

# **CSE 4283 / 6283**

## **Software Testing and QA**

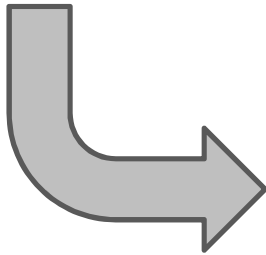
**Dr. Tanmay Bhowmik**  
**tbhowmik@cse.msstate.edu**

Special thanks to Dr. Nan Niu & Dr. Byron Williams

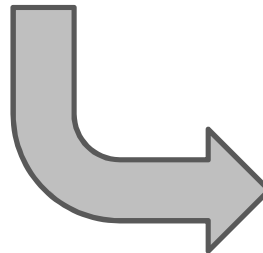


# Agenda

**Last Topic:**  
RBT



**This Topic:**  
Testability



**Next Topic:**  
Unit Testing



# To Do



- Quiz-1
  - Feb 8, online, during the first 10 mins of the class
  - Covers up to the class of Feb 6

Picture source: Internet

3



# Testing Perspectives

- **Developer** – a person whose primary responsibility is to write source code
  - the output from 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**

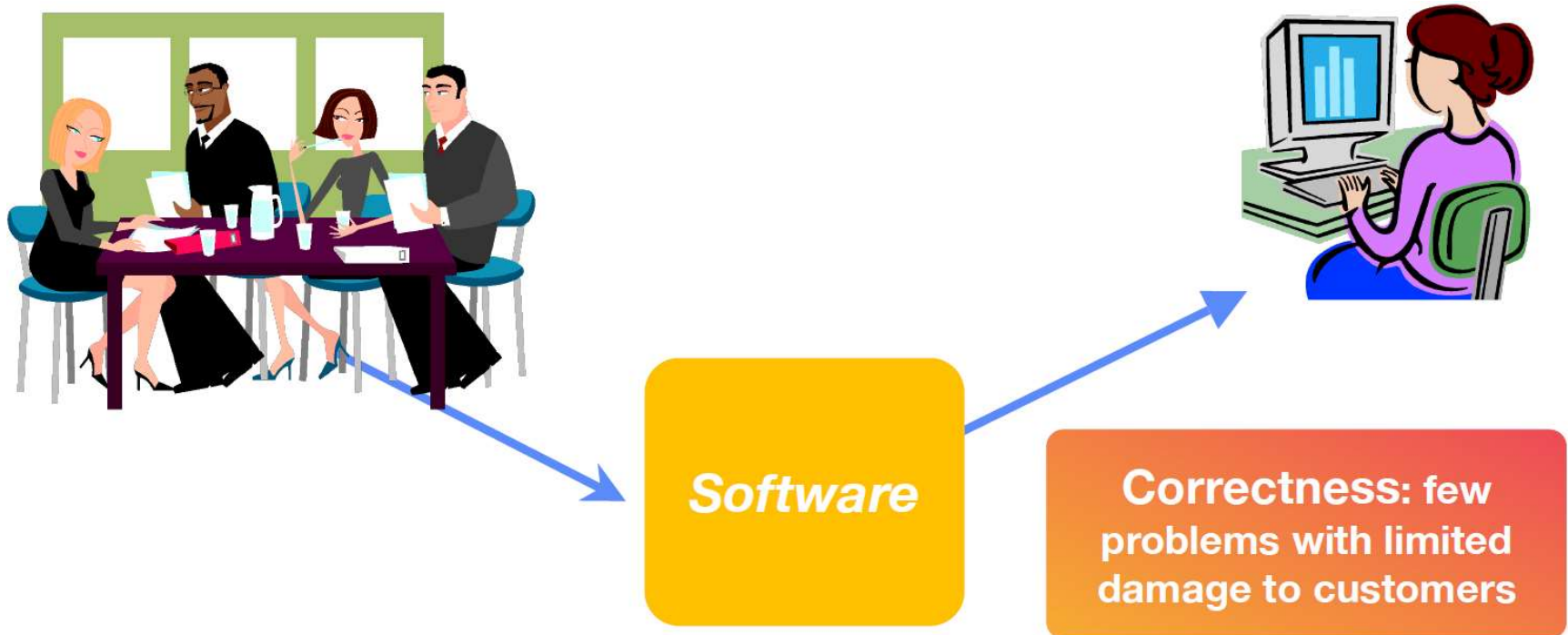
Picture source: Internet

4



# Quality Assurance

- To ensure that few, if any, defects remain in the software when it is delivered to its customers or released to the market



Picture source: Internet

5



# Testing (operational) Definition

- Defined as: the **execution of software and the observation of its behavior** or outcome demonstrated using controlled experiments
- **Purpose:**
  - To demonstrate quality / proper behavior
  - To detect problems that need to be fixed
- General Process Steps:
  1. Test Planning & Prep
  2. Test Execution
  3. Analysis & Follow-up

