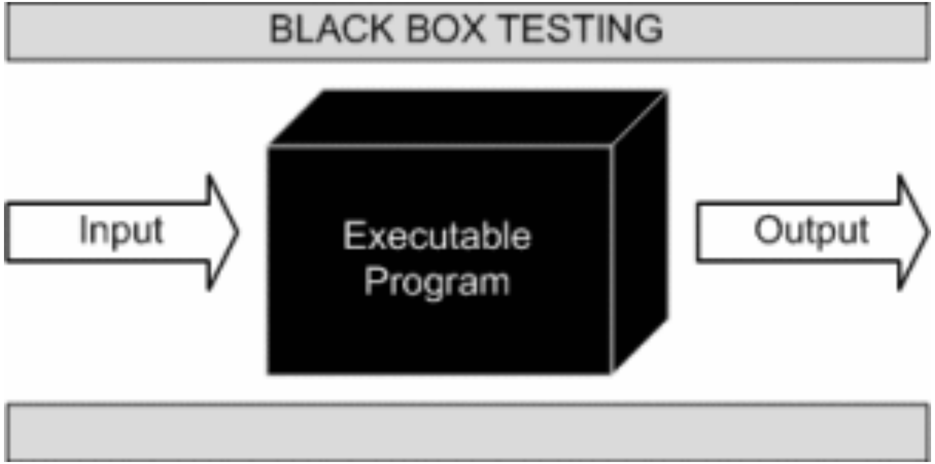
***Error*, *Fault*, or *Failure* in the system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways**

(think specifications / stories / expectations)

