Announcements

- Project 1 presentation grades out - **Mean**: 17.07/16 (max: 19), **Stdev**: 1.42/16
- GCP Assignment Grades Released - **Mean**: 4.45/5, **Stdev**: 1.05/5
- REST Assignment Grades Released - **Mean**: 3.69/5, **Stdev**: 1.83/5
- Attendance Grades Updated (until last class)
- Participation Score Updated (until last class) - max score 1.5/3
- Assignment 5: Design due today
- Assignment 6: Test releasing today - also related to Project 1
- GCP coupons for Project 2
  - 18 ($50) coupons left – 2 (or 3) per team
  - 2 students from each group assigned credits
  - More credits to follow if needed

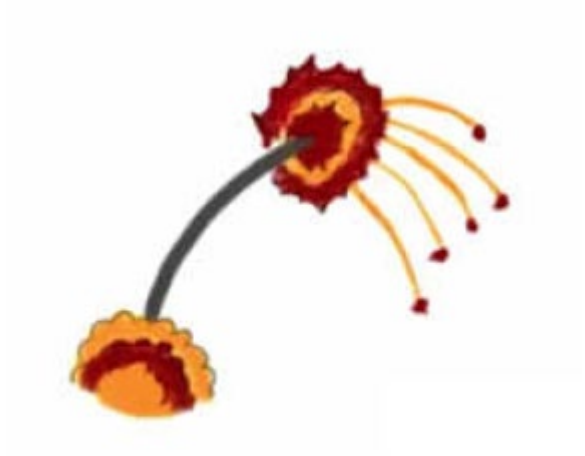
CS3300 Introduction to Software Engineering

Lecture 15: Software Testing

Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

Some Examples…



Ariane 5 Failure:
https://www.youtube.com/watch?v=gp_D8r-2hwk

Slide adapted from Alessandro Orso

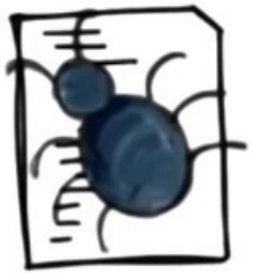
Software is Buggy!

- Cost of bugs: $ 60 B/year
- On average, 1-5 errors per 1K LOC
- Windows 10
  - 50M LOC
  - 63,000 known bugs at the time of release
  - 1.25 per 1,000 lines

- For mass market software 100% correct SW development is infeasible, but
- We must verify the SW as much as possible

Failure, Fault, Error

Failure: Observable incorrect behavior of a program. Conceptually related to the behavior of the program, rather than its code.



Fault (bug): Related to the code. Necessary (not sufficient!) condition for the occurrence of a failure.

Error: Cause of a fault. Usually a human error (conceptual, typo, etc.)

Failure, Fault, Error: Example

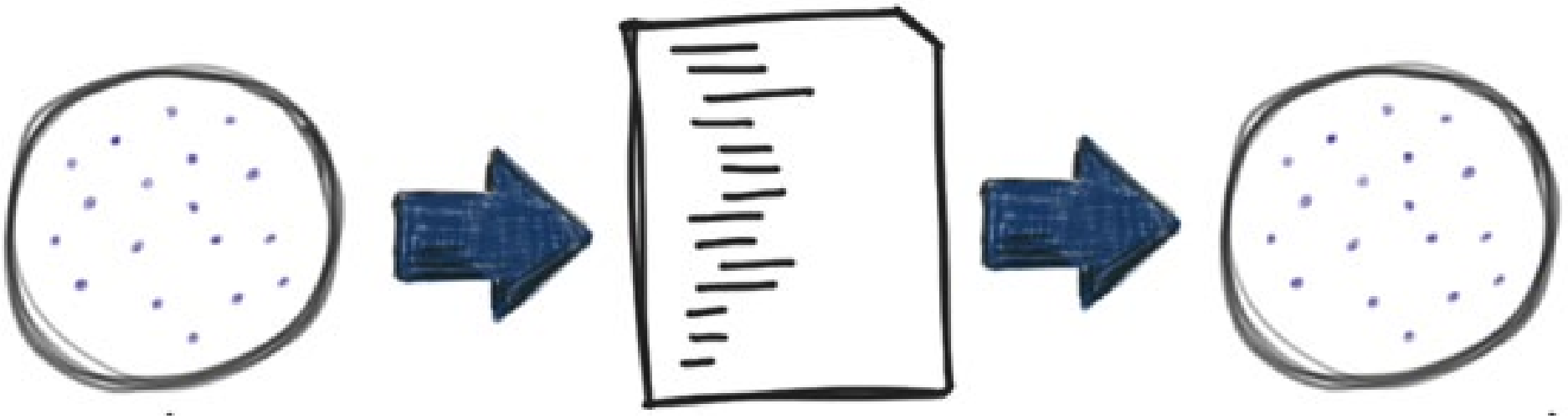
```
1.  double doubleValue(int param) {
2.      double result;
3.      result = (double) param * param;
4.      return(result);
5.  }
```

