

Test Case Design Intensive Basic

Michael Hackett

STPCon 2019
San Francisco, CA



Copyright (c) 95-19 LogiGear Corporation. All Rights Reserved.

1

2

Today's Workshop Objectives

This workshop focuses on a strategic and tactical Test Case Development. Our goal is to help you maximize test productivity while minimizing maintenance cost and maximizing communication.

Basic Session Takeaways:

- Understanding the big picture with Test Documentation
- Knowledge and implementation of Lean Software Development ideas.
- Understand Test Case Basics
- Authoring excellent test cases in the *Agile age*.
- Test Case Best Practices for Automation.
- Test case design as a product design and specification activity.

Test Case Design Intensive

Chapter 1

Introduction & Themes of this workshop

Introduction

Today's workshop is divided into 4 sections

- Introduction
 - Glossary- getting it right.
 - Test Case authoring
 - Basics
 - Data Driven Testing
 - ABT
 - BDD
 - Other Test Case considerations
 - Test Case Maintenance
 - Test Cases for Automation
- ...with examples along the way.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

5

5

Pre-requisites

Out of scope for this workshop:

A great knowledge of testing:

- Fully understanding Regression
- Coverage
- Traceability
- Quality Costs
- Difference between bug finding and verification/validation

Use of test case tools:

- TFS, Jira, Zephyr...

Fluency in SDLC:

- Waterfall and terms
- Agile
- Scrum
- Kanban

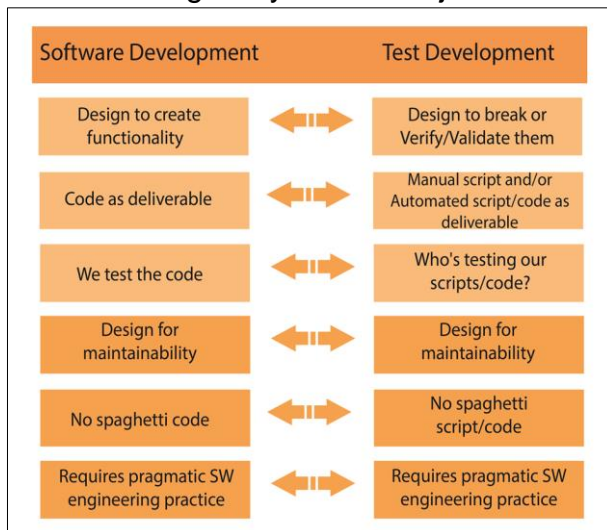
Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

6

6

Test development is as comprehensive as Software Development

There is a testing lifecycle. Its not just about test cases.



Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

7

7

Test Design

Writing test cases often goes by many names.

- Test design
- Test development
- Test authoring

A better way to think of it- its like software development.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

8

8

Test Development

1 – Test planning

Strategy is about how things will get done.

Test plans are about schedules, risk, coverage, scope, costs, tools—not test cases.

Test strategy and plan together—these can be high level: think coverage, environments, platform and compatibility, among other things.

2 – Organization

How to organize the project: test types, test suites, test modules

Module or folder structure is organization

3 – Test design—This is about authoring test cases.

Test Design

What does good test design look like?

- The test design is how you will get the testing done; designing the tests that are used to execute the testing project.
- It includes what tests you need to write to get the job accomplished.
- Test authoring is the big production part, and it includes defining the action works or keywords, but not the low level verbose step-by-step test execution. It's about what you are supposed to be testing, and how to drive data through the test.

Test Design

Test Design is engineering your tests.

- Design the test so you can *throw a bunch of data at it*.
- Design for both validating and exposing bugs. While engineering the tests, you should also think about design for re-use and easy maintainability.

Test Cases

Test case are the essence of testing.

- The long lasting asset and artifact
- The coming together of creativity, brains, skill, subject matter expertise...
- The key to successful automation.

Test Cases as Product Artifact

Test Cases can serve other purposes as well.

We will know how it works better than anyone else.

- Product spec
- Training tool
- Compliance
 - regulatory or contractual
- Traceability