White Box vs. Black Box Testing

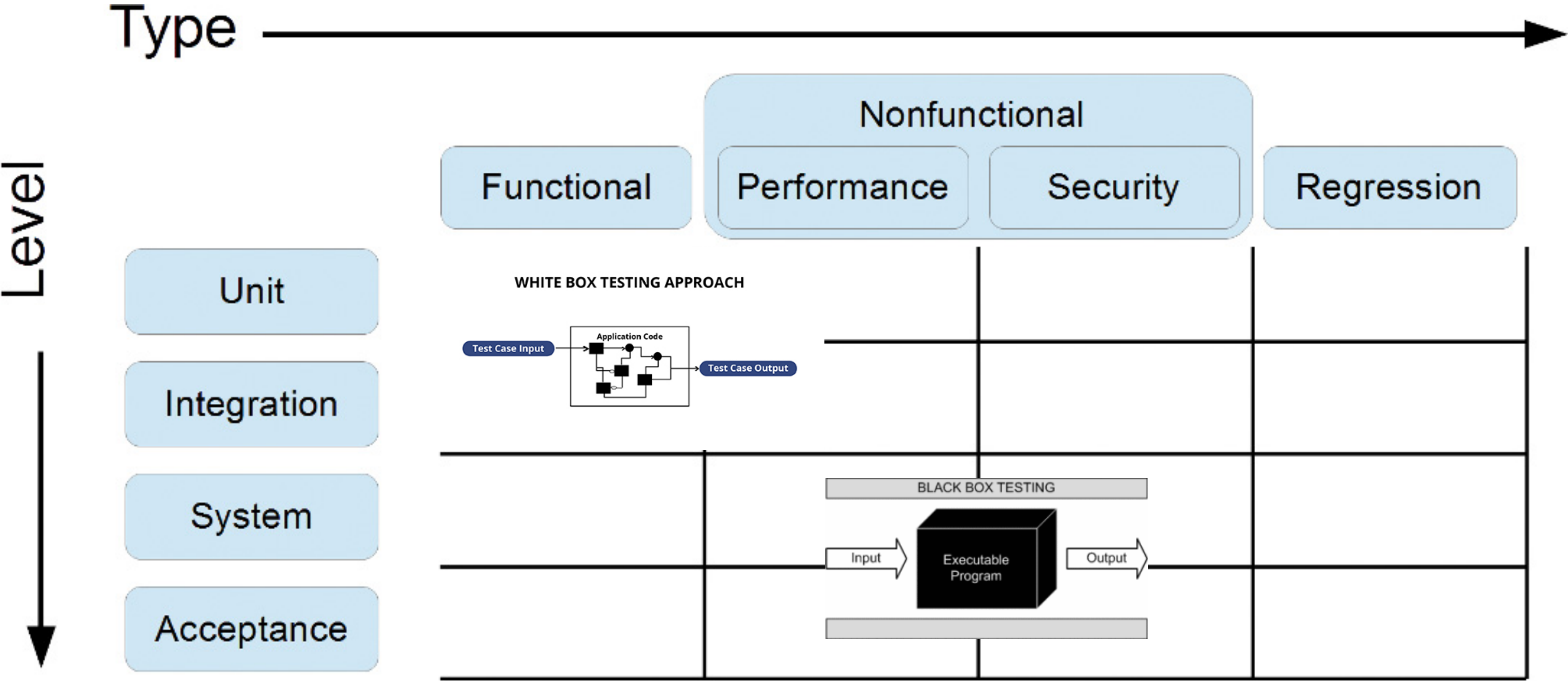


https://www.invensis.net/blog/it/white-box-software-testing-advantages-disadvantages/



http://softwaretestingfundamentals.com/black-box-testing/

Testing Levels



Type

Level

Nonfunctional

Functional

Performance

Security

Regression

Unit

Integration

System

Acceptance

Agile Testing Quadrants

- **Business**
  If a customer uses direct bank payments to pay for our product and pays too much, does he or she get a refund, or is the excess amount stored and used in the next transaction?

- **Tech**
  If validation of the credit card fails, the transaction enclosing the purchase is rolled back, nothing is stored in the database, and the event is logged.

Business facing

Guide development	Examples A/B tests Story tests (written first) UX (user experience) testing Prototypes Simulations Q2	Exploratory testing Workflows System integration (business oriented) Usability testing User acceptance testing Q3	Critique the product
	Q1 Unit tests Component tests (code level) Testing connectivity	Q4 Performance testing Load testing Security testing Quality attributes (… ilities)	

