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.

Test Plans and Test Cases

- Testing is done by executing a test plan
- *Test plan*: A list of related test cases that are run together.
- *Test case*: Smallest unit of a test plan that tests an individual behavior
  - Describes what is to be tested and how to test it
  - Describes expected behavior

A Test Case mainly consists of…

- *Preconditions*: State of the system before testing
  - Environment / global variable values, …
  - State of the screen, state of the database, …
- *Execution Steps*: Steps to obtain postconditions
- *Postconditions*: **Expected** state of the system after testing
  - Environment / global variables are set, …
  - Output printed to screen, network packet sent, …

See IEEE 829, "Standard for Software Test Documentation", at resources/IEEE829.pdf

Example

When shopping cart is empty, when I click "Buy Widget", the number of widgets in the shopping cart should become one.

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

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

- *Test Case*: A description of the test case

If doing method unit testing, we also add

- *Input values*: Values passed as method arguments
- *Output values*: **Expected** return value(s) from method
- Difference between input values and preconditions?
  - Everything other than arguments that impacts method is a precondition
    - Value of a global variable read by method
    - Contents of file read by method
- Difference between output values and postconditions?
  - Everything other than return value that method impacts is a postcondition
    - Value of a global variable modified by method
    - Contents of file modified by method

Example

When SORT_ASCENDING flag is set, calling the sort method with [9,3,4,2] should return a new array sorted from high to low: [2,3,4,9].

Preconditions: SORT_ASCENDING flag is set
Input values: Array [9,3,4,2]
Execution steps: Call sort method with input values
Output values: Array [2,3,4,9]
Postconditions: None

Another Example

```
int print_hello_world() {
    System.out.print("Hello World");
    return 1;
}
```

- Suppose you wanted to write a test case for above method:
  - What would be the output values?
  - What would be the postconditions?
  - Output values: 1
  - Postconditions: Hello World is printed

In full, a test case contains the following items

- Identifier
- Test Case
- Preconditions
- Input Values
- Execution Steps
- Output Values
- Postconditions

See IEEE 829, "Standard for Software Test Documentation", at resources/IEEE829.pdf

Test Plan

- List of test cases for a (sub)system or a feature

- Examples:
  - Database Connectivity Test Plan
  - Pop-up Warning Test Plan
  - Pressure Safety Lock Test Plan
  - System Shutdown 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 Regression Test Plan
  - Security Regression Test Plan

- *Regression*: A failure of a previously-working functionality caused by (seemingly) unrelated added functionality or defect fixes

Creating a test suite…

- Start top-down
  - What is a good way to subdivide 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 / edge or corner cases / etc.?
- Test cases should be independent of each other, and reproducible!
  - Independent: One test case shouldn’t be affected by the execution of another
  - Reproducible: Preconditions + execution steps always results in postconditions

Test Run – Actual execution

- *Test run*: An actual execution of a test plan or test suite.

- What if you find a test case is not independent?
  - Behavior changes depending on whether you run another test case
- What if you find a test case is not reproducible?
  - Behavior changes sometimes even with same input and steps

- Most likely both happen due to not having set up preconditions
  - Variations in OS state, DB state, filesystem state, memory state, …

Test Fixture – Preconditions Setup Script

- A test plan may rely on certain preconditions
  - If so, must set it up before running the test plan
  - If a test case changes preconditions,
    must set it up again in the middle of a test run
  - Setting up preconditions is often tedious
- *Test fixture* – script to set up preconditions
  - May involve (re)installing software on a machine
  - Populating database with data for testing
  - Populating memory with data
  - Reinitializing hardware devices in the system

Status after Test Run

- During test run, tester manually (or automatically) executes each test case and sets the status for each

- 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
- We will talk about filing defects on the next lecture

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!
  - How do you keep track of what’s being tested and by how much?