## Another Definition:

“Testing is the process of evaluating a product by learning about it through exploration and experimentation, which includes to some degree: questioning, study, modeling, observation, inference, etc.”

(Bach 2013).



# Defects / Bugs

- **Error**: A mistake made by a programmer or software engineer which caused the fault, which in turn may cause a failure
  - conceptual mistakes
  - human misunderstanding
    - e.g., this is a Linux App. So it is going to be more secure anyway



Picture source: Internet

7



# Defects / Bugs

- **Fault**: Condition / internal characteristic that *may* cause a failure in the system
  - a mistake written down in code and/or document
  - e.g., `if(current_enroll = max_enrol) { //cannot enroll any more }`
  - **SHOULD BE** `if(current_enroll == max_enrol) { //cannot enroll any more }`



Picture source: Internet

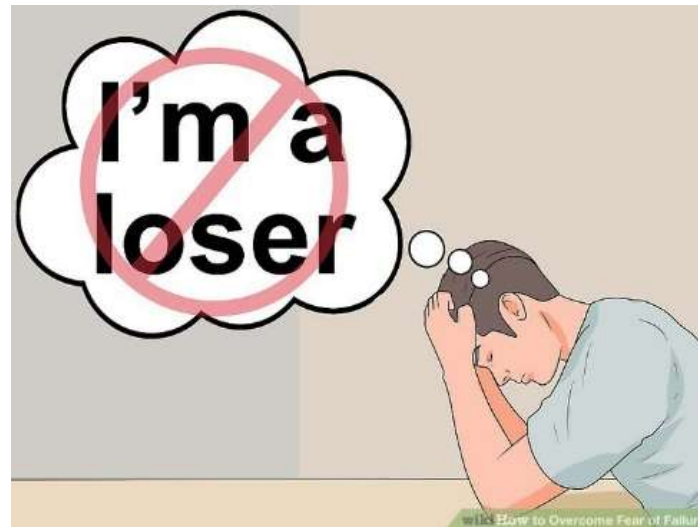
8





# Defects / Bugs

- **Failure**: Inability of the system to perform a function according to its specification due to some fault
  - deviation from expected behavior
  - something goes wrong at execution
    - e.g., student cannot enroll in a course even if nobody is currently enrolled
    - Can not withdraw money despite sufficient balance