Technology facing

UF Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

Testability

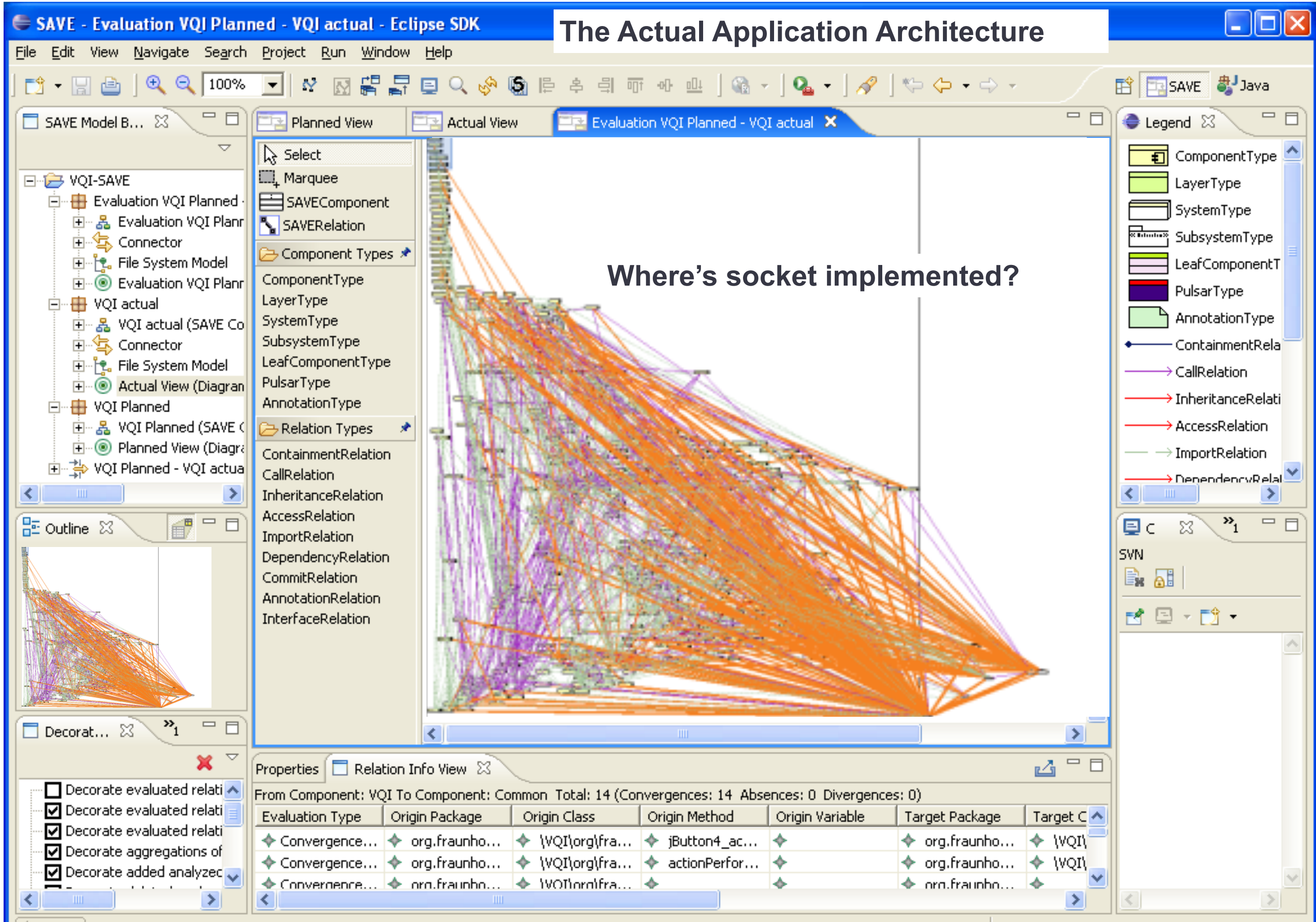
Testability

- The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
  - i.e., “program elements” - can be put into a known state, acted on, then observed
- Plainly speaking – **how hard it is to find faults in the software**
- Testability is dominated by two practical problems
  - How to provide the test values to the software
  - How to observe the results of test execution

Andy Glover cartoontester.blogspot.com © 2012

Testability Challenges

- Spaghetti code
- convoluted configs
- long start-up time
- difficulty in producing certain states
- e.g., required to login to UI, perform steps navigating multiple views before reaching the functionality needed



The Actual Application Architecture

Where's socket implemented?

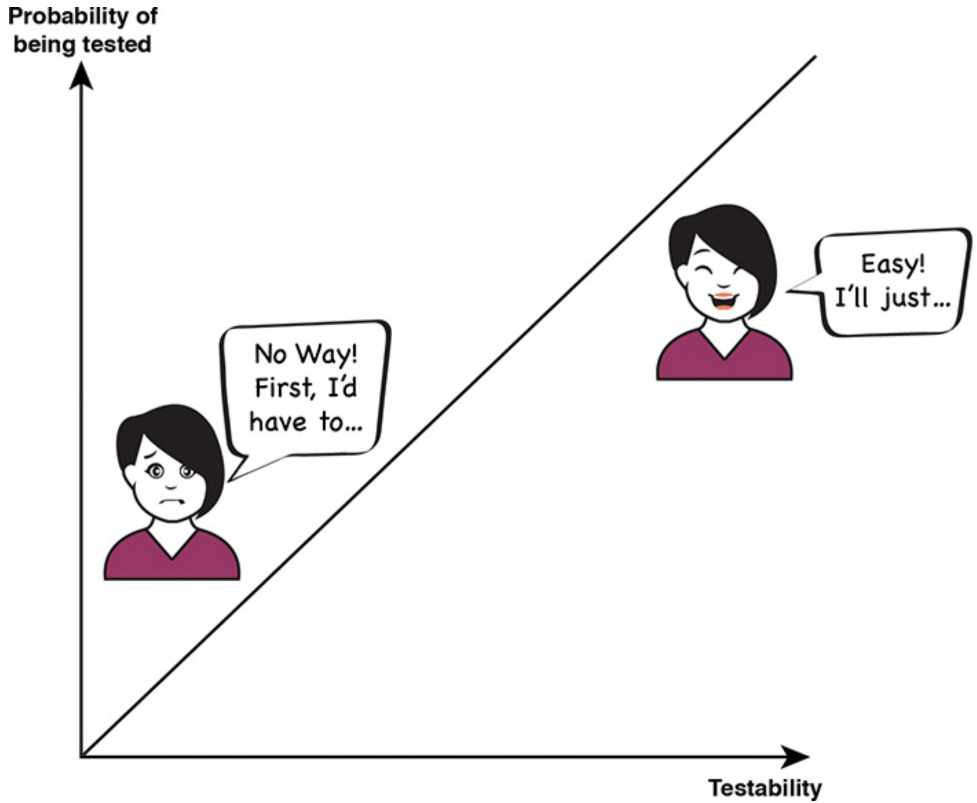
From Component: VQI To Component: Common Total: 14 (Convergences: 14 Absences: 0 Divergences: 0)

