



TESTING, TDD

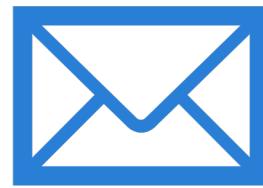
WEEK 6 – 10/02/2019



UPDATE



Next week lectures are
going to delivered by Annika



If you need anything feel
free to shoot me an email



In case of anything urgent,
reach out to your TAs

THINGS TO REMEMBER

- Please submit your work to canvas even if you completed it in the lab and committed everything to git.
- In Homework 1, if you are going for the extra credit and do opt to add additional content on your website, please list them down in a text file and upload them on Canvas. That will make it easier for the TA to grade you for the extra credit.

MIDTERM 1 SCHEDULE

- Date: Wednesday, Oct 16, 2019
- Time: 7pm - 8:30pm
- Duration: 45 mins
- Location:

| Last Names | Room |
|------------|-----------|
| A - K | RAMY C250 |
| L - S | FLMG 155 |
| T - Z | KOBL 210 |

- If you need accommodations, please email me ASAP

WHY TEST?



You already do!

- No one writes code, and doesn't want to see it run.

It's the only way to know if a project is done.

- Until it is tested, code is just structured hopes.

Objectively measure software performance

- What will your users really see under worst conditions?

Prevent expensive / stressful patches & hotfixes

- Planning can save your time and reputation.

CAUSES OF ERRORS

Software systems are complicated!

Bugs in code

Error in requirements

Integration issues

Security flaws

Unexpected States & Conditions





DIFFERENT KINDS OF TESTING



USE CASE TESTING – matching functionality to requirements



LOAD TESTING – how does it perform under stress?



REGRESSION TESTING
– How do I know my changes didn't break something that was working before?



USER ACCEPTANCE TESTING - Does it do what they expect? Can they use it?



SYSTEM TESTING – Combining many working components to see if they work together.



UNIT TESTING – Does this piece of code do what it is written to do?



INTEGRATION TESTING – Can I add a module to existing, working software without breaking it?



BLACK BOX TESTING – Testing its features without peering into its structures/code

USE CASE TESTING

Use Cases capture the interactions between 'actors' and the 'system'.

Test cases based on use cases and are referred as scenarios.

Capability to identify gaps in the system which would not be found by testing individual components in isolation.

Very effective in defining the scope of acceptance tests.

REGRESSION TESTING

- Consists of re-executing those tests that are impacted by the code changes.
- These tests should be executed as often as possible throughout the software development life cycle.
- Types of Regression Tests:
 - **Final Regression Tests:** -
 - A "final regression testing" is performed to validate the build that hasn't changed for a period of time.
 - This build is deployed or shipped to customers.
 - **Regression Tests:** -
 - A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

USER ACCEPTANCE TESTING

- Testing methodology where the clients/end users involved in testing the product
- Users validate the product against their requirements.
- It is performed at client location at developer's site.
- The acceptance test activities are designed to reach at one of the conclusions :
 - Accept the system as delivered
 - Accept the system after the requested modifications have been made
 - Do not accept the system

USER ACCEPTANCE TESTING

Objectives

- Confirm that the system meets requirements
- Identify, document and resolve any discrepancies
- Determine readiness for code/module/system deployment to production
- Can be a set of steps to ensure functionality and/or a series of subjective questions
- Each test is planned and documented according to requirements
- Each test execution is documented
- Any bugs are passed back to the developers for re-work
- Very time consuming, but very necessary

THE IDEAL USER ACCEPTANCE TESTER

Background

- Understands the user requirements Skills

Good Communicator

- Understands the System
- Technical OR Non-Technical
- Attention to detail

Availability

- Fully Dedicated to Conducting the Tests,
Documenting Results



UAT – DELIVERABLES

Test plan - Outlines the testing strategy:

- Acceptance Testing
- Entry and Fail Criteria
- Test Execution Team
- Test Script Developer