Picture source: Internet

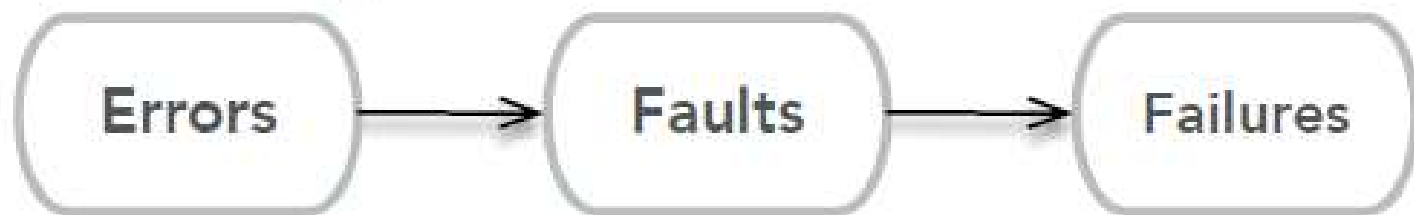
9



# Defects / Bugs

- **Bug**: An abstract way of describing the above - problematic terms; avoid

Relationship:



Picture source: Internet

10



# Defects

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)



# 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
- 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



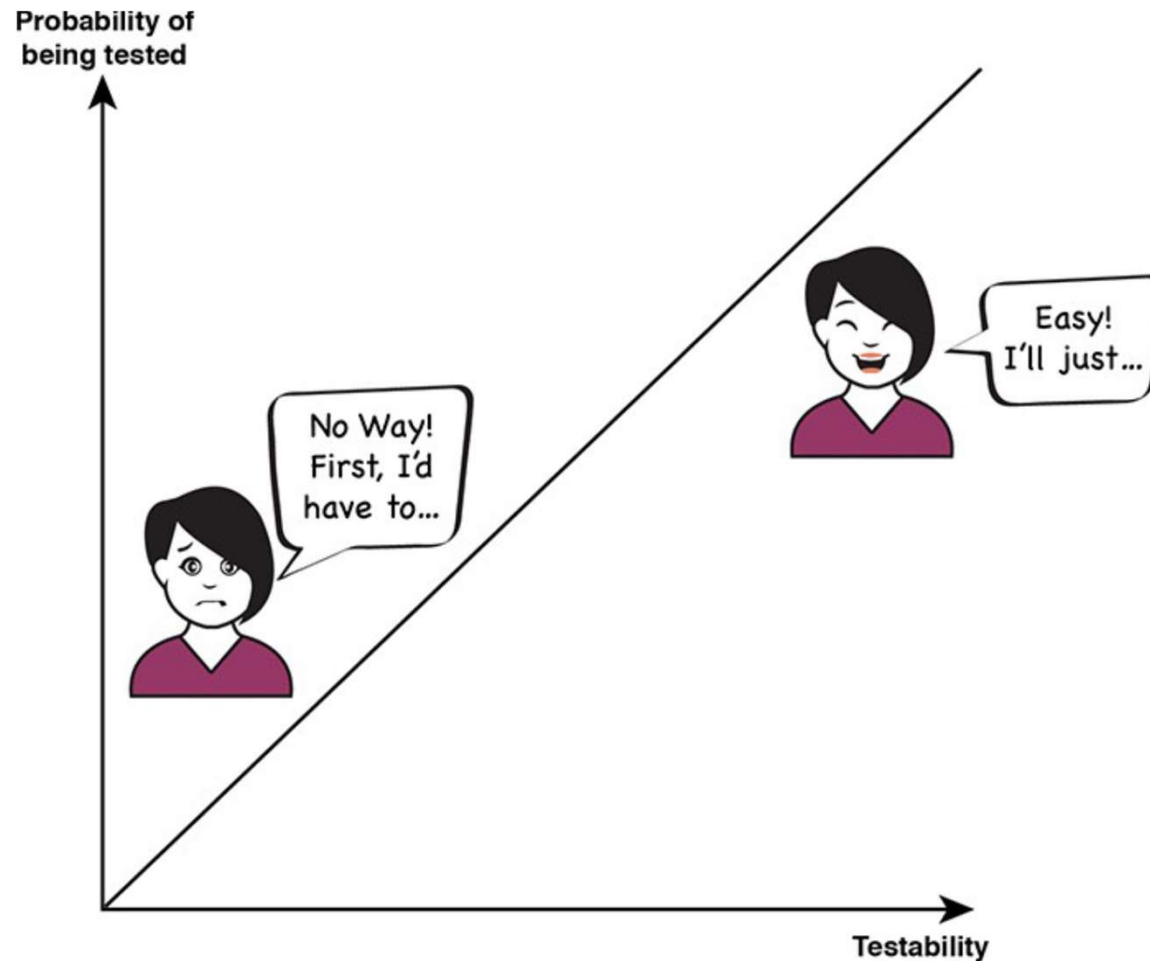
# Testable Software

- Testability is linked to our **prior experience** of the things we want to test and our tolerance for defects - **behavioral / human aspect**
- The 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



# Testable Software

Is untestable software going to be tested?



