Software Testing

CENG 316 – Software Engineering

22 March 2022, @IZTECH

Agenda

- Verification & Validation
- Software testing
- Defect, Infection, Failure
- Test case
- Objectives of testing
- Complete testing
- White-box & Black-box testing
- Unit, Integration, & System testing
- Acceptance testing
- Alpha & Beta testing
- Regression testing
- Testing types
- Mock objects
- Test-driven Development
- Use case testing
- Test coverage

Validation & Verification

Validation

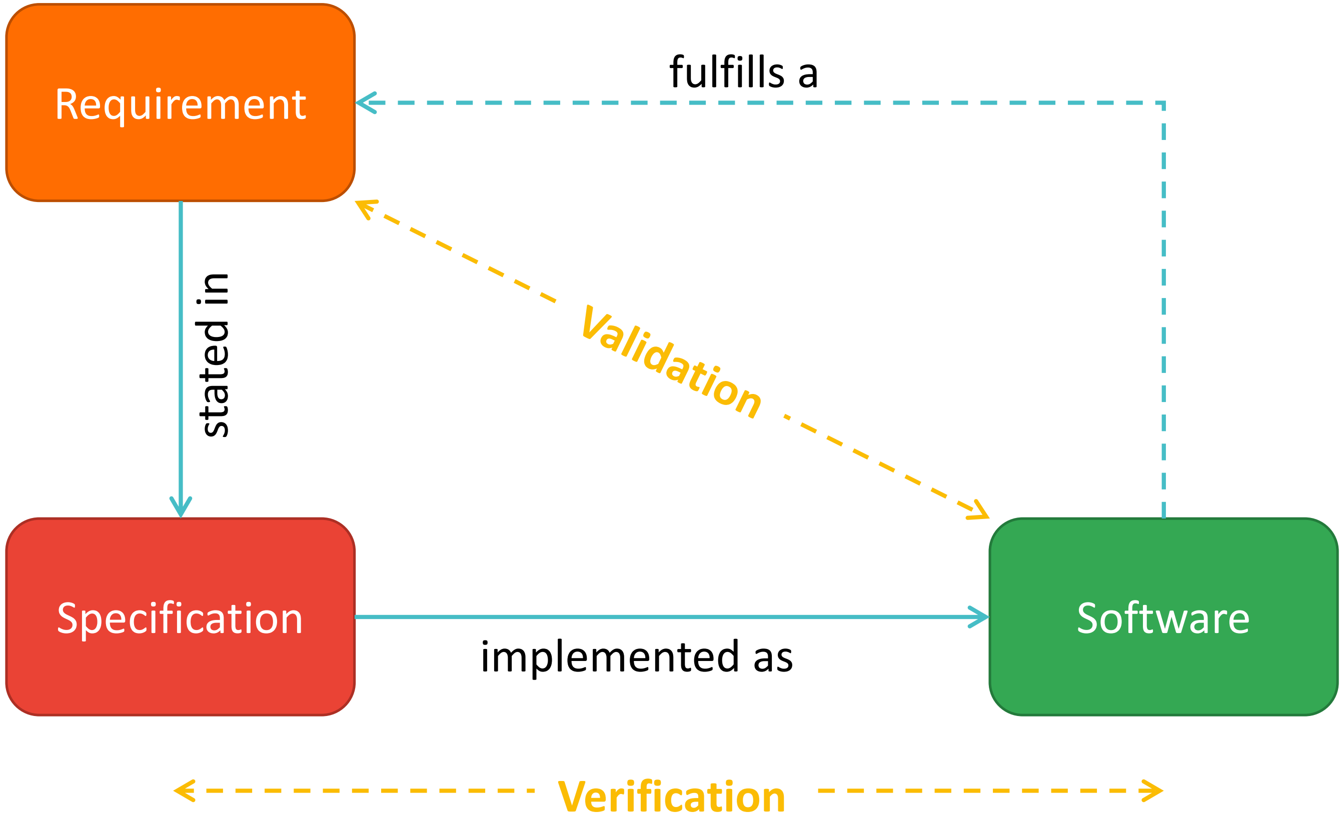
- Are we building the **right product**?
- The software should do what the **user really requires**.

Verification

- Are we building the **product right**?
- The software should **conform to its specification**.

Ref: [Sommerville]

Validation & Verification



Verification Activities & Testing

Verification Activities

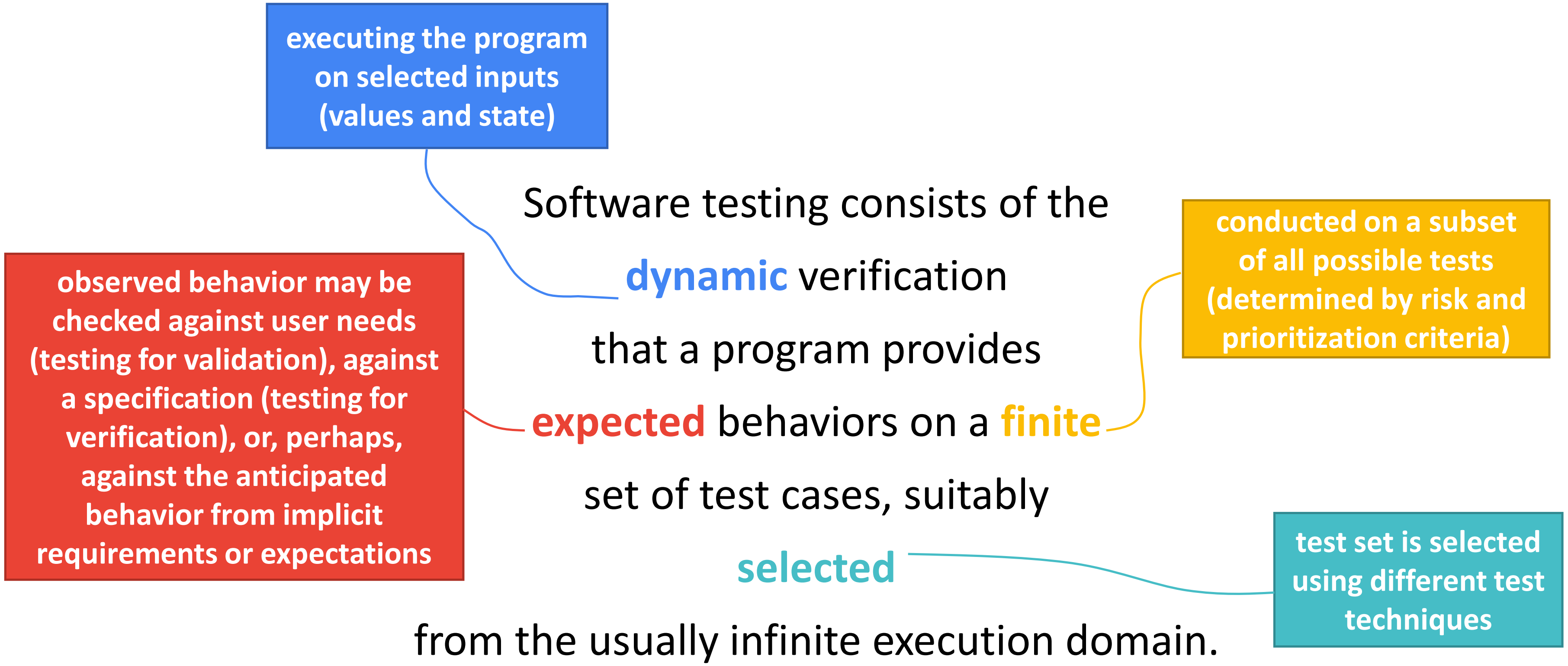
- testing without executing the program
- includes reviews, walkthroughs, inspections and some forms of analysis
- also known as "static testing", "static analysis"

Testing

- testing by executing the program with inputs
- also known as "dynamic testing", "dynamic analysis"

Ref: [Ammann], [Naik]

Software Testing - Definition



Software testing consists of the **dynamic** verification that a program provides **expected** behaviors on a **finite** set of test cases, suitably **selected** from the usually infinite execution domain.

executing the program on selected inputs (values and state)

