
[CS3704] Software Engineering

Dr. Chris Brown

Virginia Tech

11/13/2023

Announcements

- Homework 4 due Friday (11/17)
 - Code metrics activity
 - Submit on Canvas

Assignment	Percentage
Project	35%
Exam	25%
Discussion Presentation	14%
Workshops	14%
Homework	8%
Ut Prosim	2%
Attendance & Participation	2%

Ut Prosim

The Virginia Tech motto is Ut Prosim which means “*That I May Serve*”. To embody this as stakeholders in the VT community, part of your course grade will be determined by completing **four** service activities to the university or CS department.

- Ex.) participating in a research study, attending a department seminar, volunteer project, etc.

Testing

Types of Testing

Software Testing Strategies

Testing Practices and Advanced Testing Techniques

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- Use techniques and processes to create and analyze requirements for an application
- Use techniques and processes to design a software system
- **Identify processes, methods, and tools related to phases of the SDLC**
- Explain the differences between software engineering processes
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

Testing



Goal: Execute software with intent of finding errors

- While you can't test until there is code to run, you can start planning testing when you're analyzing the requirements
 - Includes Unit (Ut) and System (St) tests to verify code and functionality
 - *Software Artifacts*: test code, bug database, test database, test inputs and outputs, documentation
-

Warm-Up

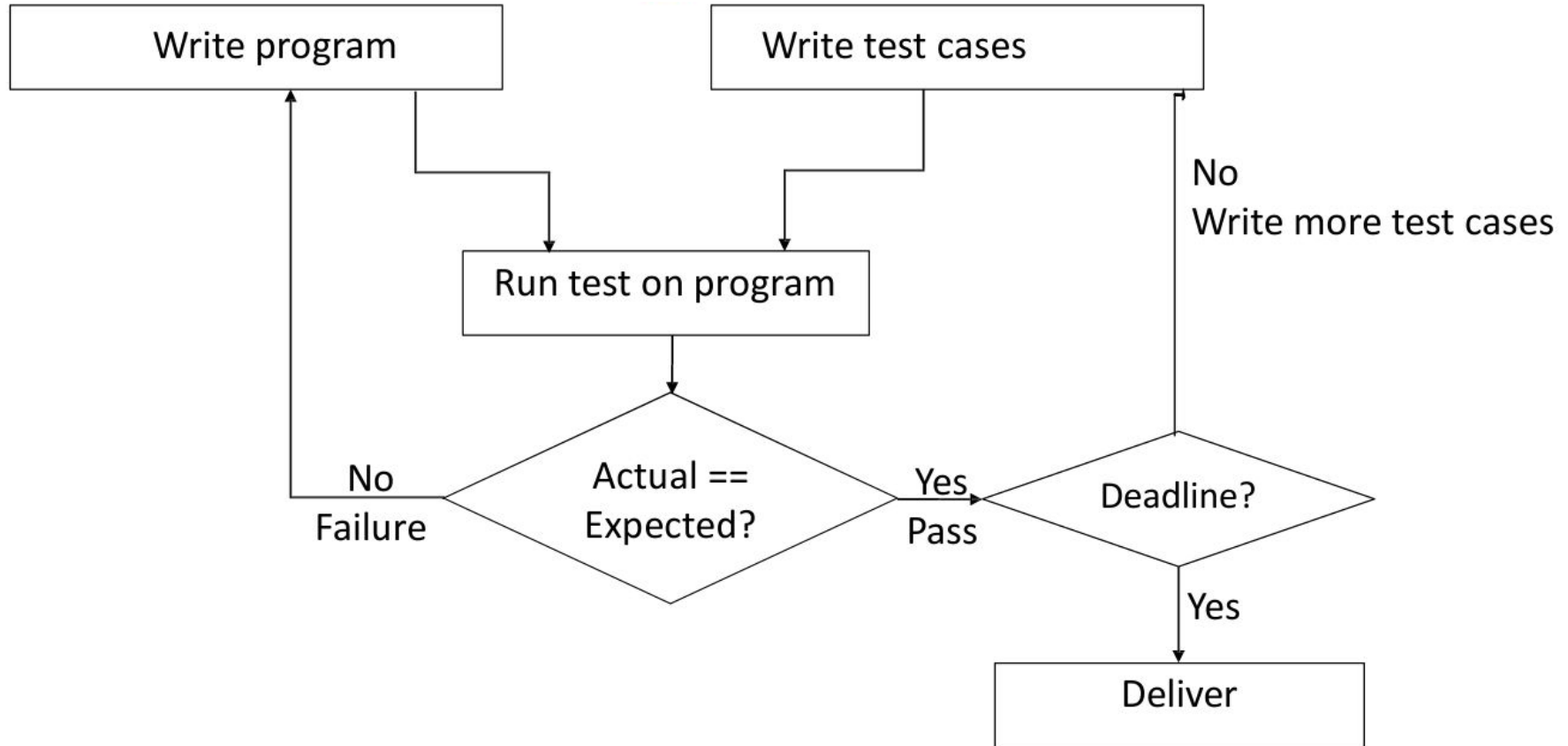
Find a partner or small group to discuss the following:

