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
  - How critical is the software that you are testing?
  - How many times is the test plan going to be used?

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
  - You can think of one input value as one test case
  - 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 perform test

- *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
  - e.g. “When shopping cart is empty, when I click Buy Widget, the number of widgets in the shopping cart should become one.”

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 (E.g. value of a global variable, contents of file read by method)
- Difference between output values and postconditions?
  - Everything other than return value that method impacts is a postcondition (E.g. value of a global variable, 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 low to high: [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
  - System Shutdown Test Plan
  - Pressure Safety Lock 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 enhancements or defect fixes
  - Regression tests are run regularly to check entire system on code updates

Creating a test suite…

- Start top-down

- Subdivide system into features or subsystems
  - Create a test plan for each of those features
- For each feature, decide on what aspects to test
- For each aspect, decide on which inputs or user interactions to test
  - Create a test case for each input, under the subsystem test plan
  - Hit different base, edge, and corner cases for good test coverage

Test Run – Actual execution

- *Test run*: Actual execution of a test case / test plan / test suite
  - Subsets of test cases may be chosen to run or the entire test suite
  - All depends on the type of code modification and the testing context

- The purpose of a test run is to obtain *observed behavior*
  - Passes or fails after comparing *observed behavior* with *postcondition*

Status after Test Run

- 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: Did not complete because precondition not fulfilled
  - ERROR: Problem with running test itself

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

Defects

- If the test case fails, a defect should be filed
  - Unless the defect has already been filed, of course
  - You don’t need to re-file a duplicate

- We will talk about filing defects on the next lecture

Creating Good Test Cases

- Besides good test coverage, what else makes a good test case?

- A good test case as the following additional characteristics:
  - Reproducibility
  - Independence

A Good Test Case is Reproducible

- Preconditions + Execution Steps always result in same behavior
- What happens when a test case is unreproducible?
  - Defect found by test may not manifest when developer tries to debug it
  - Test does not find defect but defect manifests when software is deployed
- What causes a test case to be unreproducible?
  - Incomplete preconditions (OS state, DB state, filesystem state, memory state)
    - E.g. OS environment variable that impacts test case is not specified
    - E.g. A configuration file that impacts test case is not specified
    - E.g. Java compiler version that impacts test case is not specified
  - Imprecise execution steps
    - E.g. “Open new browser window” → Multiple ways: Ctrl+N, Menu, Icon double click

A Good Test Case is Independent

- Test case shouldn’t depend on the execution of a previous test case
    - E.g. Should not depend on database entries inserted by previous test case
- What happens when a test case is dependent?
    - Test cases may be executed selectively, causing previous case to not execute
    - Test cases may execute out of order, causing previous case to execute later (Often, test cases are run in parallel to save testing time)
- What causes a test case to be dependent?
    - Again, incomplete preconditions
    - You are relying on previous test case to fulfill part of the preconditions

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!*
- *Likely leads to low test coverage.*