A call to double(3) returns 9. What is this?	The result 9 is a failure- it is an observable behavior
Where is the fault?	Line 3
What is the error that caused the fault?	N/A. Maybe typo, erroneous copy paste, or conceptual. Only the developer knows.

Approaches to Verification

- **Testing** (dynamic verification): exercising software to try and generate failures
- **Static analysis**: identify (specific) problems statically, that is, considering all possible executions
- **Inspections/reviews/walkthroughs**: systematic group review of program text to detect faults
- **Formal verification** (proof of correctness): proving that the program implements the program specification

Testing



Input Domain D

Software

Output Domain O

Test Case: {i ∈ D, o ∈ O}

Test Suite: A set of Test Cases

Static Verification

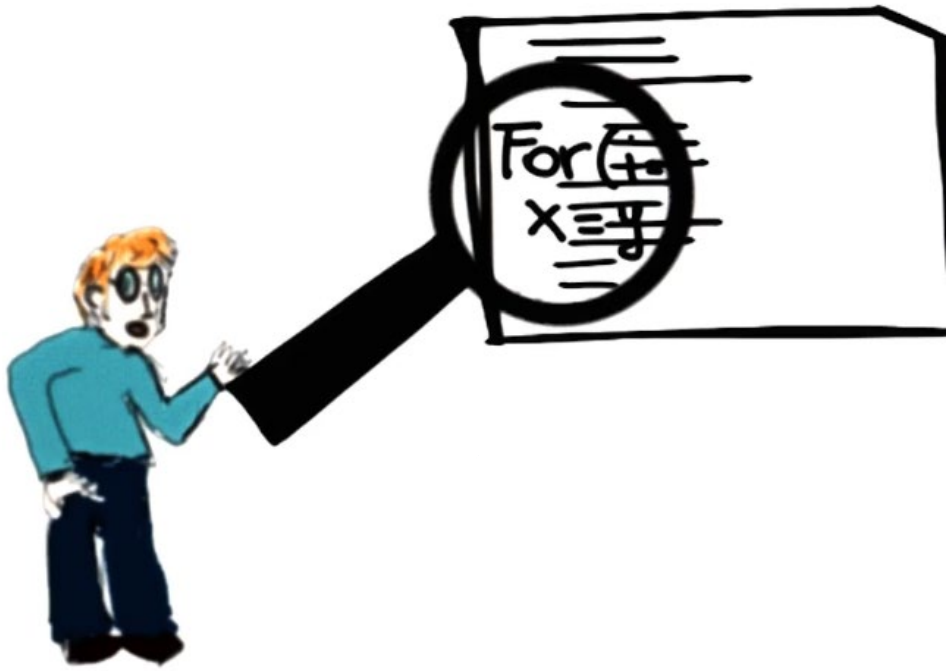
Input Domain D

Software

Output Domain O

Considers all possible inputs (execution/behaviors)

Inspections/Reviews/Walkthroughs



Human intensive activity
Manual
Group activity
Inspect defects in the artifacts

Formal Proof (Of correctness)

Program

Specification

Given a formal specification, checks that the code corresponds to such specification

Sophisticated mathematical analysis

Comparison among the 4 techniques

PROS

CONS

Testing	No False Positives	Highly Incomplete
Static Verification	Considers all program behaviors, Complete	False Positives, Expensive
Inspections	Systematic, Thorough	Informal, Subjective
Formal Proofs of Correctness	Strong Guarantees	Complex, Expensive to build/prove a mathematical basis

Today, Quality Assurance (Verification) is mostly Testing

"50% of my company employees are testers, and the rest spend 50% of their time testing". Who said that?