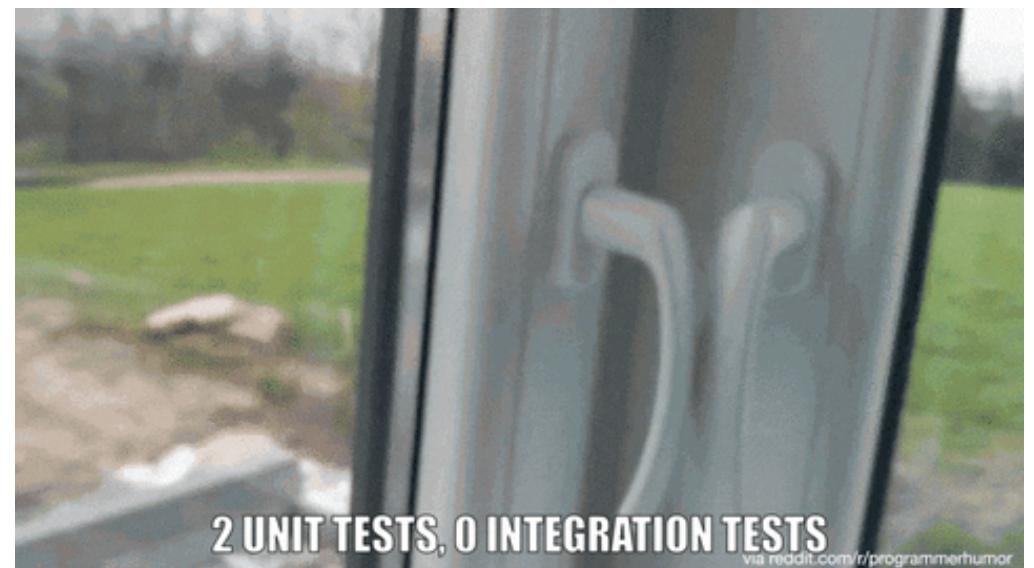
Test cases - Guide the team to effectively test the application

Test Log - records all the test cases executed and the actual results

User Sign Off - indicates that the customers are satisfied with the product.

INTEGRATION TESTING

- Individual units are combined and tested as a group.
- The purpose of this level of testing is to expose faults in the interaction between integrated units.
- Test drivers and test stubs are used to assist in Integration Testing.
- Identify, document and resolve any discrepancies



PERFORMANCE TESTING

- The primary goal of performance testing includes establishing the benchmark behavior of the system.
- Some types of performance testing include:
 - Load testing:
 - Volume testing: Mainly focuses on databases
 - Endurance testing: Testing the sustainability of the system under continued load
 - Stress testing:
 - Analyze post-crash reports to define the behavior of the application after failure.
 - Ensure that the system does not compromise the security of sensitive data after the failure.

UNIT TESTING

- Ensure that every line of code does exactly what it is supposed to do.
- Change anything? You must do regression testing to make sure you didn't break anything.
- A “unit” = the smallest possible unit of code behavior that can be tested in isolation.
- Test the code for
 - Handling expected inputs properly
 - Handling unexpected inputs properly
 - Graceful error handling
 - Edge case: handling the extremes
 - 21 bytes in a 20-byte field
 - Character data in a numeric field
 - Divide by zero

WHY AUTOMATE UNIT TESTING?



Speeds up Unit Testing



Enables Speedy, Reliable
Regression Testing



Ensures that Function
meets Design

HOW TO AUTOMATE UNIT TESTING?

Use a “Testing Framework” to develop some testing software that will do this for a given action in your code:

- Determine the expected value (the value which should be produced if the software is working correctly)
- Determine the actual value (the value which the software is actually computing)
- Compare the two:
 - If they agree, the test passes
 - If they disagree, the test fails

AUTOMATED UNIT TESTING FRAMEWORKS

- Jest: for JS <https://jestjs.io/docs/en/getting-started>
- Jasmine: for JS <https://jasmine.github.io/>
- Chai: for JS <https://www.chaijs.com/>
- PyUnit: <http://wiki.python.org/moin/PyUnit>
- Python's unittest
<https://docs.python.org/3/library/unittest.html>
- PHPUnit: <https://phpunit.de/>
- Unit testing with C#:
<http://www.csunit.org/tutorials/tutorial7/>

TEST-DRIVEN DEVELOPMENT (TDD)



TDD is a developer process of writing unit-tests first, then writing code to pass the tests.



Tests are executable requirements/specifications



End-result is a complete system (working code) with corresponding, automated unit tests

Assist with regression testing
Enable refactoring



NO CODE is ever written without a test being created first

TEST-DRIVEN DEVELOPMENT (TDD)



DEVELOPMENT
BEGINS AND ENDS
WITH TESTING



KNOW IMMEDIATELY
WHEN AND WHERE
THINGS BREAK



FUNCTION OF EACH
UNIT IS EVIDENT
FROM ITS TEST



GOAL: 100% TEST
COVERAGE

TDD

Part of agile approaches like XP (Extreme Programming)

Supported by testing framework tools (like Chai, PyUnit)

TDD Is more than a mere testing technique; it incorporates much of the detail design work

Useful for many code behaviors, but not really for GUI or Database functionality

It is not a replacement for UA, System, Performance testing

BENEFITS OF RIGOROUS TESTING / TDD



Certainty



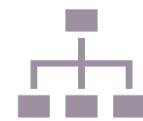
Reduced Bugs



Courage



Documentation



Enforces
Modular Design



Deliberate
Design

THINGS TO REMEMBER ABOUT TESTING



Test your code!

Any amount of testing, of any kind,
is better than no testing.



Please, test your code!

I'm not kidding! It will make you a
happier, better person!



**Please, test as much of your code
as possible, as soon as possible!**

Would you rather debug a 20 line
function or a 2000 line module?