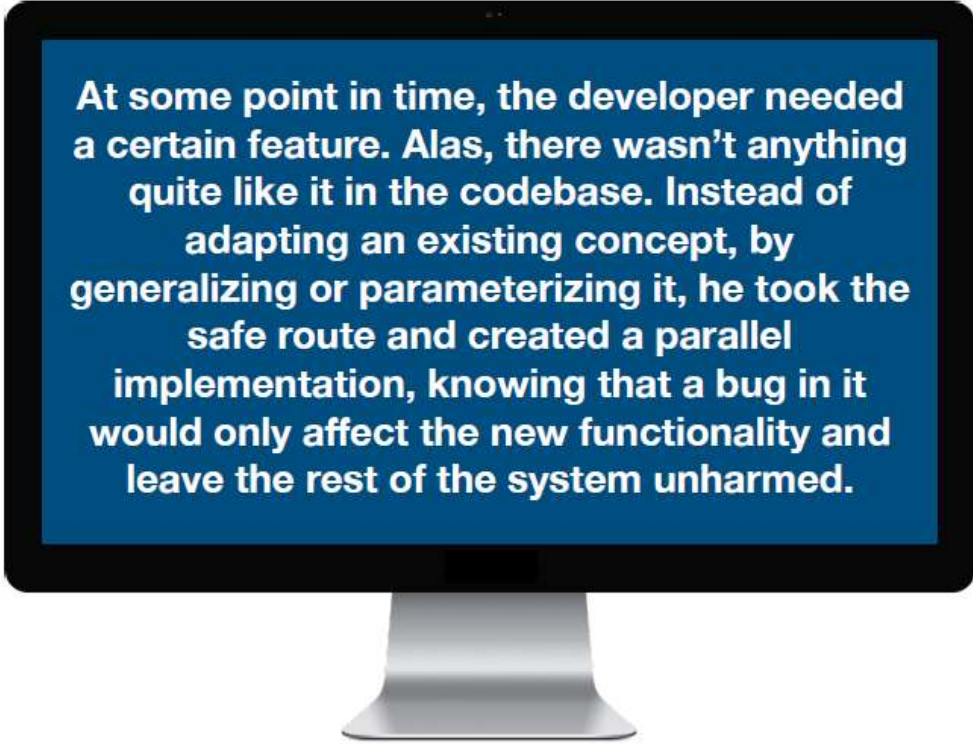
Picture source: Internet

14



# 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
- More easily changed – Fear results in duplication of defects



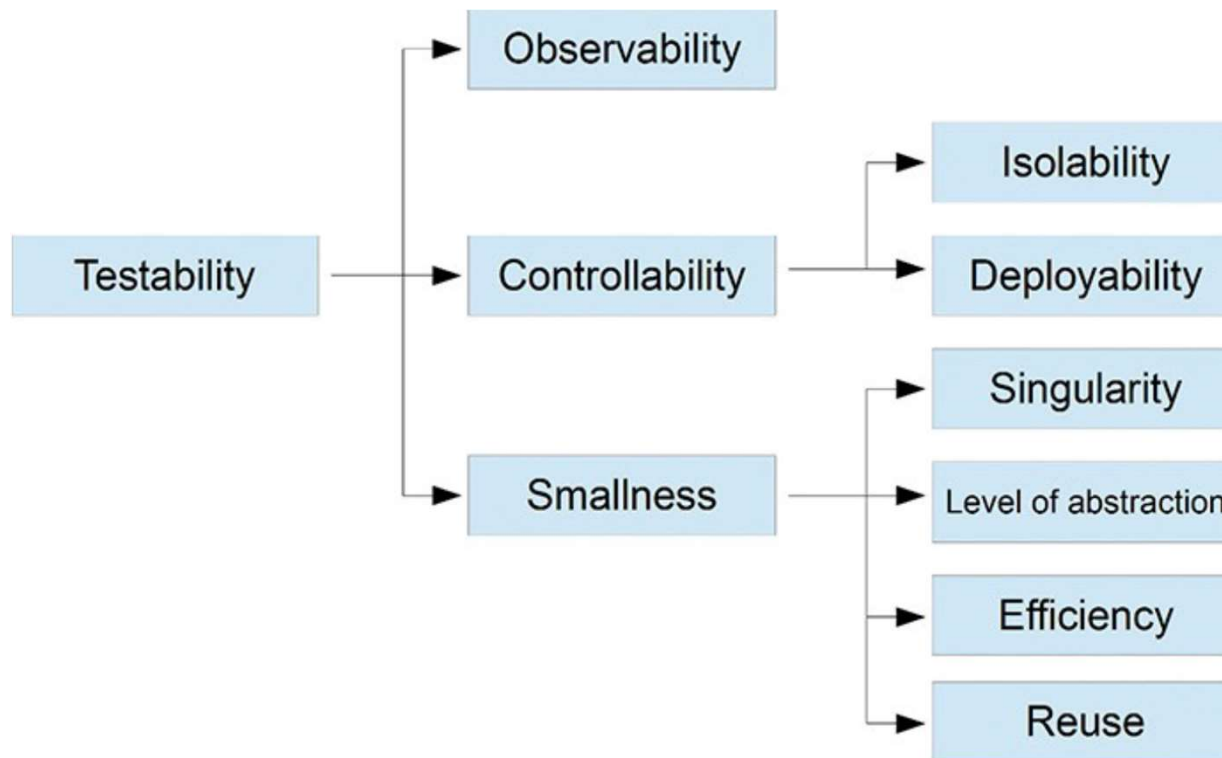
At some point in time, the developer needed a certain feature. Alas, there wasn't anything quite like it in the codebase. Instead of adapting an existing concept, by generalizing or parameterizing it, he took the safe route and created a parallel implementation, knowing that a bug in it would only affect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