Evaluation Type	Origin Package	Origin Class	Origin Method	Origin Variable	Target Package	Target C
Convergence...	org.fraunho...	\VQI\org\fra...	jButton4_ac...		org.fraunho...	\VQI\
Convergence...	org.fraunho...	\VQI\org\fra...	actionPerfor...		org.fraunho...	\VQI\
Convergence...	org.fraunho...	\VQI\org\fra...			org.fraunho...	

M. Lindvall et. al,, Architecture Analysis of Evolving Complex Systems of Systems, Technical Presentation - Software Assurance Symposium, 2008

Testable Software

- testability is linked to our prior experience of the things we want to test and our tolerance for defects - **behavioral / human aspect**
- **the code is designed with testability in mind from the start and each program element has a single area of responsibility**
- The more testable the software, the greater the chance that somebody will test it, that is, verify that it behaves correctly with respect to a specification or some other expectations
- **Based on:** How well do we know the product and the technology used to build it? How good are our testing skills? What's our testing strategy?

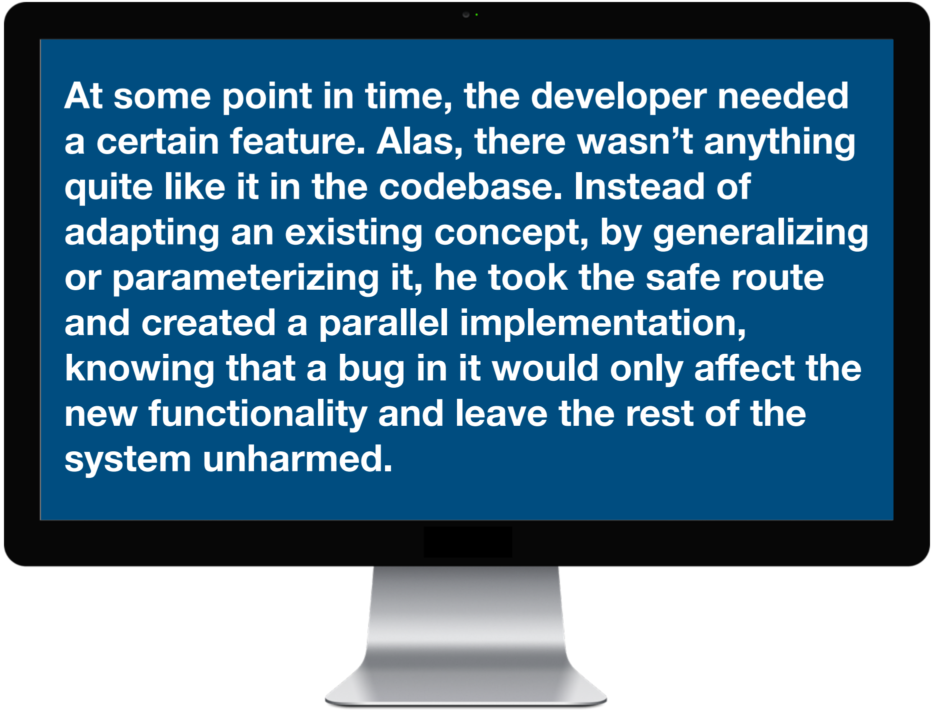


Is untestable software going to be tested?