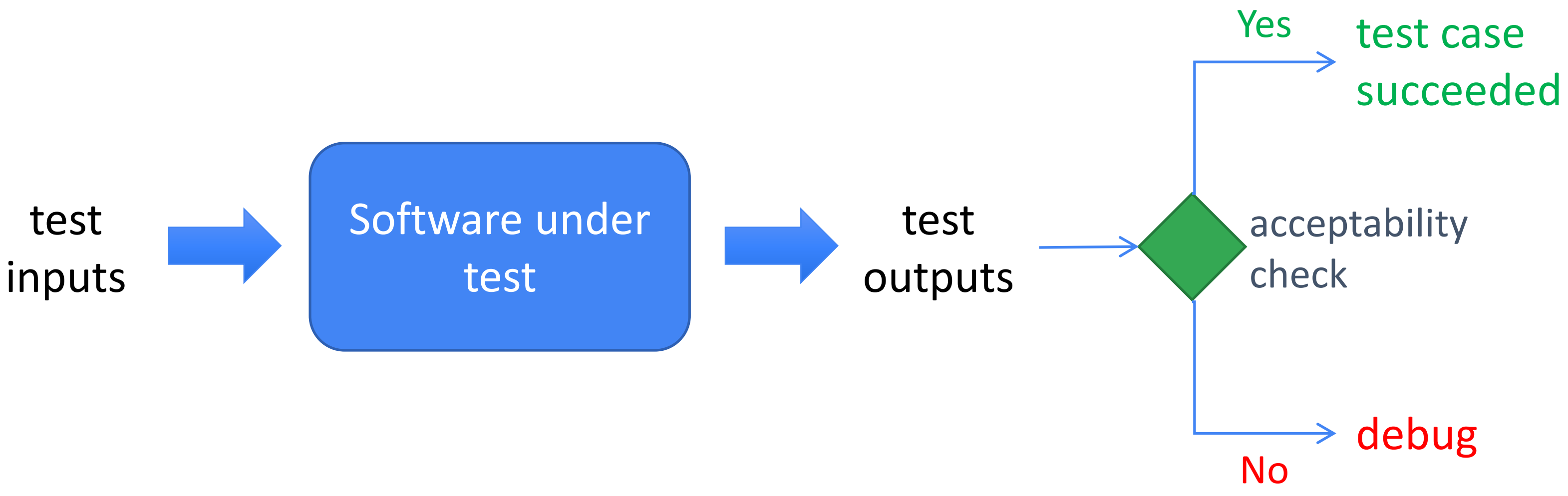
observed behavior may be checked against user needs (testing for validation), against a specification (testing for verification), or, perhaps, against the anticipated behavior from implicit requirements or expectations

conducted on a subset of all possible tests (determined by risk and prioritization criteria)

test set is selected using different test techniques

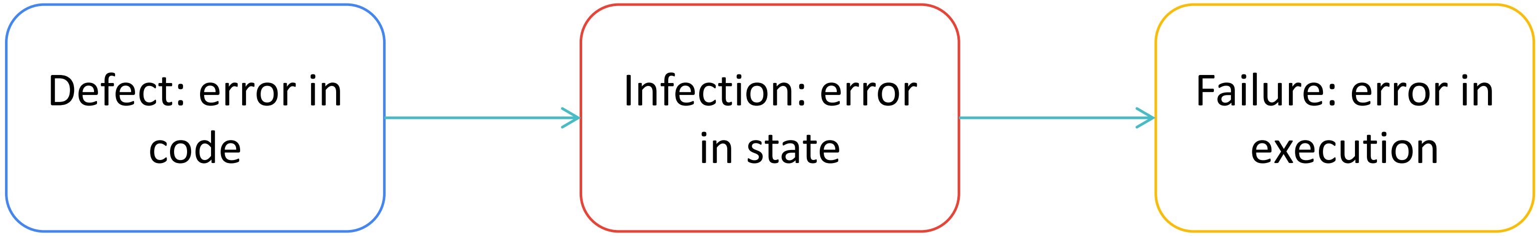
Ref: [SWEBOK]

What is Testing?



Ref: [Regehr]

Defect, Infection, Failure



Defect: error in code

Infection: error in state

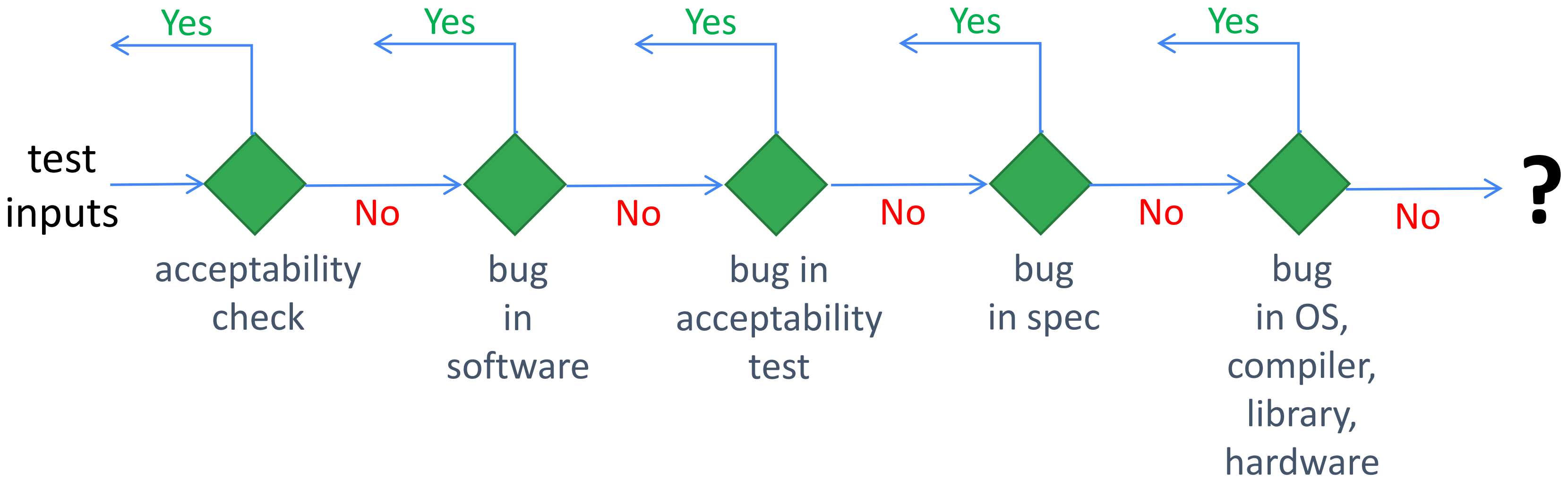
Failure: error in execution

also called bug or fault (?)

observable incorrect behavior

Ref: [Zeller]; [Orso]

What is going on when we test?



Ref: [Regehr]

Test Case

a simple pair of <input, expected outcome>

#	Step	Explanation
1	< check balance, $500.00 >	assume that the user account has $500.00 on it
2	< withdraw, «amount?» >	the user wants to withdraw an amount
3	< $200.00, «$200.00» >	the withdraw amount is $200.00
4	< check balance, $300.00 >	after the withdrawal operation, the user account should have $300.00 on it

Ref: [Naik]

Expected Outcome

- Values produced by the software
    - Outputs for local observation (integer, text, audio, image)
    - Outputs (messages) for remote storage, manipulation, or observation
- State change
    - State change of the software
    - State change of the database (due to insert, delete, and update operations)
- A sequence or set of values which must be interpreted together for the outcome to be valid

Ref: [Naik]

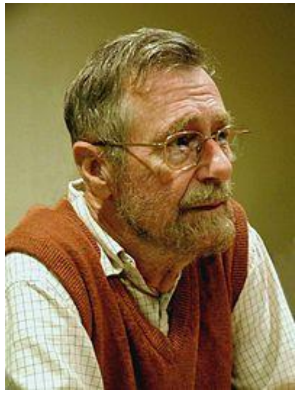
Objectives of Testing

- demonstrating that errors are not present → **NO**
- executing a program with the intent of finding errors → **YES**
- reduce the risk of failure → **YES**
- reduce the cost of testing → **YES**

Ref: [Myers]; [Naik]

«Testing can only show the presence of errors, not their absence»

Edsger W. Dijkstra



Ref: [Zeller]

Complete (Exhaustive) Testing

Definition: There are no undiscovered defects at the end of the test phase

Is it possible?→ **NO**

- Number of possible inputs (valid and invalid)
- Number of states
- Timing constraints on the inputs
- Design issues (too complex)
- Execution environment

Ref: [Naik]

Complete (Exhaustive) Testing Example

How long would it take to exhaustively test the function

`printSum(int a, int b)`?