[ ] Mark Zuckerberg

[ ] Steve Jobs

[ ] Henry Ford

[✓] Bill Gates

[ ] Frank Gehry

What is Testing?

Testing == To execute a program with a sample of the input data

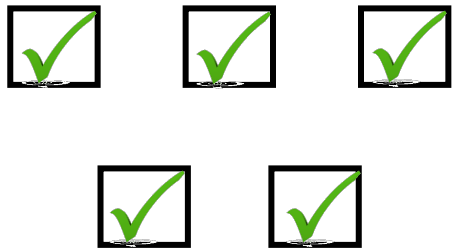
- Dynamic technique: program must be executed
- Optimistic approximation:
  - The program under test is exercised with a (very small) subset of all the possible input data
  - We **assume** that the behavior with any other input is consistent with the behavior shown for the selected subset of input data

Successful Tests

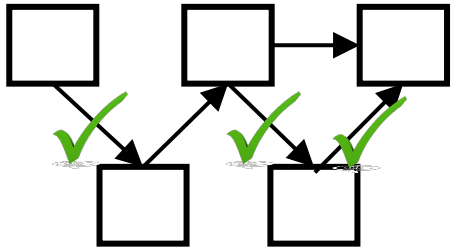
-Goodenough and Gerhart (1985). “Towards a Theory of Test data selection”. *IEEE Transactions of Software Engineering*, Jan 1985