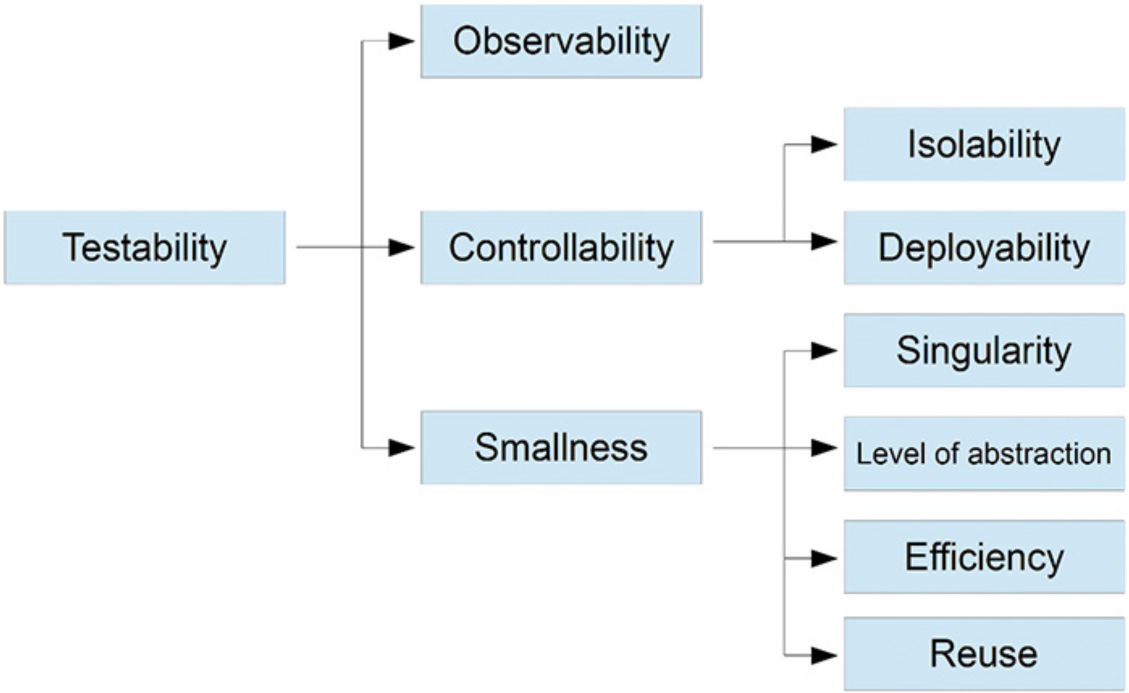
people (generally) follow the path of least resistance in their work, and if testing isn't along that path, it's very likely not going to be performed

Testability Benefits

- If the software is developed so that its behavior can be verified, it's easy to confirm that it supports a certain feature
- Fewer Surprises - e.g., 95% done with new feature X…how do you know you didn't break anything?
- More easily changed (safety & cost) - Fear results in duplication (DRY)



At some point in time, the developer needed a certain feature. Alas, there wasn't anything quite like it in the codebase. Instead of adapting an existing concept, by generalizing or parameterizing it, he took the safe route and created a parallel implementation, knowing that a bug in it would only affect the new functionality and leave the rest of the system unharmed.