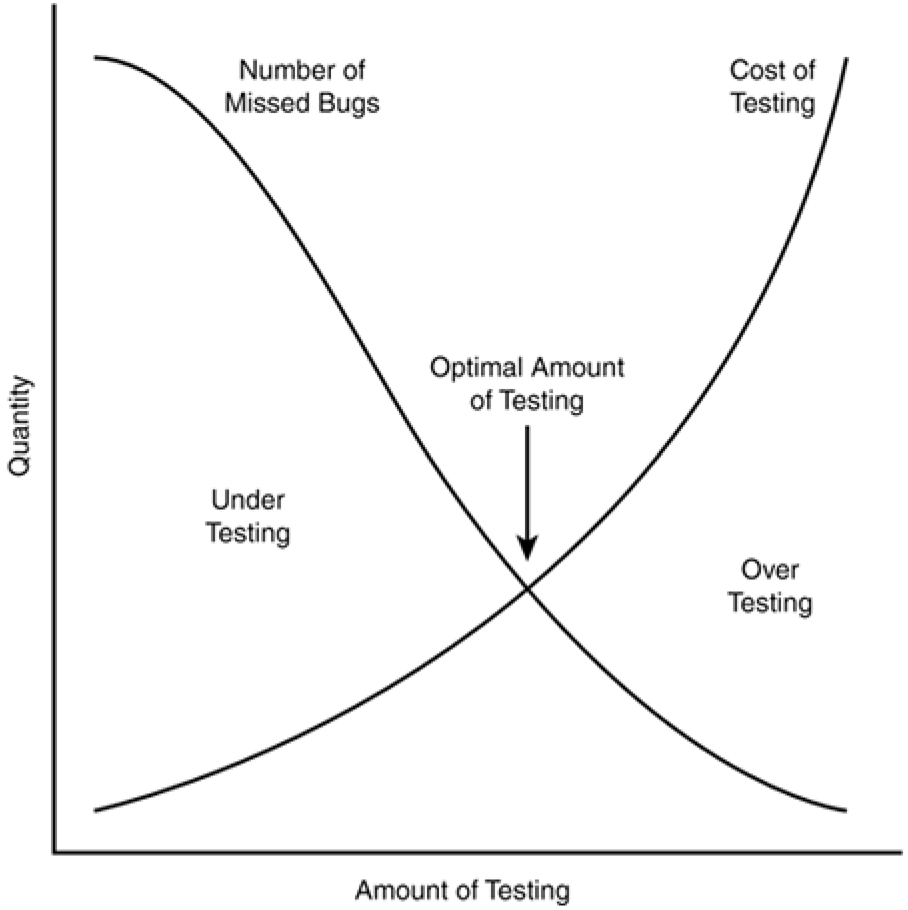
2^{32} x 2^{32} = 2^{64} ≈ 10^{19} tests

1 test per nanosecond (10^{9} tests/sec)

10^{10} seconds overall ≈ 600 years

Ref: [Orso]

Optimal Amount of Testing



Ref: [Patton]

White-box & Black-box Testing

White-box testing

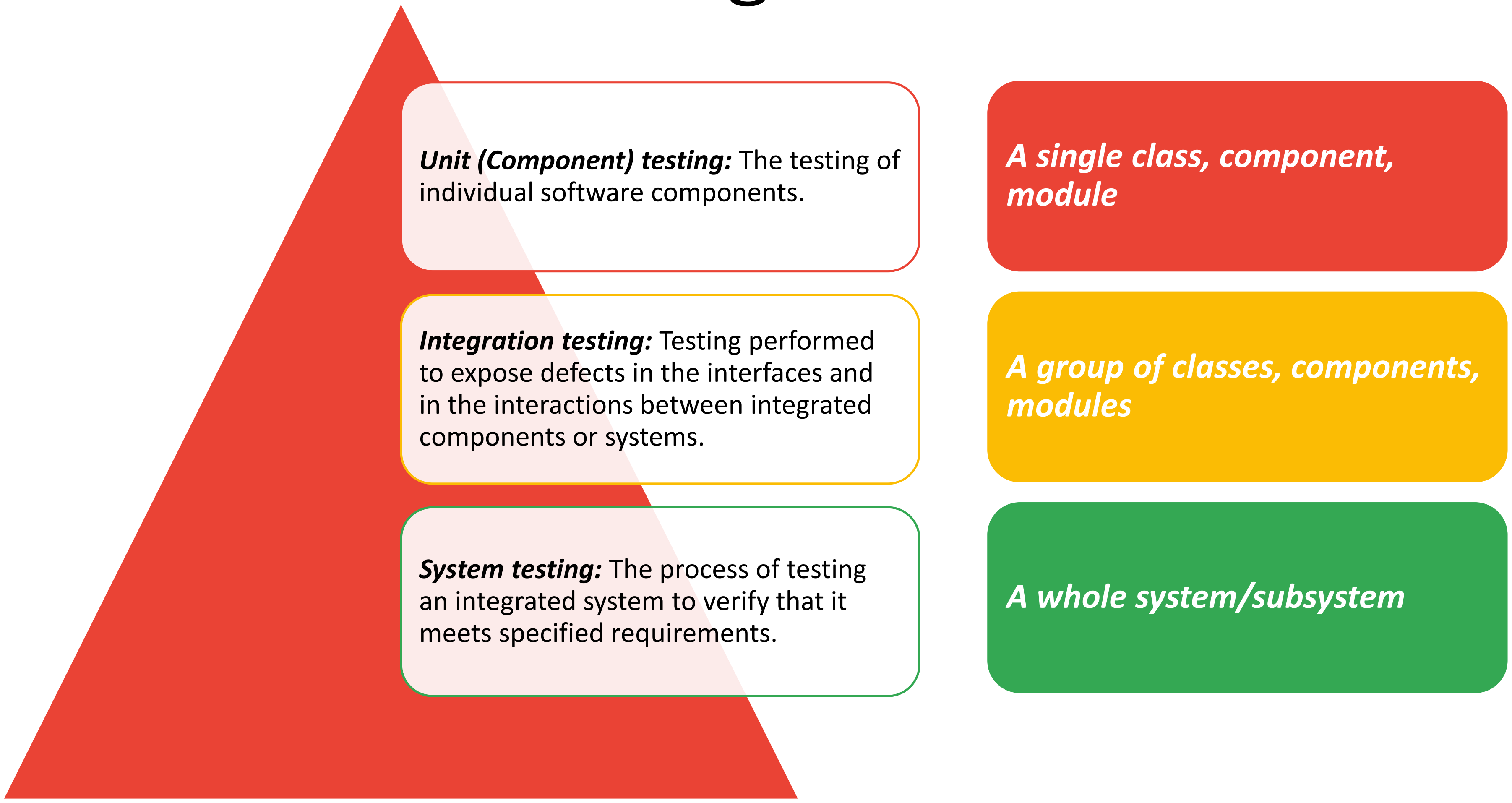
- also known as "structural testing"
- based on code
- cannot reveal defects due to missing requirements

Black-box testing

- also known as "functional testing"
- based on specification
- cannot reveal defects due to implementation defects

Ref: [Orso]

Test Levels: Unit, Integration, & System Testing



Unit (Component) testing: The testing of individual software components.

A single class, component, module

Integration testing: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.

A group of classes, components, modules

System testing: The process of testing an integrated system to verify that it meets specified requirements.