- **Stand-up: Done, TODO, Blockers**
 - **What is your experience with testing software? Discuss based on prior industry experience, classwork, projects, etc.**
-

What is Testing?

- Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.
 - **Discovery:** *Executing a program or system with the intent of finding errors. [\[Myers\]](#).*
 - **Verify:** *Any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. [\[Hetzel\]](#)*
-

Testing Process



Discovery: Types of Bugs

- Requirements and specification bugs
 - Feature bugs
 - Feature interaction bugs
 - Logic bugs
 - Data-flow bugs
 - Processing bugs
 - Integration bugs
 - Coding bugs...
-

Benefits



- Improves code quality,
 - Catching bugs earlier reduces debugging time and costs,
 - Provides documentation,
 - Enhances software maintenance,
 - Increases ability to add more features iteratively (agile),
 - and more!
-

Who should test software?



developer

Understands the system
but, will test "gently"
and, is driven by "delivery"



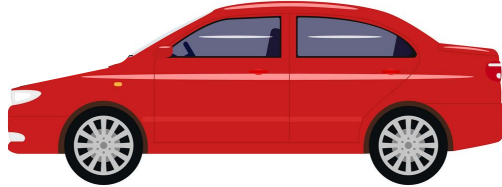
independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

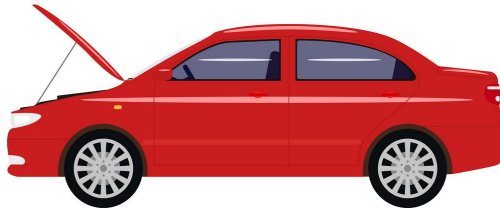
Ideally, both

Types of Testing

1. **Black Box Testing:** ignores the internals of the program and focuses on functionality.



2. **White Box Testing:** uses code to guide testing.



Black Box Testing

- Focus on I/O behavior
 - If for any given input, we can predict the output, then the component passes the test.
 - Requires a test oracle
 - Reduce number of test cases by using *equivalence classes*
 - Divide input conditions into classes of values
 - Choose one test case for each class
 - ***Another measure for minimum number of test cases***
 - i.e. dealing with integers, use: >; =; and <
-

Black Box Testing: Equivalence Classes

```
public class MyCalendar {  
    public int getNumDaysInMonth(int month, int year)  
        throws InvalidMonthException  
    { ... }  
}
```

Representation for month:

1: January, 2: February,, 12: December

3!

Below range (i.e. 0); Within range (i.e. 3); and Above range (i.e. 14)

Boundary Value Analysis

- Similar concept to equivalence classes.
- Testing values around boundaries with conditional statements.

...

```
Scanner console = new Scanner(System.in);  
System.out.print("Enter a number between 1 and 10: ");  
int num = console.nextInt();  
if (num >= 1 && num <= 10) {
```

... **Ex) 0, 1, 2, 9, 10, 11**

White Box Testing

- **Reminder:** Have knowledge of the code, and use it to motivate testing
 - Most popular: *unit testing*
 - **Unit:** individual functions or methods
 - Usually organized in a separate test directory in the same repository to test your program methods.
 - Differs depending on programming language, team, etc.
-

White Box Testing (cont.)

Cyclomatic Complexity!

- Number of independent paths in a program
Cyclomatic Complexity = #decisions + 1
 - ***Minimum*** number of white box test cases to write for a method.
-

Cyclomatic Complexity Review

```
public static String evenOdd(int num) {  
    if (num == 0) {  
        return "Even";  
    } else if (num == 2) {  
        return "Even";  
    } else if (num == 4) {  
        return "Even";  
    } else if (num == 6) {  
        return "Even";  
    } else if (num == 8) {  
        return "Even";  
    } else {  
        return "Odd";  
    }  
}
```

6

White Box Testing (cont.)