Picture source: Internet

15



# 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 or interfering any other program elements**

Picture source: Internet

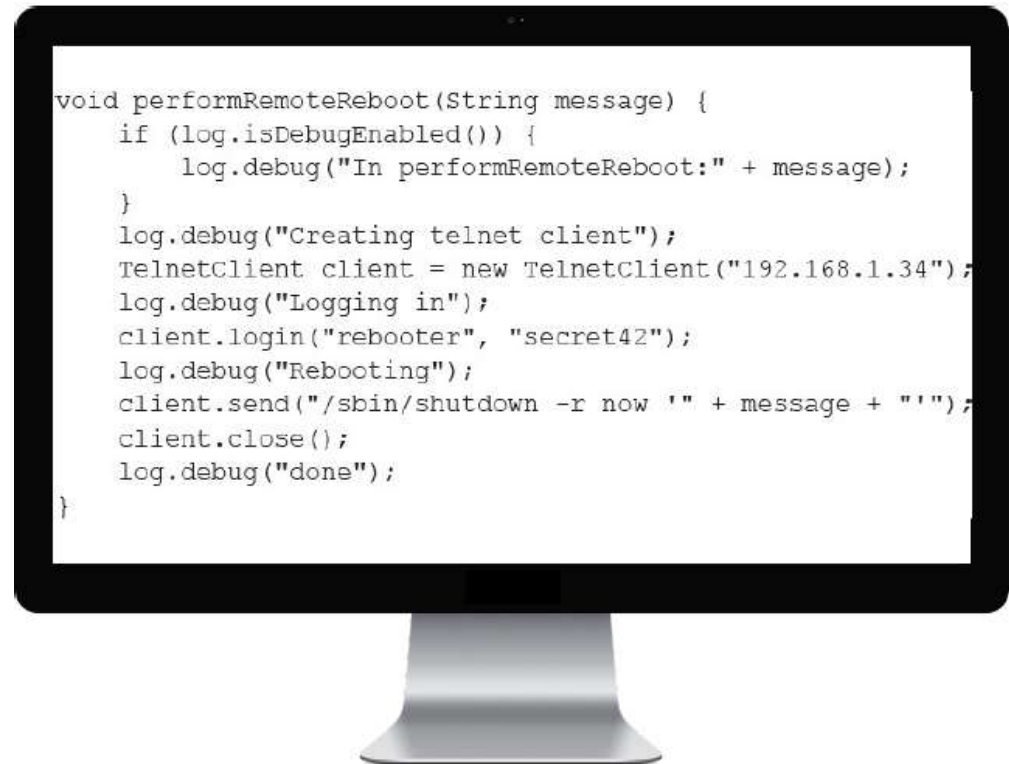
16





# 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



Picture source: Internet

17



# 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
    - open up the encapsulation by relaxing restrictions on accessibility to increase both observability and controllability
    - consider the fact that testing 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** at the given moment



Picture source: Internet

18



# Controllability: Isolability

- 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**.