A whole system/subsystem

Ref: [ISTQB Glossary], [SWEBOK]

Objectives of Unit, Integration, & System Testing

Unit Testing

- verify that each distinct execution of a program unit produces the expected result
- errors found during testing can be attributed to a specific unit so that it can be easily fixed
- prove that a piece of code does what the developer thinks it should do
- does NOT prove that the piece of code does what the customer wants (Acceptance testing proves that)

Integration Testing

- build a “working” version of the system **incrementally** by ensuring that the additional modules work as expected without disturbing the functionalities of the modules already put together

System Testing

- verify the behavior of the whole system/product as defined by the scope of a development project or product

Ref: [Naik], [Hunt], [Graham]

Unit, Integration, and System Testing

	Unit Test	Integration Test	System Test
Test cases derived from	module specifications	architecture and design specifications	requirements specification
Visibility required	all the details of the code	some details of the code, mainly interfaces	no details of the code
Scaffolding required	Potentially complex, to simulate the activation environment (drivers), the modules called by the module under test (stubs) and test oracles	Depends on architecture and integration order. Modules and subsystems can be incrementally integrated to reduce need for drivers and stubs.	Mostly limited to test oracles, since the whole system should not require additional drivers or stubs to be executed. Sometimes includes a simulated execution environment (e.g., for embedded systems).
Focus on	behavior of individual modules	module integration and interaction	system functionality

Ref: [Pezze]

Acceptance Testing

Definition

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

Objectives

- establish confidence in the system, part of the system or specific non-functional characteristics, e.g. usability, of the system
- often focused on a validation type of testing

Ref: [ISTQB Glossary], [Graham]

Alpha and Beta Testing

Alpha testing

- takes place at the developer’s site
- done by an internal team independent of the development team before release to external customers

Beta testing

- takes place at the customer’s site
- done by a group of customers, who use the product at their own locations and provide feedback, before the system is released
- often called “field testing”

Ref: [Hambling]

Regression Testing

Definition

- done whenever the software changes, either as a result of fixes or new or changed functionality
- involves executing test cases that have been executed and passed before

Objectives

- verify that modifications in the software or the environment have not caused unintended adverse side effects and that the system still meets its requirements

Ref: [Graham]

Testing Types

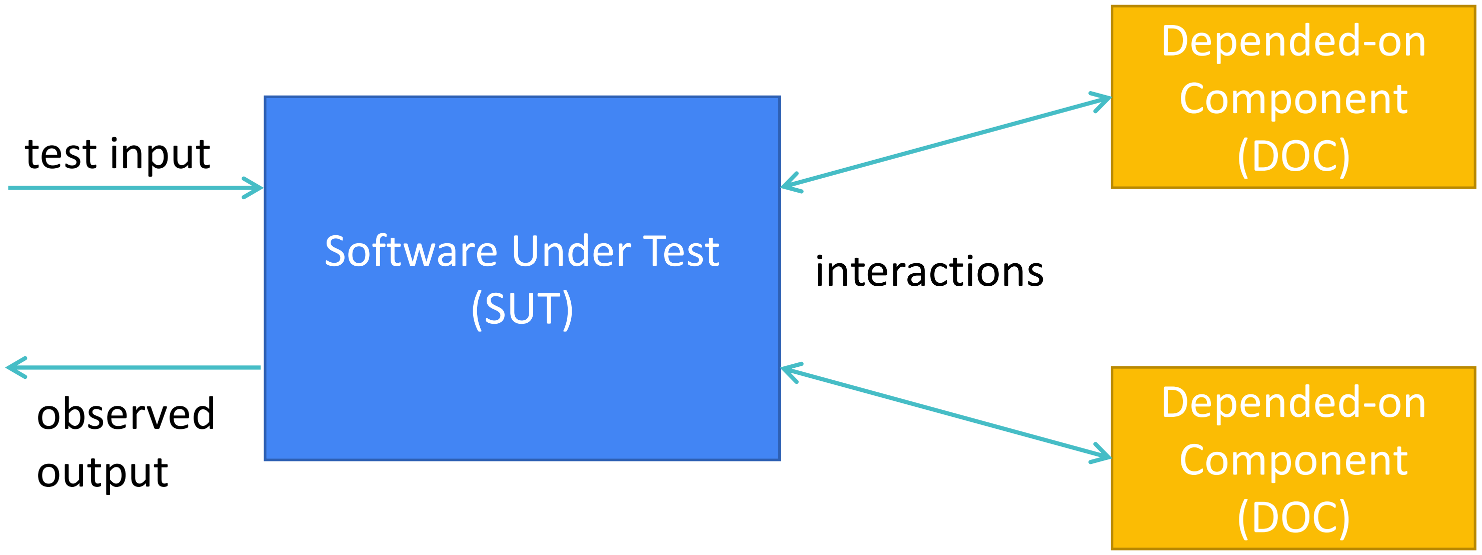
Functional requirements	Functional testing	• based on an analysis of the specification of the functionality of a component or system
Non-functional requirements (Quality attributes)	Security Testing	• Is the system and its data secure against accidental or malicious damage?
	Performance Testing	• Response time to user interaction • Time to complete specified operations
	Load Testing	• a type of performance testing • evaluate the behavior of a component or system with increasing load, e.g. numbers of parallel users and/or numbers of transactions, to determine what load can be handled by the component or system
	Installability Testing	• testing the installability of a software product in a specified environment
	Other types of testing …	• …

Ref: [ISTQB Glossary]

Problem: Testing a group of dependent components

When a component/class depends on other components/classes

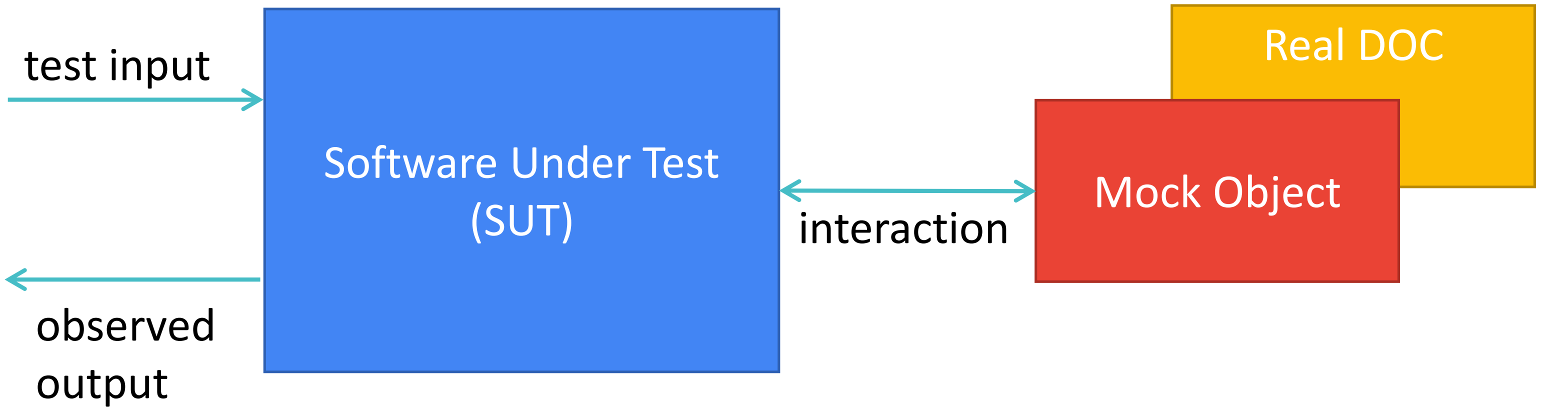
- Option 1: test it together with all the other classes it depends on
- Option 2: isolate it from the other classes so that we can test it by itself


test input
Software Under Test (SUT)
observed output
interactions
Depended-on Component (DOC)
Depended-on Component (DOC)

Ref: [Meszaros]

Solution: Mock Objects

replace the real DOC with a mock object that we can control



```
class Real_DOC
{
    function get_name(long id)
    {
        call remote function
        parse response
        return name
    }
}
```

```
class Mock_Object
{
    function get_name(long id)
    {
        return "a name"
    }
}
```