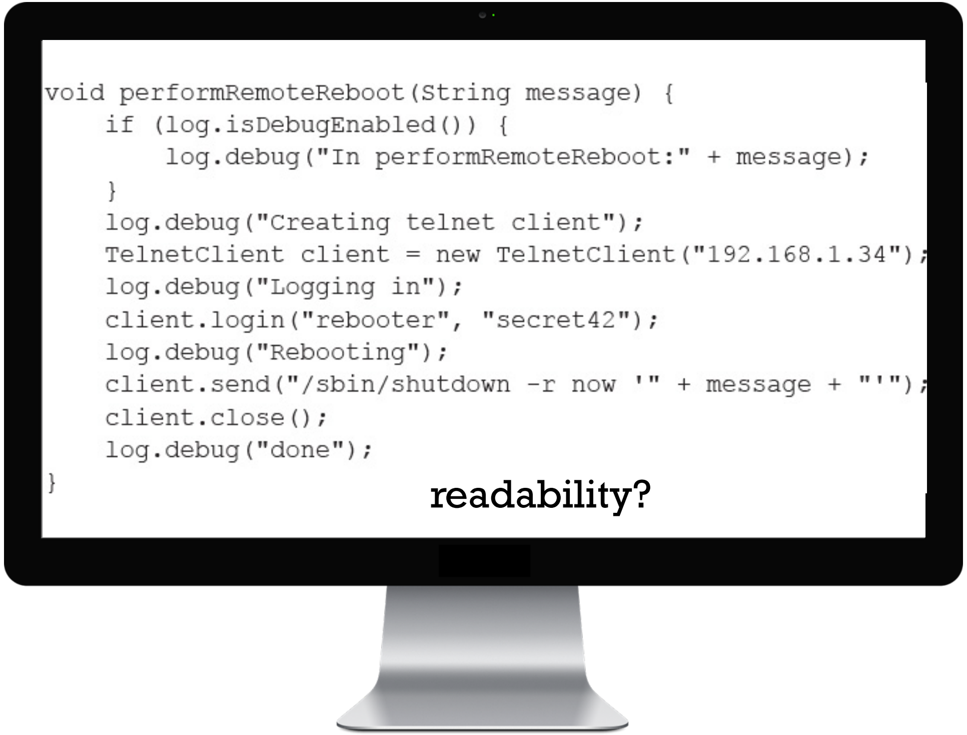


Testability Quality Decomposed

NOTE: When a ***program element*** is testable, it means that it can be put in a **known state**, **acted on,** and then **observed.** Further, it means that this can be done **without affecting any other program elements** and **without them interfering**

Observability

- The best test in the world isn't worth anything unless its effects can be seen (observing output is obvious)
- Other output not meant for end-users
  - e.g., logs, temp files, diagnostic info - program intrusions
  - Achievable only by developers (e.g., debugging)



```
void performRemoteReboot(String message) {
    if (log.isDebugEnabled()) {
        log.debug("In performRemoteReboot:" + message);
    }
    log.debug("Creating telnet client");
    TelnetClient client = new TelnetClient("192.168.1.34");
    log.debug("Logging in");
    client.login("rebooter", "secret42");
    log.debug("Rebooting");
    client.send("/sbin/shutdown -r now '" + message + "'");
    client.close();
    log.debug("done");
}
```

readability?

Observability & Encapsulation

- Observability and information hiding are often at odds with each other
- Although all of this is true, the root cause of the problem isn't really information hiding or encapsulation, but poor design and implementation, which, in turn, forces us to ask the question of the decade:
- **Should I test private methods?**
- Two Options

1. open up the encapsulation by relaxing restrictions on accessibility to increase both observability and controllability (e.g., package scoping)

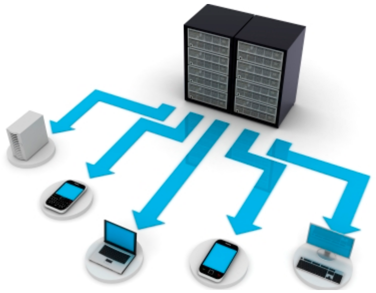
2. at a level where we need to worry about the observability of deeply buried monolithic spaghetti isn't the course of action that gives the **best bang for the buck** - focus only on system or integration tests
(i.e., white box your own code & black box all others)