Assert Statements

- Used to programmatically verify the *expected* output matches the *actual* output
 - Assertion libraries differ based on testing framework, programming language, etc.
-

Assert Statements Example

```
public static String evenOdd(int num) {  
    if (num == 0 || num == 2 || num == 4 || num == 6 || num == 8) {  
        return "Even";  
    } else {  
        return "Odd";  
    }  
}
```

Given the code above, will the following (JUnit) test cases pass or fail?

1. assertEquals("The expected output is Even", "Even", evenOdd(6)); **// PASS**
2. assertEquals("The expected output is Odd", "Odd", evenOdd(1)); **// PASS**
3. assertEquals("The expected output is Even", "Even", evenOdd(10)); **// FAIL**
4. assertTrue("The expected output is Odd", "Odd".equals(evenOdd(99))); **// PASS**
5. assertFalse("The expected output is Even", evenOdd(8).equals("Even")); **// FAIL**

Writing Test Cases

- **Test case:** Specific functionality or feature to test
 - Used for black box and white box testing
 - **Required test case information:**
 1. Unique Identifier
 2. Input
 3. Expected Output
 4. Actual Results
-

Black Box Test Cases

STATUS: OPEN VERIFIED CANCELED

Test ID	Description	Expected Results	Actual Results
TestName (Test Author)	Preconditions:	Test Outputs	Actual Outputs
Test Type	Steps: Test Inputs		

White Box Test Cases

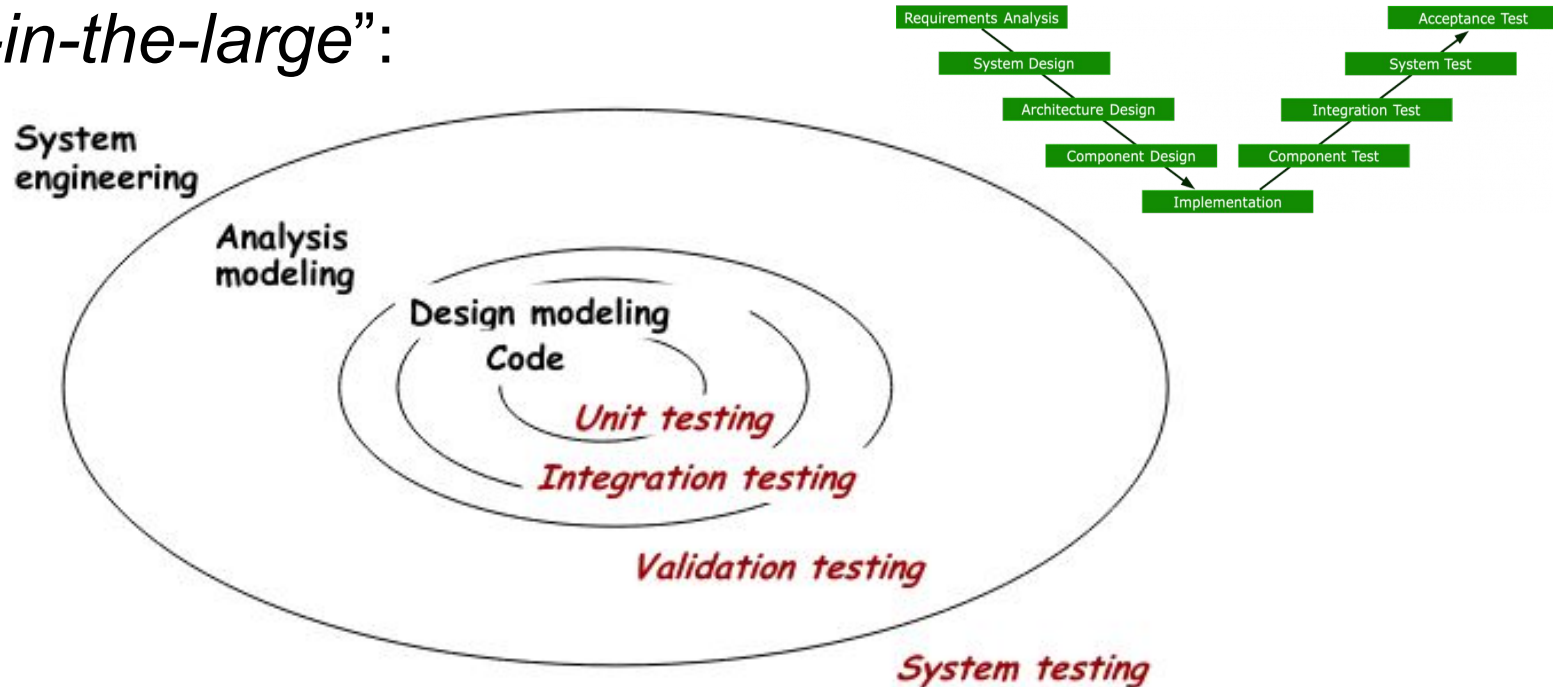
```
... /**
...  * Test evenOdd method with zero
...  */
... @Test
... public void testEvenOddZero() {
...     int num = 0;
...     assertEquals("Expected output for 0 is Even", "Even", EvenOrOdd.evenOdd(num));
... }
```