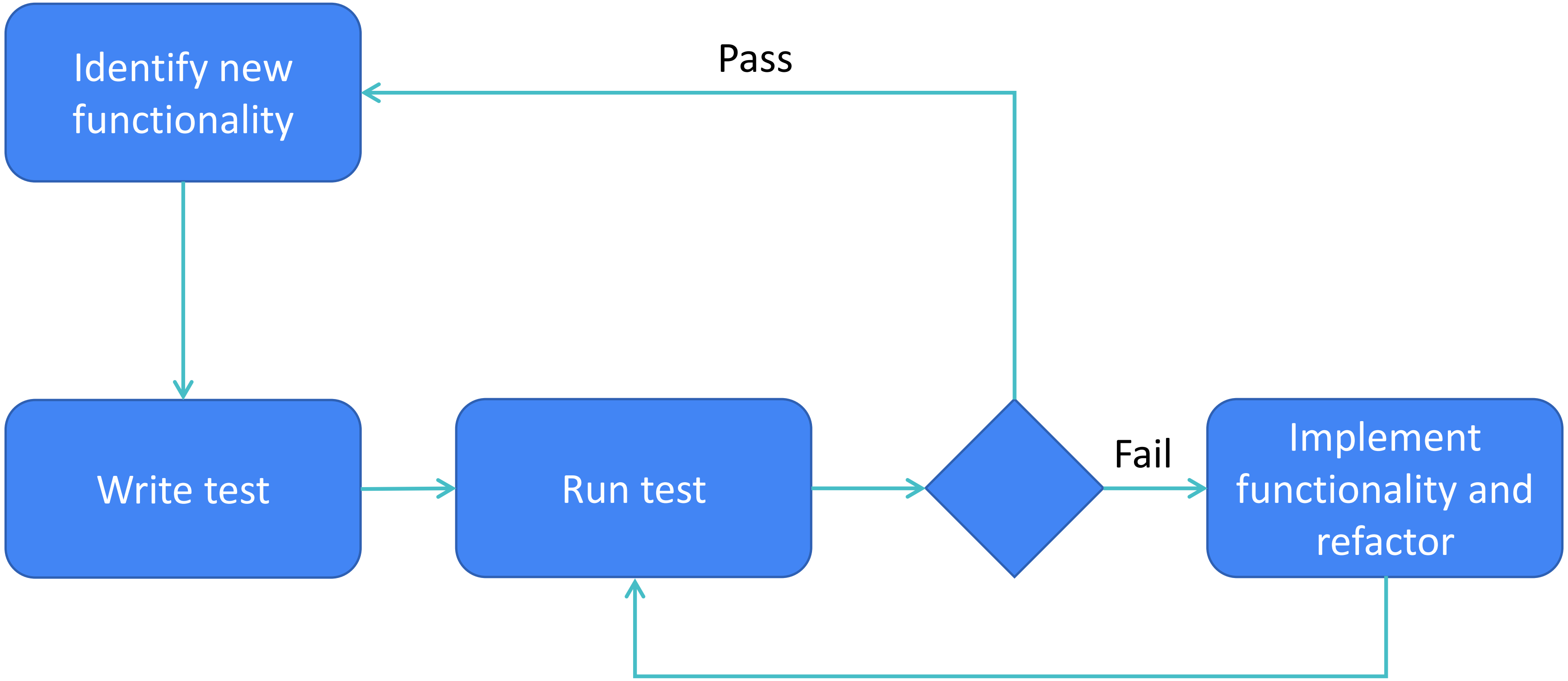
Ref: [Meszaros]

Test-driven Development (TDD)

- an approach to software development
- was introduced as part of agile methods such as XP
- develop code incrementally, along with a test for that increment
- do not move on to the next increment until the code passes its test

Ref: [Sommerville]

TDD Process



Identify new functionality
Pass
Write test
Run test
Fail
Implement functionality and refactor

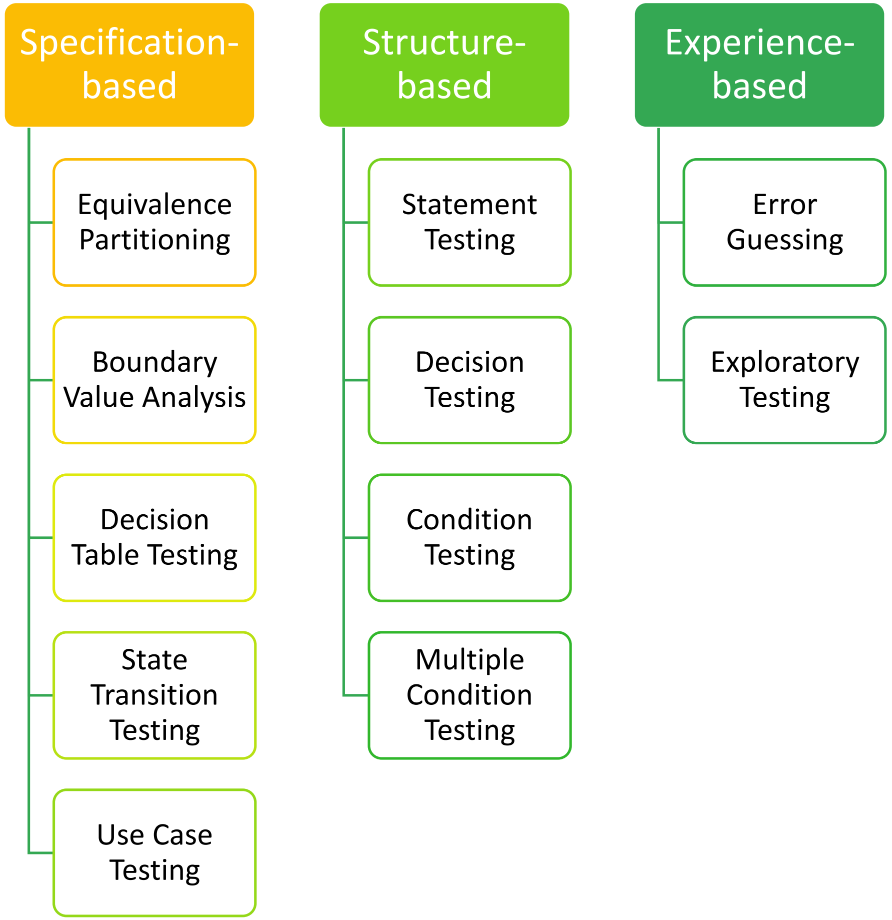
Ref: [Sommerville]

TDD Benefits

- Better problem understanding
- Code coverage
- Regression testing
- Simplified debugging
- System documentation

Ref: [Sommerville]

Test Design Techniques



Ref: [ISTQB FLS]

Use Case Testing

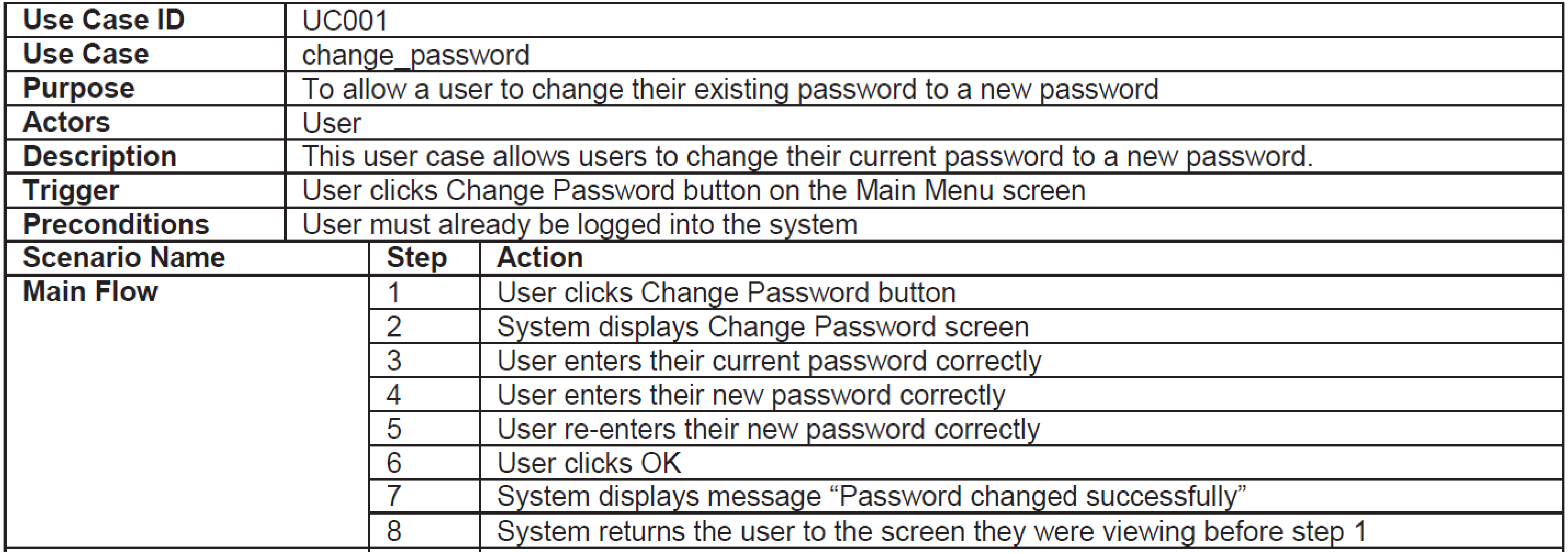
a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