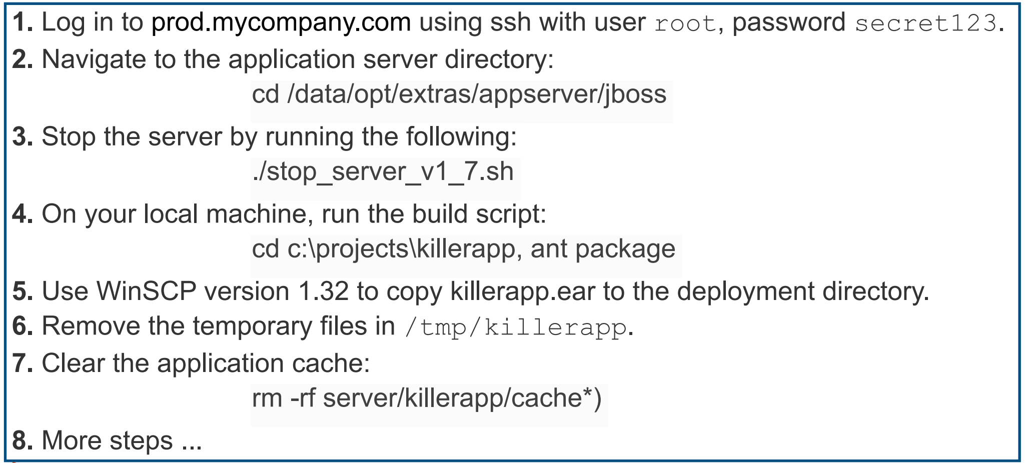
Controllability

- The ability to put something in a specific **state**
  - we like to deal with determinism
- Is of paramount importance to any kind of testing because it leads to ***reproducibility (?)***
  - When we get a bug report, we want to be able to **reproduce the bug** so that we may understand under what conditions it occurs. Given that understanding, we can fix it. The ability to reproduce a given condition in a system, component, or class (program element) depends on the ability to isolate it and manipulate its **internal state**

Deployability

- is a measure of the amount of work needed to deploy the system, most notably, into production
- Deployability affects the developers' ability to run their code in a production-like environment.





1. Log in to prod.mycompany.com using ssh with user `root`, password `secret123`.

2. Navigate to the application server directory:

```
cd /data/opt/extras/appserver/jboss
```

3. Stop the server by running the following:

```
./stop_server_v1_7.sh
```

4. On your local machine, run the build script:

```
cd c:\projects\killerapp, ant package
```

5. Use WinSCP version 1.32 to copy killerapp.ear to the deployment directory.

6. Remove the temporary files in `/tmp/killerapp`.

7. Clear the application cache:

```
rm -rf server/killerapp/cache*)
```

8. More steps ...

> "How long does it take to get a change that affects one line of code into production?"

Deployability

- The bottom line is that developers are not to consider themselves **finished** with their code until **they've executed** it in an environment that resembles the actual **production environment**

HEROKU

docker

Google Cloud Platform

aws

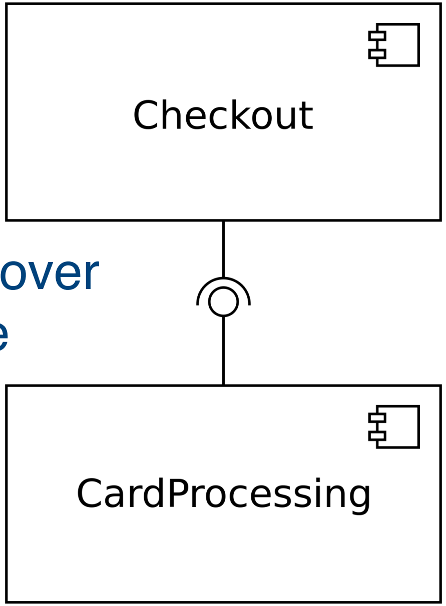
"It works on my machine."