Testing Granularity Levels

Unit Testing

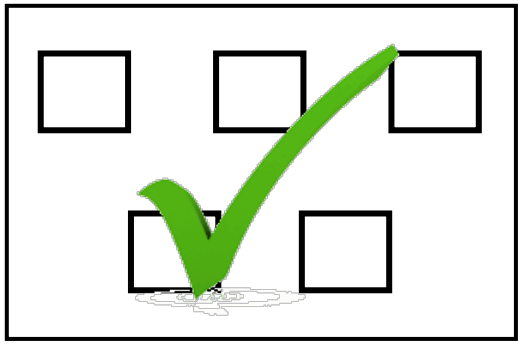


Integration



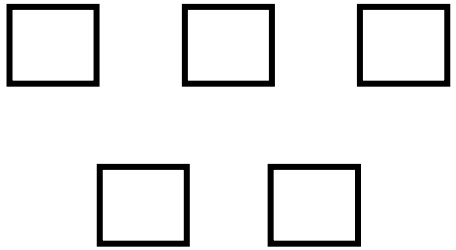
Big Bang

System



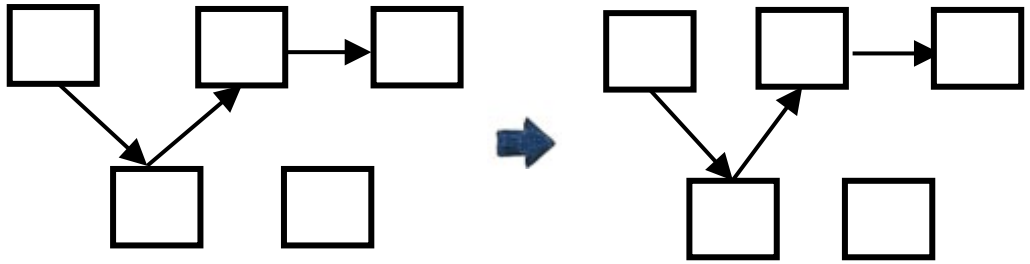
Functional/Non-functional

Acceptance Testing



Customer

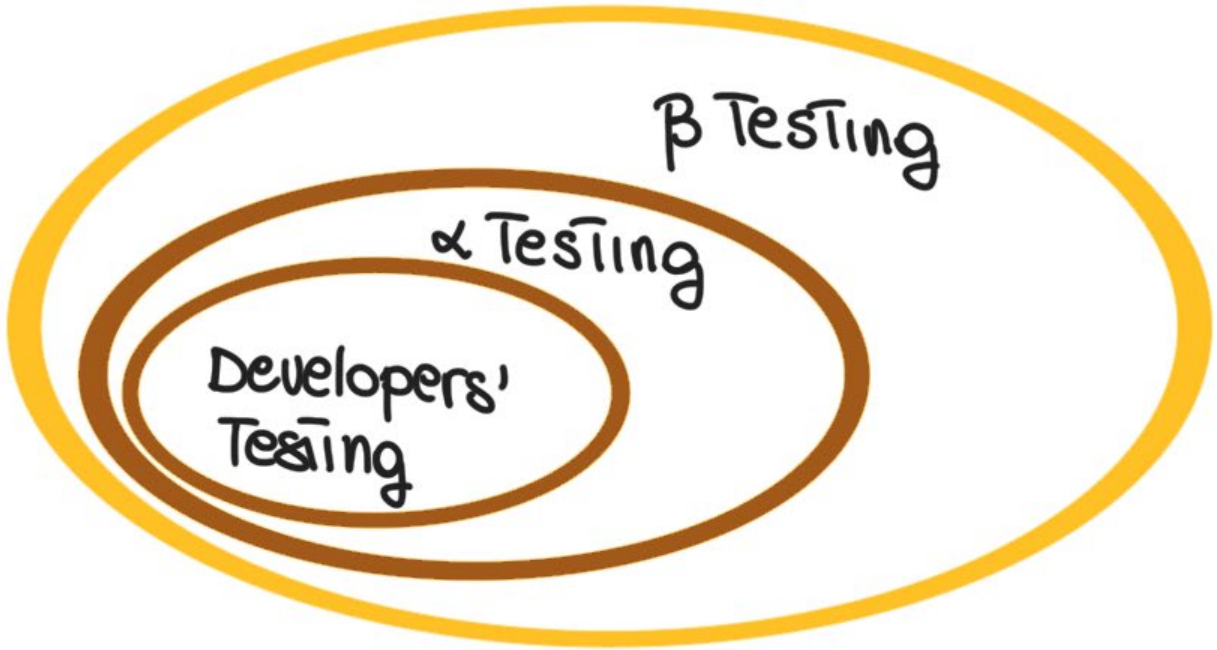
Regression Testing



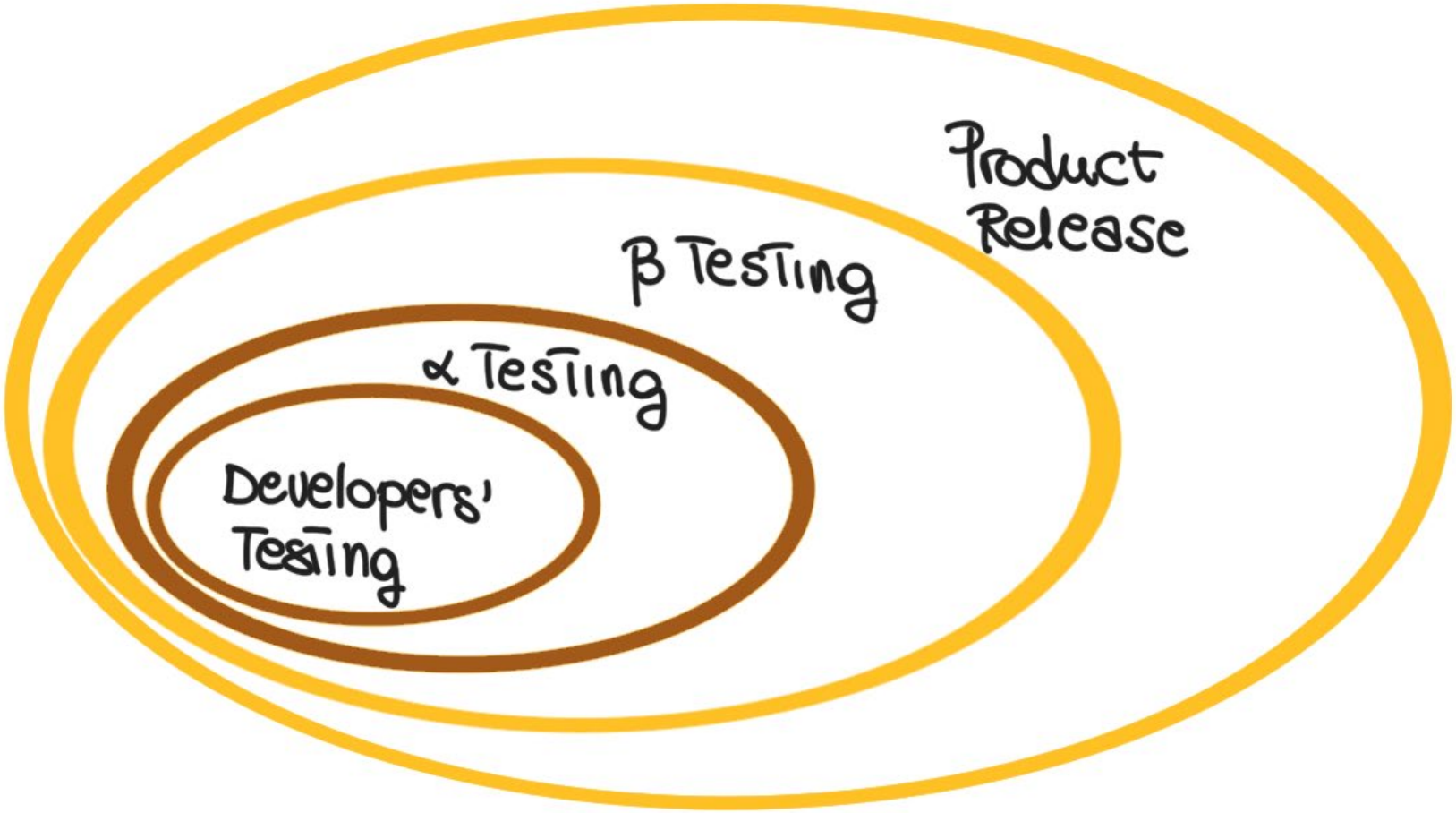
Testing Stages

Testing Stages

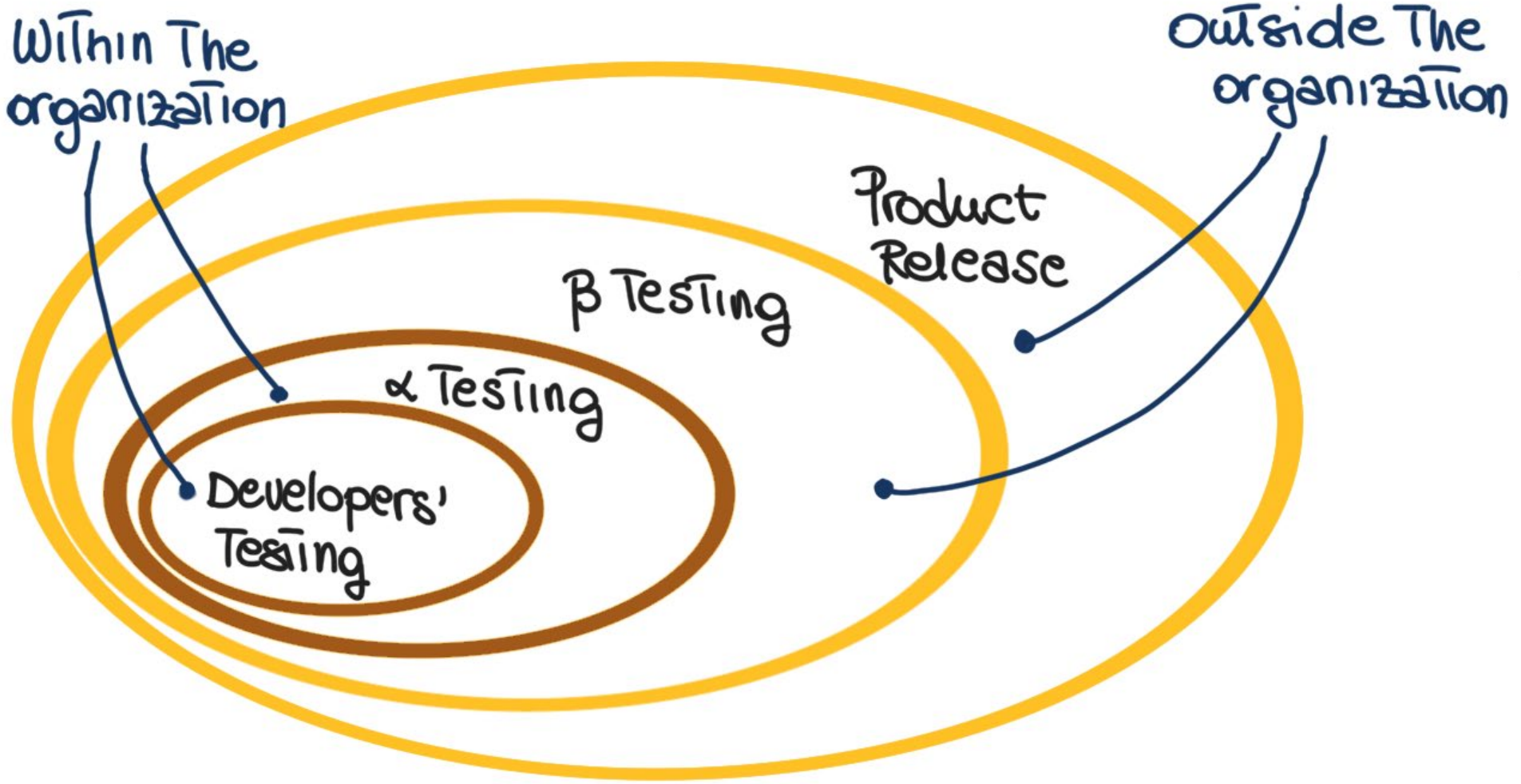
Testing Stages



Testing Stages



Testing Stages



Testing Techniques

There are several techniques

- Different processes
- Different artifacts
- Different approaches