#	Input	Expected Output	Use Case Scenario Tested
TC_1	• insert valid card • valid PIN	• account accessed	Main Success Scenario Steps: 1-5
TC_2	• insert invalid card	• "Card not valid" message displayed • card rejected	Extension Steps: 1, 2a
TC_3	• insert valid card • invalid PIN	• "PIN not valid" message displayed • asked for re-entering PIN	Extension Steps: 1, 2, 3, 4, 4a
TC_4	• insert valid card • invalid PIN (3 times)	• "PIN invalid 3 times" message displayed • card eaten	Extension Steps: 1, 2, 3, 4, 4b

Ref: [Graham]

An Example: change_password use case



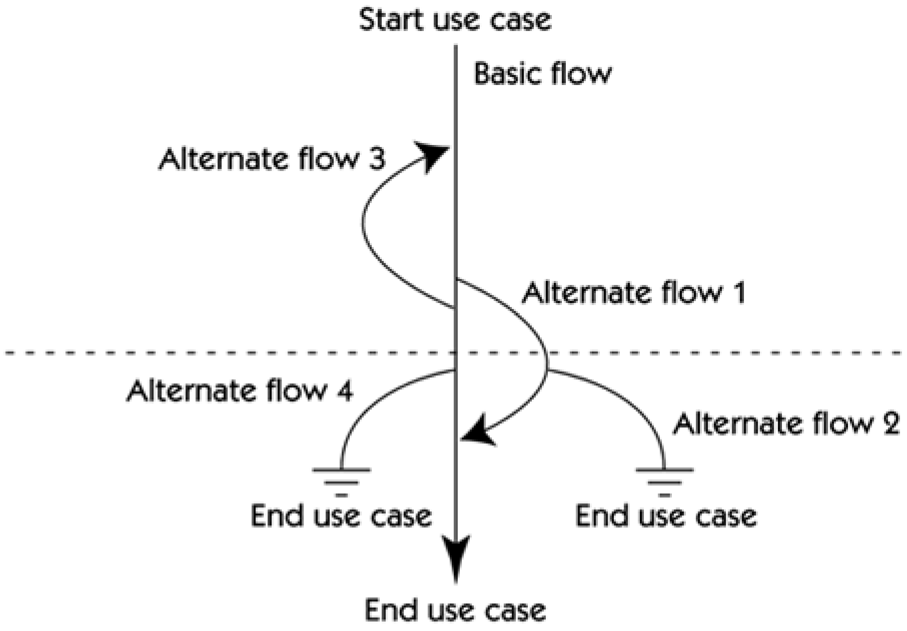
Use Case ID	UC001
Use Case	change_password
Purpose	To allow a user to change their existing password to a new password
Actors	User
Description	This user case allows users to change their current password to a new password.
Trigger	User clicks Change Password button on the Main Menu screen
Preconditions	User must already be logged into the system

Scenario Name	**Step**	**Action**
Main Flow	1	User clicks Change Password button
	2	System displays Change Password screen
	3	User enters their current password correctly
	4	User enters their new password correctly
	5	User re-enters their new password correctly
	6	User clicks OK
	7	System displays message “Password changed successfully”
	8	System returns the user to the screen they were viewing before step 1

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

An Example: Typical & Alternate Scenarios in change_password use case

Flow 1	Typical Flow
Flow 2	Alternate Flow – Existing Password Incorrect
Flow 3	Alternate Flow – New Password Less Than 8 Characters
Flow 4	Alternate Flow – New Password Same as Current Password
Flow 5	Alternate Flow – New Passwords Do Not Match



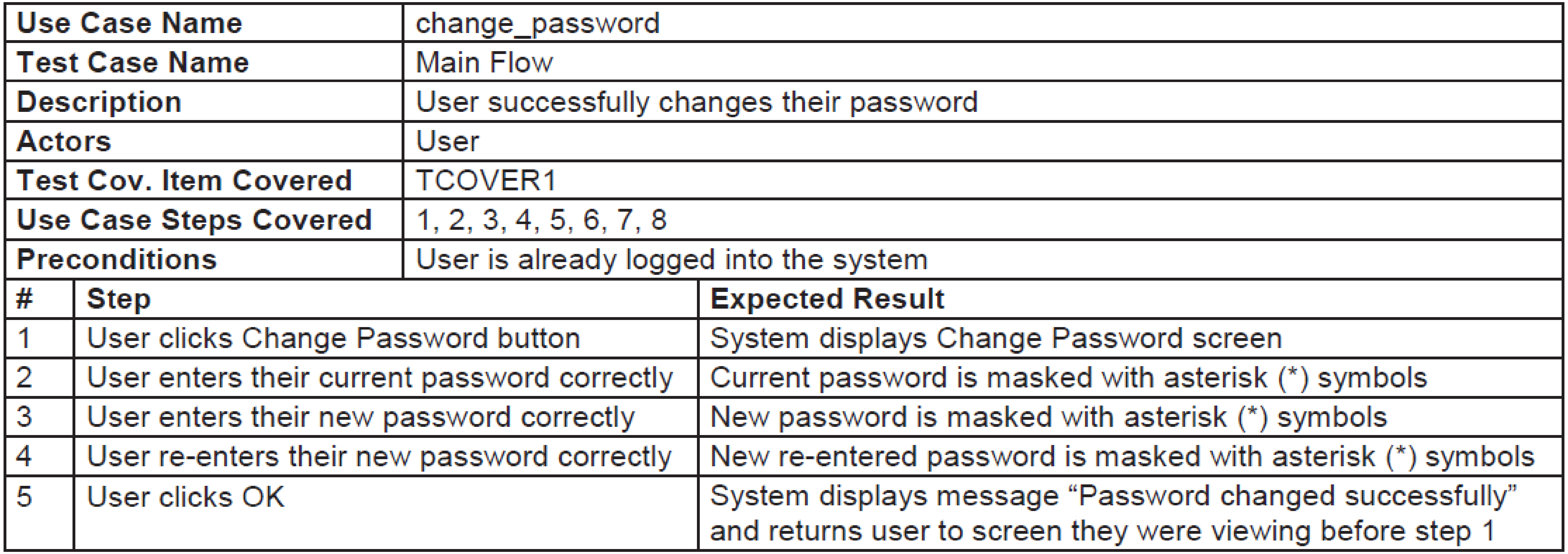
Ref:[ISO/IEC/IEEE P29119-4-DISMay2013], [Leffingwell]

Example: Derive Test Cases for change_password use case

Test cases are derived by **selecting a scenario** to cover, identifying **input**s to exercise the path covered by the test case, determining the **expected result** and **repeating until all use case scenarios are covered as required**.

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Example: Test case for Flow 1

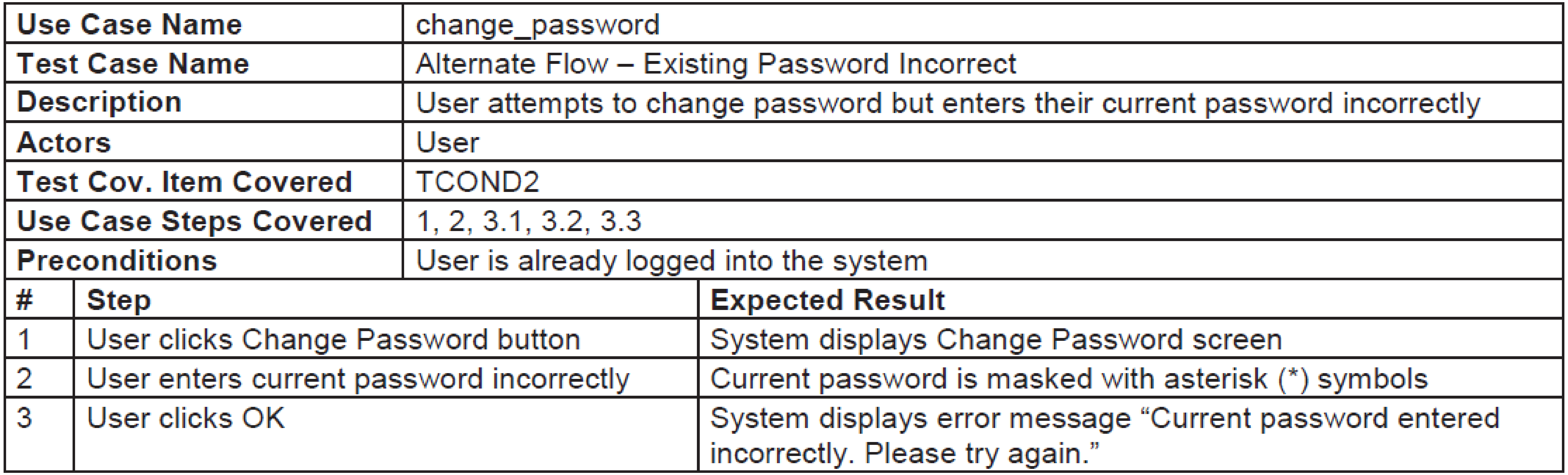


Use Case Name	change_password
Test Case Name	Main Flow
Description	User successfully changes their password
Actors	User
Test Cov. Item Covered	TCOVER1
Use Case Steps Covered	1, 2, 3, 4, 5, 6, 7, 8
Preconditions	User is already logged into the system