Code Testing Metrics

- Total number of test cases
 - Number of test cases passed
 - Number of test cases failed
 - Number of test cases blocked
 - Number of defects found
 - Number of defects accepted
 - Number of defects rejected
 - Number of defects deferred
 - Number of critical defects
 - Number of planned test hours
 - Number of actual test hours
 - Number of bugs found after shipping
 - ...
-

Testing Strategies

Begin by “*testing-in-the-small*” and move toward “*testing-in-the-large*”:



Strategy 1: Unit Testing

- **What are you testing:** Individual components on the smallest unit of software design (function, class, or subsystem)
 - **Who?:** Carried out by developers
 - **Goal:** Confirm that units are correctly coded and information flows in and out of modules properly.
-

Strategy 2: Integration Testing

- **What are you testing?:** Groups of subsystems, and eventually the entire system
 - **Who?:** Carried out by developers (mostly, could also be testers)
 - **Goal:** To construct tests to uncover errors associated with interfacing between units
-

Strategy 3: Validation Testing

- Also known as *acceptance testing*
 - **What are you testing?:** the system delivered by developers
 - **Who?:** Carried out by testers and/or clients (through typical product interactions on a trial basis)
 - **Goal:** Demonstrate that the system meets requirements and is ready to use
-

Strategy 4: System Testing

- **What are you testing?:** The entire system
 - **Who?:** Developers and/or testers
 - **Goal:** Determine if the system meets the requirements (functional and nonfunctional)
 - **How is this different from validation testing?**
 - *Not from the user's perspective*
-

Advanced Testing Strategies

- There are a lot more recent strategies for testing and deploying software, which we will discuss later in the lecture...

Testing Practices

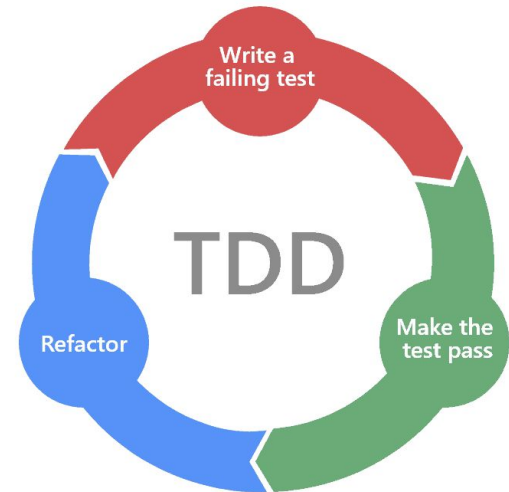
Processes used to test software in industry

- TDD and BDD
 - Mocking
 - Test Frameworks
 - Automation
 - Mutation Testing
-

Test-Driven Development (TDD)

- Write tests before writing code
- Emphasizes tests and software quality over features

1. Write unit tests (**white box!**) based on requirements
2. Run the tests
 - a. Expected to fail ❌
3. Write the simplest code to pass the tests
4. Run the tests
 - a. Expected to pass ✅
5. Refactor code as needed
6. Repeat



TDD (cont.)


“What Do We Know about Test-Driven Development?” [Shull, 2010]

...evidence from controlled experiments suggests an improvement in productivity when TDD is used. However...pilot studies provide mixed evidence, some in favor of and others against TDD. In the industrial studies...evidence suggests that TDD yields worse productivity. Even when considering only the more rigorous studies...the evidence is equally split for and against a positive effect.