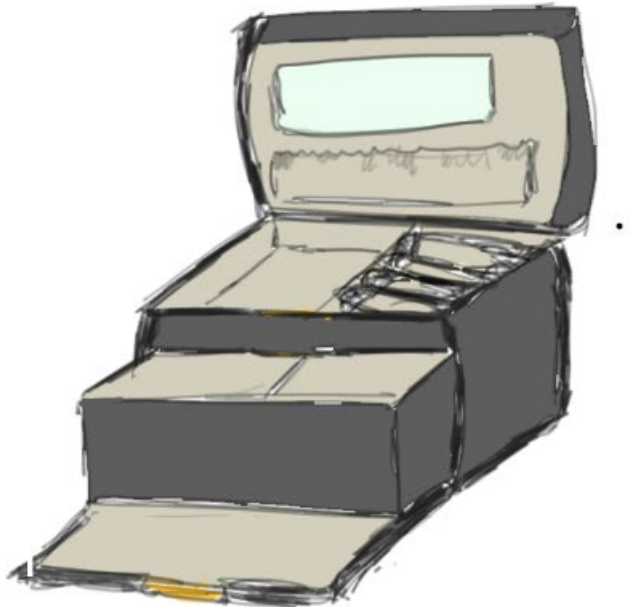
There are no perfect techniques

- Testing is a best-effort activity

There is no best technique

- Different contexts
- Complementary strengths and weaknesses
- Trade-offs

Testing Techniques



BLACK BOX TESTING

- Based on a description of the software (specification)
- Cover as much specified behavior as possible
- Cannot reveal errors due to implementation details

WHITE BOX TESTING

- Based on the code
- Cover as much coded behavior as possible
- Cannot reveal errors due to missing paths

Black-Box Testing Example

Specification: Inputs an integer and prints it

```
1. void printNumBytes (param){
2.      if (param < 1024) printf("%d", param);
3.     else  printf('%d KB" ,  param/124);
4. }
```

Blackbox testing attempts: Inputs +, -, and 0

Will blackbox testing be able to catch the failure? **Most likely Not**

White-Box Testing Example

Specification: inputs an integer param and returns half of its value if even, its value otherwise

```
1. int  fun(int param){
2.    int result;
3.    result = param/2;
4.    return result;
5.  }
```

Will whitebox testing be able to catch the failure? **No**

Blackbox testing attempts: atleast one odd and one even input – catches failure