# Controllability: Isolability

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



# Controllability: 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?"

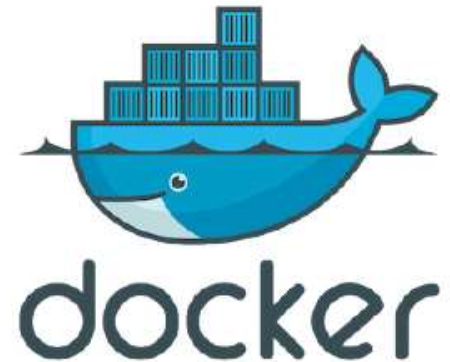
Picture source: Internet

21



# Controllability: 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**



Google Cloud Platform

Picture source: Internet

22



# 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





# Smallness

## **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)

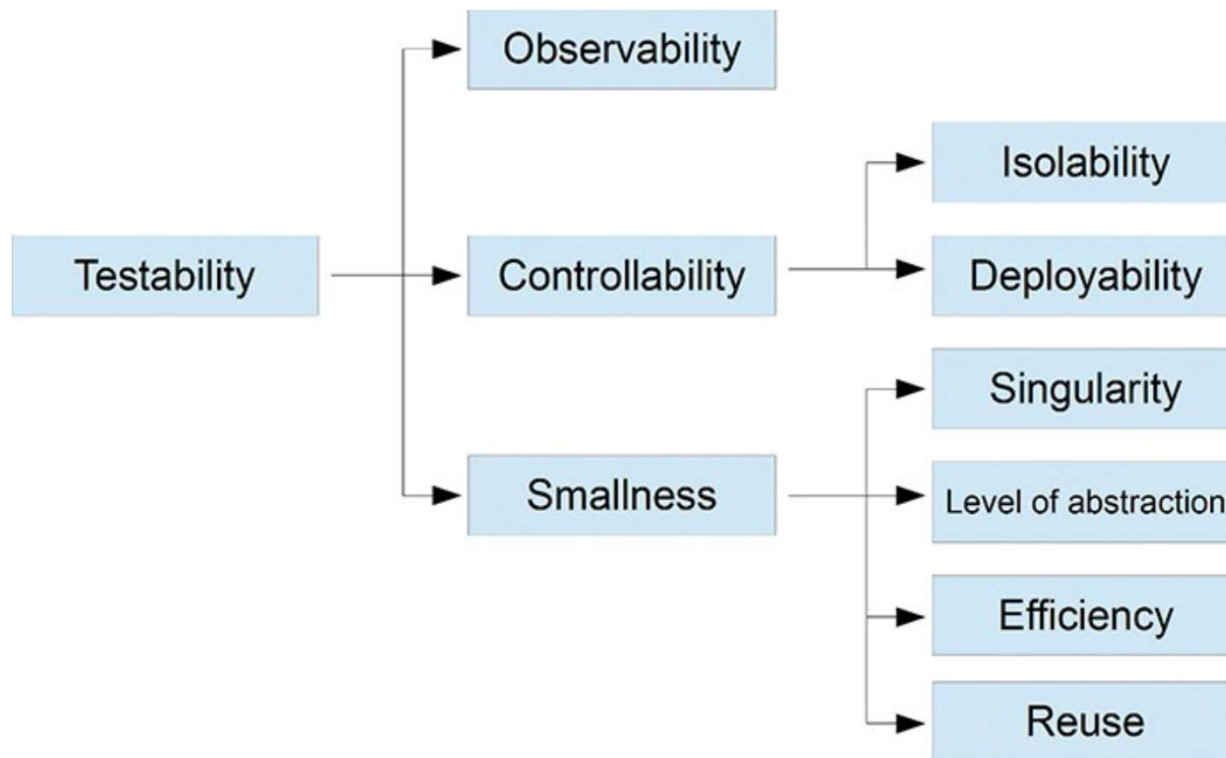
## **Reuse:**

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**

Picture source: Internet

25



# A Limited View of Correctness

- In traditional methods, we try to define all correct behavior completely, at the beginning
  - What is correctness?
  - Does “correctness” mean anything in large engineering products?
  - People are VERY BAD at completely defining correctness
- In agile methods, one way to redefine correctness is to be relative to a specific set of tests
  - If the software behaves correctly on the tests, it is “correct”
  - Instead of defining all behaviors, we demonstrate **some behaviors**
- Mathematicians may be disappointed at the lack of completeness

Correctness: few problems with limited damage to customers **demonstrated by observed (tested) behavior**



# Test Harnesses Verify Correctness

A *test harness* runs all automated tests efficiently and reports results to the developers

- Tests must be **automated**
  - Test automation is a prerequisite to test-driven development