Problematic Traceability Matrix 2

REQ1: TEST_CASE_1, TEST_CASE_2

REQ2: TEST_CASE_3

REQ3: TEST_CASE_4, TEST_CASE_7

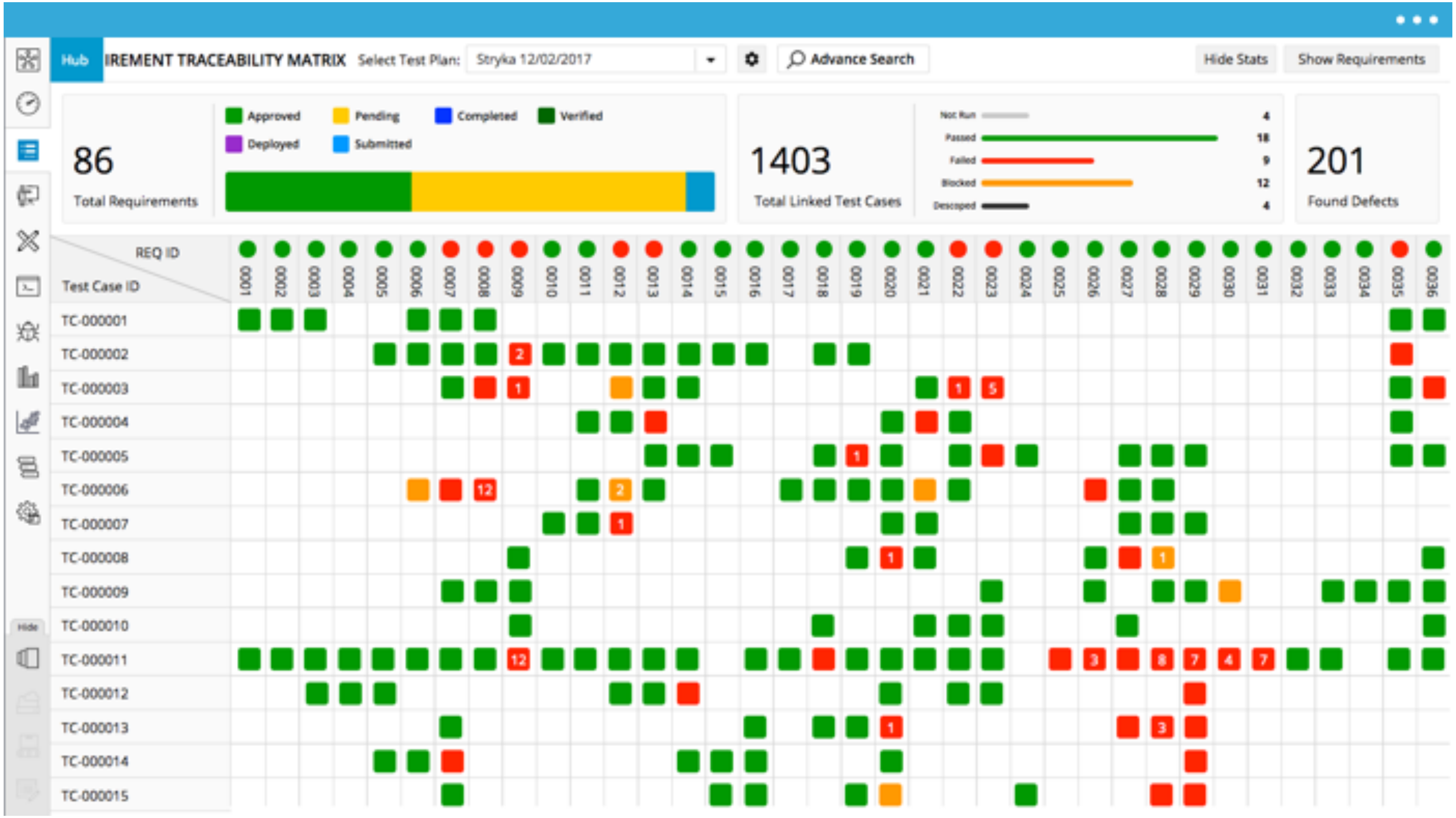
REQ4: TEST_CASE_5, TEST_CASE_9

REQ5: TEST_CASE_6, TEST_CASE_10

?????: TEST_CASE_11

- *Test case 11 not checking any requirement → Remove!*

Traceability Matrix in Actual Matrix Format



Hub IREMENT TRACEABILITY MATRIX Select Test Plan: Stryka 12/02/2017

86
Total Requirements

🟩	Approved
🟨	Pending
🟦	Completed
🟩	Verified
🟪	Deployed
🟦	Submitted

1403
Total Linked Test Cases

Not Run	4
Passed	18
Failed	9
Blocked	12
Descoped	4

201
Found Defects

REQ ID Test Case ID	🟢 0001	🟢 0002	🟢 0003	🟢 0004	🟢 0005	🟢 0006	🔴 0007	🔴 0008	🔴 0009	🟢 0010	🟢 0011	🔴 0012	🔴 0013	🟢 0014	🟢 0015	🟢 0016	🟢 0017	🟢 0018	🟢 0019	🟢 0020	🟢 0021	🔴 0022	🔴 0023	🟢 0024	🟢 0025	🟢 0026	🟢 0027	🟢 0028	🟢 0029	🟢 0030	🟢 0031	🟢 0032	🟢 0033	🟢 0034	🔴 0035	🟢 0036
TC-000001	🟩	🟩	🟩			🟩	🟩	🟩																											🟩	🟩
TC-000002					🟩	🟩	🟩	🟩	🟥 2	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩																🟥	
TC-000003							🟩	🟥	🟥 1			🟧	🟩	🟩							🟩	🟥 1	🟥 5												🟩	🟥
TC-000004											🟩	🟩	🟥							🟩	🟥	🟩													🟩	
TC-000005													🟩	🟩	🟩			🟩	🟥 1	🟩		🟩	🟥	🟩			🟩	🟩	🟩						🟩	🟩
TC-000006						🟧	🟥	🟥 12			🟩	🟧 2	🟩				🟩	🟩	🟩	🟩	🟧	🟩				🟥	🟩	🟩								
TC-000007										🟩	🟩	🟥 1								🟩	🟩						🟩	🟩	🟩							
TC-000008									🟩										🟩	🟥 1	🟩					🟩	🟥	🟧 1								🟩
TC-000009							🟩	🟩	🟩														🟩			🟩		🟩	🟩	🟧			🟩	🟩	🟩	🟩
TC-000010									🟩									🟩			🟩	🟩	🟩				🟩									🟩
TC-000011	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥 12	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟥	🟩	🟩	🟩	🟩	🟩		🟥	🟥 3	🟥	🟥 8	🟥 7	🟥 4	🟥 7	🟩	🟩		🟩	🟩
TC-000012			🟩	🟩	🟩							🟩	🟩	🟥						🟩		🟩	🟩						🟥							
TC-000013							🟩									🟩		🟩	🟩	🟥 1							🟥	🟥 3	🟥							
TC-000014					🟩	🟩	🟥							🟩	🟩	🟩				🟩									🟥							
TC-000015							🟩								🟩	🟩			🟩	🟧				🟩				🟥	🟥							

Reference: reportportal.io

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