CS1632, Lecture 4: Test Plans and TM

Wonsun Ahn

You’ve got requirements.
You’re looking for defects.

How?

Develop a test plan!

Formality

- This could be as formal or informal as necessary.
- Think about what you are testing – what level of responsibility / tracking is necessary?

What are you testing?

- Throw-away script?
- Development tool?
- Internal website?
- Enterprise software?
- Commercial software?
- Operating system?
- Avionics software?

Testing is context-dependent

- How you test
- How much you test
- What tools you use
- What documentation you provide

- ...All vary based on software context.

Formal Test Plans

- A test plan is a sequence of test cases.

- A test case is the fundamental “unit” of a test plan.

A test case mainly consists of…

- Preconditions
- Execution Steps
- Postconditions

See IEEE 829, "Standard for Software Test Documentation", for more details

Example

Assuming an empty shopping cart, when I click "Buy Widget", the number of widgets in the shopping cart becomes one.

Preconditions: Empty shopping cart
Execution Steps: Click "Buy Widget“
Postconditions: Shopping cart displays one widget

Example

Assuming that the SORT_ASCENDING flag is set, calling the sort method with [9,3,4,2] will return a new array sorted from high to low, i.e., [2,3,4,9].

Precondition: SORT_ASCENDING flag is set
Execution steps: Call sort method with argument [9,3,4,2]
Postconditions: [2,3,4,9] is returned

We also want to add:

- Identifier: A way to identify the test case
  - Could be a number
  - Often a label, e.g. INVALID-PASSWORD-THREE-TIMES-TEST
- Description: A description of the test case, describing what it is supposed to test.

Test Plan

- A collection of test cases for testing a system
- These do not always test an entire system
- They may test a subsystem or related piece of functionality
  - Examples:
  - Database Connectivity Test Plan
  - Pop-up Warning Test Plan
  - Pressure Safety Lock Test Plan
  - Regression Test Plan

Pressure Safety Lock Test Plan

LOW-PRESSURE-TEST
HIGH-PRESSURE-TEST
SAFETY-LIGHT-TEST
SAFETY-LIGHT-OFF-TEST
RESET-SWITCH-TEST
RESET-SWITCH2-TEST
FAST-MOVEMENT-TEST
RAPID-CHANGE-TEST
GRADUAL-CHANGE-TEST
MEDIAN-PRESSURE-TEST
LIGHT-FAILURE-TEST
SENSOR-FAILURE-TEST
SENSOR-INVALID-TEST

A group of test plans make up a test suite…

- Regression Test Suite
  - Pressure Safety Regression Test Plan
  - Power Regulation Regression Test Plan
  - Water Flow Regression Test Plan
  - Control Flow Test Plan
  - Security Regression Test Plan
  - Secondary Safety Process Test Plan

Test Run – Actual execution

- Test run: An actual execution of a test plan or test suite.
- Analogy time: class vs object, test plan vs test run
  - The test plan is the structure, but you need to actually execute
- During the test run, the tester manually (or automatically) executes each test case and sets the status

Possible Statuses

- PASSED – Completed with expected result
- FAILED – Completed but unexpected result
- PAUSED – Test paused in middle of execution
- RUNNING – Test in the middle of execution
- BLOCKED – Cannot be completed because precondition not fulfilled
- ERROR – Problem with running test itself

Defects

- If the test case fails, a defect should be filed
  - Unless the test case has already failed, of course.
  - You don’t need to re-file a duplicate of
- We will talk about filing defects on the next lecture

Creating a test plan…

- Start top-down: what is a good way to subdivide the system into features (test plans)?
- For a given feature (test plan), what aspects do I want to test?
- For each aspect, what test cases do I want that will hit different equivalence classes / success or failure cases / edge or corner cases / etc.?
- How deep should I go down?
- Test cases should be independent of each other, and reproducible!

Traceability Matrix

- Consider:
  - One test case may test multiple requirements
  - One requirement may be tested by multiple test cases
  - It’s a complex many-to-many relationship!

- **Traceability Matrix**: table that describes the relationship between requirements and test cases
  - Keeps track how requirements are enforced throughout software development
  - Can tell us where we are missing test coverage, or have superfluous tests

Good Traceability Matrix Example

REQ1: TEST_CASE_1, TEST_CASE_2

REQ2: TEST_CASE_3

REQ3: TEST_CASE_4, TEST_CASE_7

REQ4: TEST_CASE_5, TEST_CASE_9

REQ5: TEST_CASE_6, TEST_CASE_10

- *All requirements have at least one test case associated with them; all test cases map to a requirement.*

Problematic Traceability Matrix 1

REQ1: TEST_CASE_1, TEST_CASE_2

REQ2:

REQ3: TEST_CASE_4, TEST_CASE_7

REQ4: TEST_CASE_5, TEST_CASE_9

REQ5: TEST_CASE_6, TEST_CASE_10

- *No test case is testing requirement 2!*

Problematic Traceability Matrix 2

REQ1: TEST_CASE_1, TEST_CASE_2

REQ2: TEST_CASE_3

REQ3: TEST_CASE_4, TEST_CASE_7

REQ4: TEST_CASE_5, TEST_CASE_9

REQ5: TEST_CASE_6, TEST_CASE_10

?????: TEST_CASE_11

- *What is test case 11 checking?*

Traceability Matrix in Actual Matrix Format