Exercise

Given what you know about implementation processes and now testing, complete the following activity:

Find a partner. Pick one person to be the *advocate* and the other to be the *detractor*:

- The detractor starts by giving a reason why TDD programming wouldn't work
- The advocate explains why it will
- Iterate 

Keep a list of advantages and disadvantages discussed.

Mocking

- The process of creating temporary objects for tests.
 - **Why?:** To avoid testing dependencies on other or production systems during testing (database, filesystem, etc.)
 - Can be done using tools, third-party libraries, or hard-coded data.
-

Unit Testing Frameworks

- Used to organize and execute tests
- Examples include:
 - JUnit (Java),
 - CUnit (C),
 - pytest (Python),
 - Mocha (JS),
 - minitest (Ruby),
 - Etc.

Discuss: What is your experience with unit testing and test frameworks?

Validation Testing Frameworks

- Tools to organize and evaluate acceptance tests based on requirements
 - Mostly BDD tools
- Examples: Cucumber, rspec (Ruby), Selenium (UI)...

Feature: Browse Wikipedia

- As a user I want to be able to navigate to Wikipedia so that I can perform simple searches online

Acceptance Criteria: Load Wikipedia

When I navigate to the Wikipedia site

Then I should be on the Wikipedia site

```
@When ( "I navigate to the Wikipedia site" )  
public void navigateToWiki () {  
    driver.get( "https://de.wikipedia.org" );  
}  
  
@Then ( "I should be on the Wikipedia site" )  
public void onWiki () {  
    assertTrue( driver.getCurrentUrl().contains(  
"de.wikipedia.org/wiki" ) );  
}
```

Test Automation

- The rise of agile practices led to more frequent customer feedback.
- Too many tests were a liability
 - More difficult to update, organize, and maintain
 - Slow tests slow everyone and everything down
- **Test automation:** the practice of automatically reviewing and validating software products

★ Test automation drives CI/CD & DevOps (More on 11/27)

Advanced Testing Techniques

- As software projects become more complex, SE teams have adopted new testing strategies:
 - Regression testing
 - Smoke testing
 - Alpha/Beta testing
 - Performance testing
 - Security testing
 - Stress testing
 - Recovery testing
 - Usability testing... and more!
-

Regression Testing

- As new modules are added in integration testing, regression testing reruns the already execute tests to ensure that software changes do not cause problems in existing functions.
 - i.e, no unintended side effects of changes
-

Alpha and Beta Testing

- **Alpha testing:** Acceptance tests by a representative group of end users at the development site.
 - **Beta testing:** “Live” applications of the system in an environment with no developers present.
-

Alpha and Beta Testing (cont.)

- **Alpha Testing**

- Insight into how customers will use a program
- Informal with a series of planned or systematically executed actions
- Users use the software in a natural setting with developers “looking over the shoulder”

- **Beta Testing**

- Helps uncover errors only users would find
 - Customers record and report all problems encountered at a regular interval
-

Security Testing

- To check whether security protection mechanisms will actually protect the software.
 - Acquiring passwords, hacking software, intentional crashes, etc.
 - Given enough time, resources, etc. good security testing should always penetrate the system.
 - **Development goal:** Make penetration cost higher than the value of the information retained
-

Recovery Testing

- How does the system recover after crashing?
 - Force the software to fail in a variety of ways, then verify the recovery was properly performed.
 - Automatic recovery
 - Evaluate whether initialization, data recovery, restart, etc. are correct
 - Manual recovery
 - Evaluate Mean-Time-To-Repair (MTTR) to determine speed and acceptability of recovery
-

Attendance

Email the notes from the in-class exercise to the instructor (dcbrown@vt.edu). One student needs to send the email which should also include the names and PIDs of each student as well as the role they completed.

If you hand-wrote your notes, give them to Dr. Brown in class with the same information.

Next Time...

- Testing and Implementation workshop next class (11/15)
 - Homework 4 (due Friday at 11:59pm)
 - Discussion Presentations on *Testing* [11/17]
 - Continue working on your project!
-

References

- RS Pressman. *“Software engineering: a practitioner's approach”*.
 - Brad Myers, et al. *“The art of software testing”*. 1979.
 - William C. Hetzel. *“The complete guide to software testing”*. 1988.
 - Na Meng and Barbara Ryder
 - Chris Parnin
 - Sarah Heckman
-