Isolability (modularity, low-coupling)

- Being able to isolate the program element under test—be it a function, class, web service, or an entire system
- Modular
  - related concepts are grouped together, and changes don’t ripple across the entire system
- Components and other *program elements* with lots of **dependencies** are not only difficult to modify, but also **difficult to test**

Smallness

- General rule: The smaller the software, the better the testability
  - less to test - fewer moving parts that need to be controlled and observed
  - primarily translates into the quantity of tests needed to cover the software to achieve a sufficient degree of confidence
- # of features & size (LOC, # methods, # classes, # functions)
  - They both drive different aspects of testing (black box & white box)



Smallness

- developer perspective

Singularity:

If something is singular, there's only one instance of it. In systems with high singularity, every behavior and piece of data have a single source of truth

Efficiency:

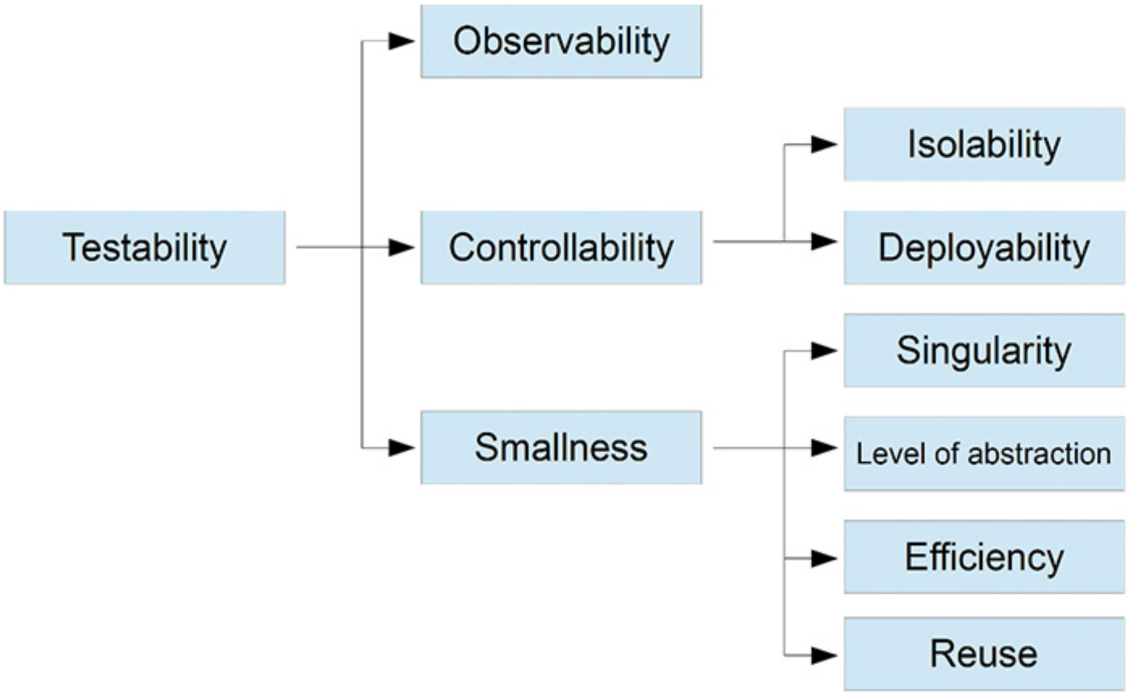
Equals the ability to express intent in the programming language in an idiomatic way and making use of that language's functionality to keep the code expressive and concise

Level of Abstraction:

Determined by the choice of programming language and frameworks. If they do the majority of the heavy lifting, the code can get both smaller and simpler. Extremes = assembly language —> high-level language (backed by a few frameworks)

Reuse:

Refers to making use of third-party components to avoid reinventing the wheel. Reduces the need for developer tests, because the functionality isn't owned by them and doesn't need to be tested



Testability Quality Decomposed

NOTE: When a ***program element*** is testable, it means that it can be put in a **known state**, **acted on,** and then **observed.** Further, it means that this can be done **without affecting any other program elements** and **without them interfering**