- Every test must include a test oracle that can evaluate whether that test executed correctly
- The tests serve as a **specification of requirements**
- Tests must be of high quality and must run quickly
- Run tests every time we make a change to the software



# Continuous Integration

- Agile methods work best when the current version of the software can be run against all tests at any time

*A continuous integration server* rebuilds the system, returns, and re-verifies tests whenever *any* update is checked into the repository

- Mistakes are caught earlier
- Other developers are aware of changes early
- The rebuild and re-verify must happen as soon as possible
  - Thus, tests need to execute quickly

*A continuous integration server* doesn't just run tests, it decides if a modified system is still correct



# Continuous Integration

- A software engineering process where isolated code changes are immediately tested and reported on as they are added to a larger codebase
- Developers incorporate their progress and code changes to the codebase daily (or more frequently)
- The goal is to provide rapid feedback to identify and correct defects as soon as they are introduced
- Again, it enables automation
- Requires dedicated tools and automated tests to be written for the system



# Summary

- Testability
  - Observability
  - Controllability
  - Smallness
- Continuous Integration
- Next
  - Unit Testing
- To DO
  - Work on Assignment -1
  - Review today's slides



# THANK YOU



31



MISSISSIPPI STATE UNIVERSITY

TANMAY BHOWMIK

COMPUTER SCIENCE AND ENGINEERING