#	**Step**	**Expected Result**
1	User clicks Change Password button	System displays Change Password screen
2	User enters their current password correctly	Current password is masked with asterisk (*) symbols
3	User enters their new password correctly	New password is masked with asterisk (*) symbols
4	User re-enters their new password correctly	New re-entered password is masked with asterisk (*) symbols
5	User clicks OK	System displays message “Password changed successfully” and returns user to screen they were viewing before step 1

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Example: Test case for Flow 2

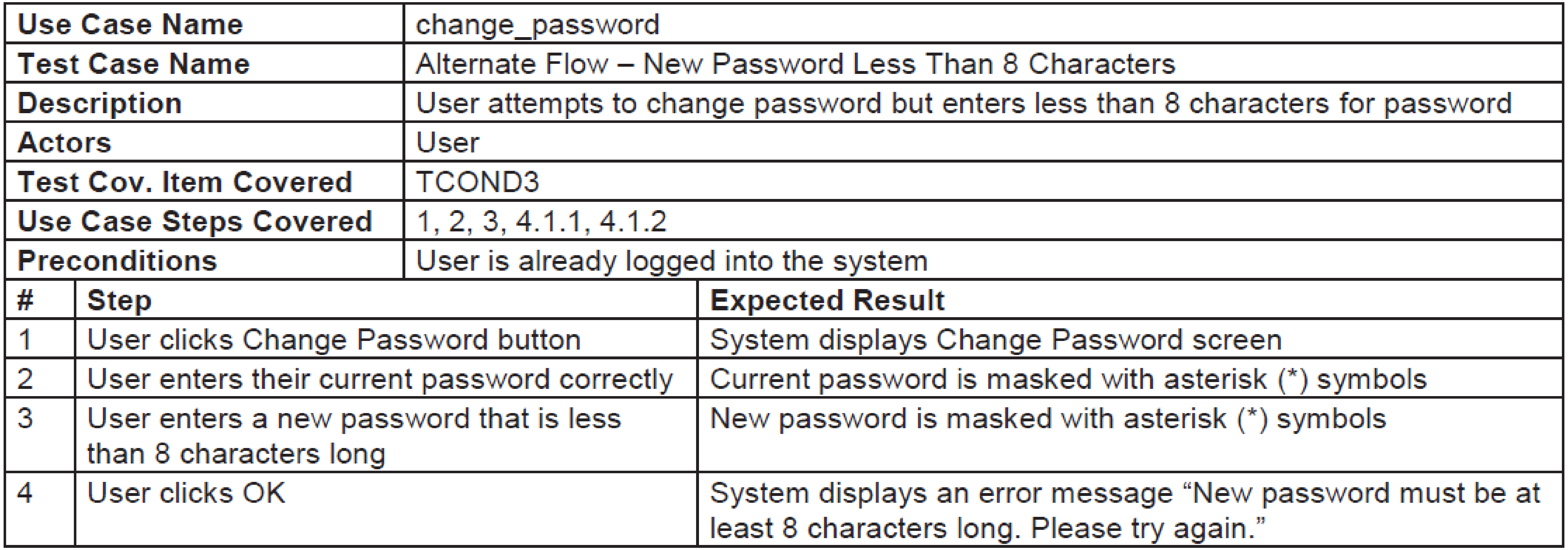


Use Case Name	change_password
Test Case Name	Alternate Flow – Existing Password Incorrect
Description	User attempts to change password but enters their current password incorrectly
Actors	User
Test Cov. Item Covered	TCOND2
Use Case Steps Covered	1, 2, 3.1, 3.2, 3.3
Preconditions	User is already logged into the system

#	**Step**	**Expected Result**
1	User clicks Change Password button	System displays Change Password screen
2	User enters current password incorrectly	Current password is masked with asterisk (*) symbols
3	User clicks OK	System displays error message “Current password entered incorrectly. Please try again.”

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Example: Test case for Flow 3

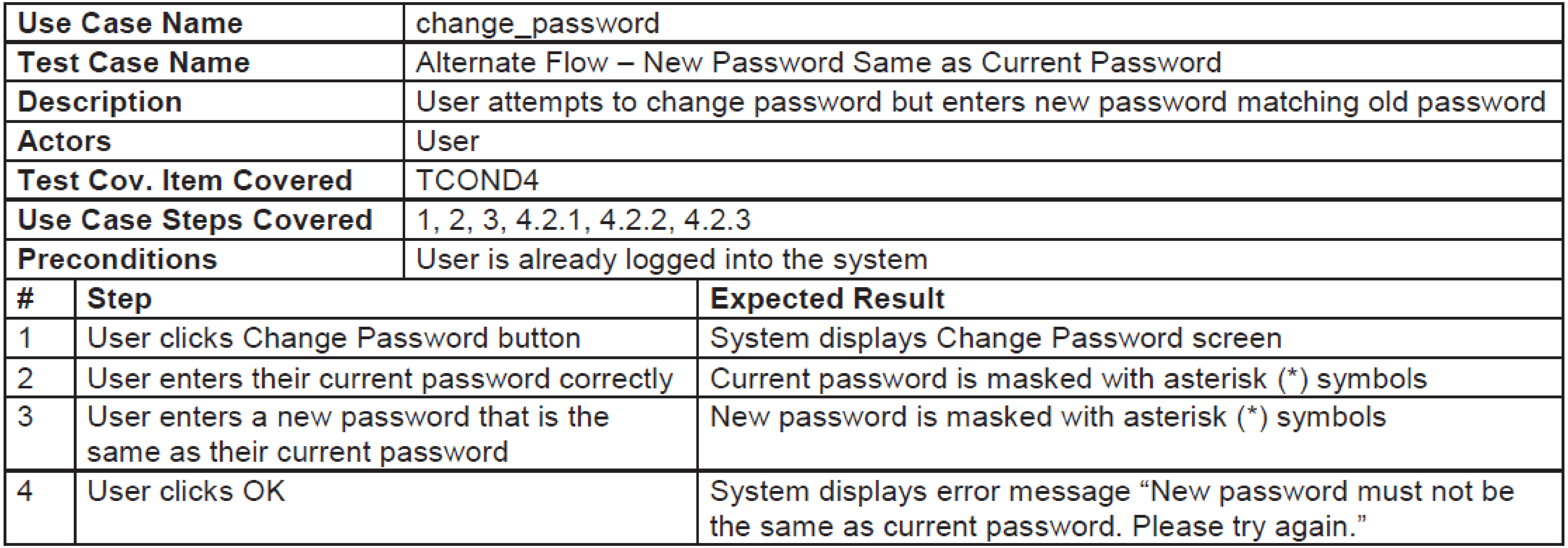


Use Case Name	change_password
Test Case Name	Alternate Flow – New Password Less Than 8 Characters
Description	User attempts to change password but enters less than 8 characters for password
Actors	User
Test Cov. Item Covered	TCOND3
Use Case Steps Covered	1, 2, 3, 4.1.1, 4.1.2
Preconditions	User is already logged into the system

#	**Step**	**Expected Result**
1	User clicks Change Password button	System displays Change Password screen
2	User enters their current password correctly	Current password is masked with asterisk (*) symbols
3	User enters a new password that is less than 8 characters long	New password is masked with asterisk (*) symbols
4	User clicks OK	System displays an error message “New password must be at least 8 characters long. Please try again.”