- **Traceability Matrix**: table that describes the relationship between requirements and test cases
  - 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

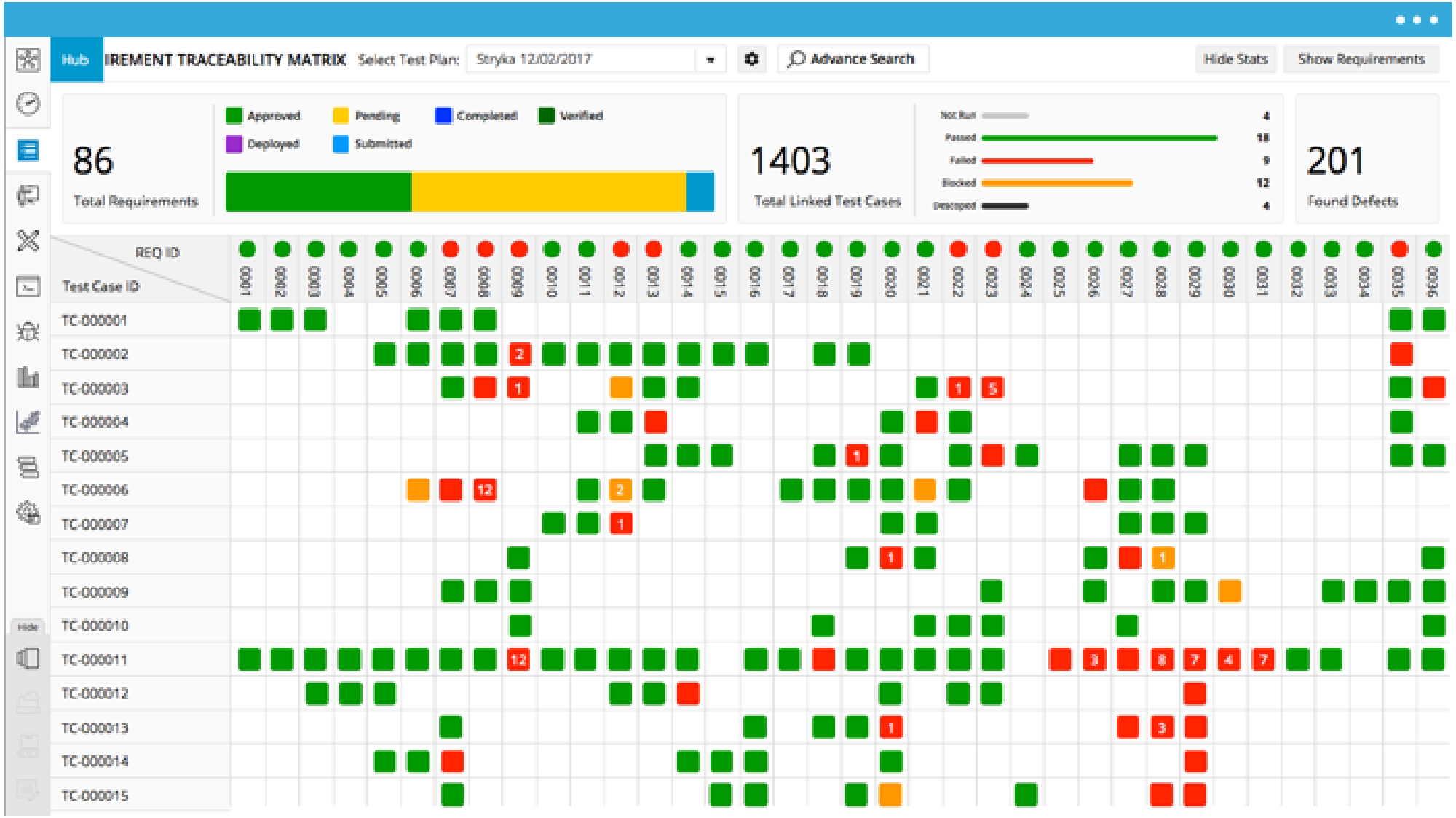
REQ4: TEST_CASE_5, TEST_CASE_9

REQ5: TEST_CASE_6, TEST_CASE_10

?????: TEST_CASE_11

- *What is test case 11 checking?*

Traceability Matrix in Actual Matrix Format



Now Please Read Textbook Chapters 6 and 8

- In particular, read Chapter 8 carefully since that's mostly what you will be doing for our first in-class exercise next week.

- If you are interested in further reading:

  IEEE Standard for Software Test Documentation (IEEE 829-2008)

- Can be found in resources/IEEE829.pdf in course repository