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Example: Test case for Flow 4

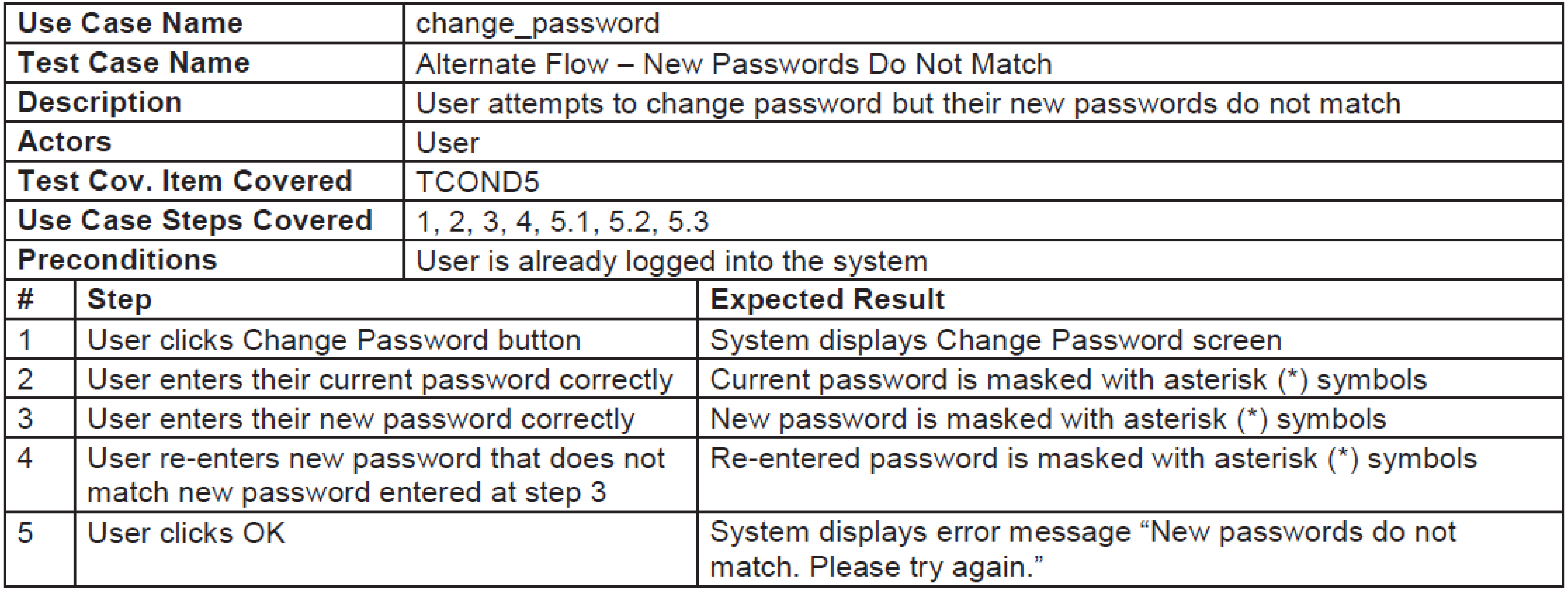


Use Case Name	change_password
Test Case Name	Alternate Flow – New Password Same as Current Password
Description	User attempts to change password but enters new password matching old password
Actors	User
Test Cov. Item Covered	TCOND4
Use Case Steps Covered	1, 2, 3, 4.2.1, 4.2.2, 4.2.3
Preconditions	User is already logged into the system

#	**Step**	**Expected Result**
1	User clicks Change Password button	System displays Change Password screen
2	User enters their current password correctly	Current password is masked with asterisk (*) symbols
3	User enters a new password that is the same as their current password	New password is masked with asterisk (*) symbols
4	User clicks OK	System displays error message “New password must not be the same as current password. Please try again.”

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Example: Test case for Flow 5



Use Case Name	change_password
Test Case Name	Alternate Flow – New Passwords Do Not Match
Description	User attempts to change password but their new passwords do not match
Actors	User
Test Cov. Item Covered	TCOND5
Use Case Steps Covered	1, 2, 3, 4, 5.1, 5.2, 5.3
Preconditions	User is already logged into the system

#	**Step**	**Expected Result**
1	User clicks Change Password button	System displays Change Password screen
2	User enters their current password correctly	Current password is masked with asterisk (*) symbols
3	User enters their new password correctly	New password is masked with asterisk (*) symbols
4	User re-enters new password that does not match new password entered at step 3	Re-entered password is masked with asterisk (*) symbols
5	User clicks OK	System displays error message “New passwords do not match. Please try again.”

Ref: [ISO/IEC/IEEE P29119-4-DISMay2013]

Test Coverage

The requirement in the form of use case		Test cases
Flow 1	Typical Flow	Test case for Flow 1
Flow 2	Alternate Flow – Existing Password Incorrect	Test case for Flow 2
Flow 3	Alternate Flow – New Password Less Than 8 Characters	Test case for Flow 3
Flow 4	Alternate Flow – New Password Same as Current Password	Test case for Flow 4
Flow 5	Alternate Flow – New Passwords Do Not Match	Test case for Flow 5

We covered all flows in change_password use case
Test coverage = 100%

$$\text{Coverage} = \frac{\text{number of coverage items exercised}}{\text{total number of coverage items}} \text{x}100\%$$

- measures the amount of testing performed by a set of tests (in some specific way)
- Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage.

Ref: [Graham]

How Many Types of Testing Coverage Metrics are There?

1. Line coverage
2. Branch coverage
3. N-length sub-path coverage
4. Path coverage
5. Multicondition or predicate coverage
6. Trigger every assertion check in the program
7. Loop coverage
8. Every module, object, component, tool, subsystem, etc.
9. …

101 types of testing coverage metric is listed in the reference

Ref: Software Negligence and Testing Coverage, Cem Kaner, 1996, http://www.badsoftware.com/coverage.htm

References

[Ammann]	Introduction to Software Testing, Paul Ammann, Jeff Offutt, Cambridge University Press, 2008.
[Graham]	Foundations of Software Testing, Revised Ed., Dorothy Graham, Erik van Veenendaal, Isabel Evans, Cengage Learning EMEA, 2008.
[Hambling]	Software Testing: An ISTQB-ISEB Foundation Guide, 2nd Ed., Brian Hambling (Editor), Peter Morgan, Angelina Samaroo, Geoff Thompson, Peter Williams, British Informatics Society Limited, 2010.
[Hunt]	Pragmatic Unit Testing in C# with NUnit, 2nd Ed, Andy Hunt, Dave Thomas, Pragmatic Bookshelf, 2007.
[ISO/IEC/IEEE P29119-4-DISMay2013]	IEEE Draft International Standard for Software and Systems Engineering - Software Testing - Part 4: Test Techniques, ISO/IEC/IEEE P29119-4-DISMay2013, 2014.
[ISTQB FLS]	Foundation Level Syllabus, International Software Testing Qualifications Board, 2011.
[ISTQB Glossary]	Glossary: Standard Glossary of Terms used in Software Testing, Version 2.4, International Software Testing Qualifications Board, 2014.
[Leffingwell]	Leffingwell, D., & Widrig, D. (2003). Managing software requirements: a use case approach (2nd Ed.). Addison-Wesley Professional.
[Meszaros]	xUnit Test Patterns: Refactoring Test Code, Gerard Meszaros, Addison-Wesley, 2007.
[Myers]	The Art of Software Testing (3rd Ed.), Glenford J. Myers, Corey Sandler, Tom Badgett, 2011.
[Naik]	Software Testing and Quality Assurance: Theory and Practice, Sagar Naik, Piyu Tripathy, Wiley-Spektrum, 2008.
[Orso]	"Software Development Life Cycles" course at udacity.com, Alex Orso, 2014.
[Patton]	Software Testing (2nd Ed.), Ron Patton, Sams Publishing, 2005.
[Pezze]	Software Testing and Analysis: Process, Principles and Techniques, Mauro Pezze, Michal Young, Wiley, 2007.
[Regehr]	"Software Testing" course at udacity.com, John Regehr, 2012.
[Sommerville]	Software Engineering, 9th ed., Ian Sommerville, Addison-Wesley, 2010.
[SWEBOK]	Guide to the Software Engineering Body of Knowledge, Version 3.0, P. Bourque and R.E. Fairley, eds., IEEE Computer Society, 2014; www.swebok.org.
[Zeller]	Why Programs Fail: A Guide to Systematic Debugging (2nd Ed.), Andreas Zeller, Morgan Kaufmann, 2009.