Test Case Design Intensive

Chapter 2

Glossary Terms and SDLC

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

15

15

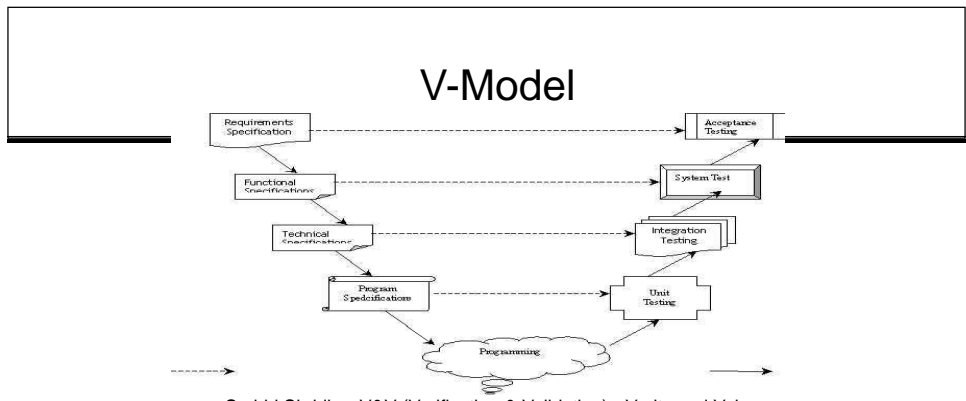
Begin scoping Test Cases

SDLC considerations and guidelines
A quick QA Glossary

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

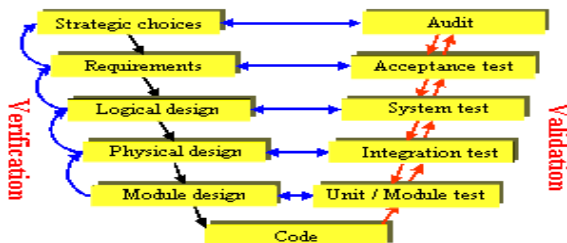
16

16



Surbhi Shridhar V&V (Verification & Validation) - Verity and Value

The V-model

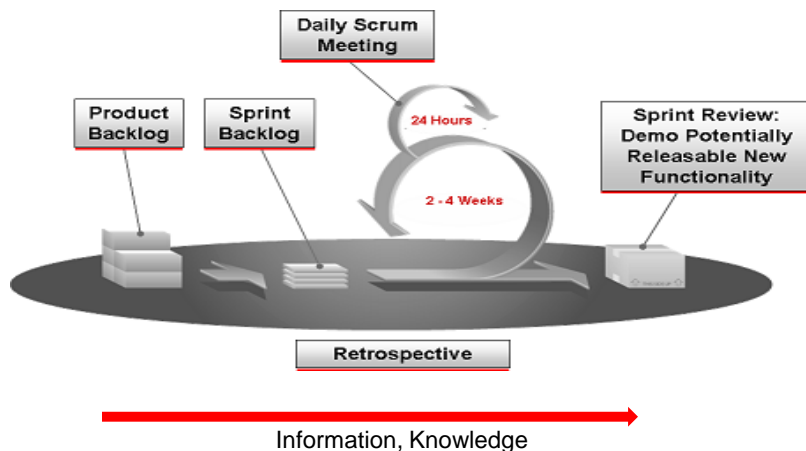


Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

Source: Glenford Myers (1979)

17

A Sprint



Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

18

18

By the book Scrum

No Test Cases.

Its all acceptance criteria.

No Bugs

Its all acceptance criteria.

Test cases and bugs are not *Lean!*

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

19

19

Lean Software Development

Lean software development is a translation of Lean manufacturing and Lean IT principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community.

7 Lean principles:

- Eliminate waste
- Decide as late as possible (JIT)
- Deliver as fast as possible
- Build integrity in/Quality at every step
- Empower the team
- Amplify learning
- See the whole

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

www.wikipedia.com

20

20

Artifacts in Agile

Common views on Lean:

If it does not go to the customer- it is a waste, you do not need it.

If it is not crucial, most important to the project- it is a waste, you do not need it.

If it is *byproduct* rather than *product*, it is waste.

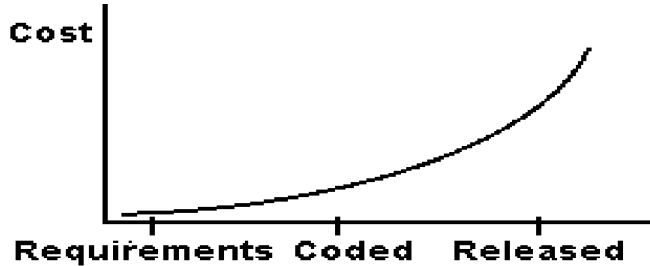
Some documentation (*waste*) is necessary- but we need to minimize it.

Lean Development

Many times I have seen the Lean principle to eliminate waste used to an extreme.

This is a common and bad idea.

Cost of Finding and Fixing Software Errors



These costs escalate because people are affected by bugs, and are more severely affected as the product gets closer to release. We all know the obvious:

- Bugs in requirements can be fixed without anything having to be recoded.
- Programmers who find their own bugs can fix them without taking time to file bug reports or explain them to someone else.
- It is hugely expensive to deal with bugs in the field (in customers' hands).

Along with this, there are many effects on other stakeholders in the company. For example, consider of the marketing assistant who spends days trying to create a demo, but can't because of bugs.

Put together a few ideas on Cost

When do you have the most information?

When is the best time for writing to keep maintenance costs low?

When do you have the most available time?

What are the uses of your test cases?

How much can you do JIT?

25

The essential Glossary

Getting the words right is really important to doing it right. We could spend a while on these words- here we will do a brief review.

- Test Goal
- Test Strategy
- Test Plan
- Test Case
- Test Suite
- Test Technique
- Test Method
- Test Types
- Test Module
- Test Steps
- Test Script
- Test Scenario

26

Test Strategy

Merriam-Webster's Dictionary

Strategy

1 a (1) : the science and art of employing the political, economic, psychological, and military forces of a nation or group of nations to afford the maximum support to adopted policies in peace or war (2) : the science and art of military command exercised to meet the enemy in combat under advantageous conditions b : a variety of or instance of the use of strategy

2 a : a careful plan or method : a clever stratagem b : the art of devising or employing plans or stratagems toward a goal

3 : an adaptation or complex of adaptations (as of behavior, metabolism, or structure) that serves or appears to serve an important function in achieving evolutionary success <foraging strategies of insects>

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

27

27

Test Strategy

A Test Strategy is **how** you will accomplish that goal.
Independent of a schedule and specific people.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

28

28

The Test Plan

A Test Plan is...

a document describing the scope, approach, resources, and schedule of intended testing activities. It defines test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

The ANSI/IEEE Standard 829-1983
for Software Test Documentation

Test Plans

“A Test Plan is a valuable tool to the extent that it helps you manage your testing project and find bugs. Beyond that, it is a diversion of resources.”

Testing Computer Software p. 205 1998

...*Lean* before it was cool.

Test Methods

Approached to develop tests for a specific purpose

Examples:

- Requirements-based testing
- Scenario-based testing
- Regression
- Forced Error Testing
- Error guessing
- Exploratory/AdHoc Testing.
- Model-based testing
- Path testing

Test Techniques

Methods used to document tests

Examples:

- Data Driven Testing
- Decision Trees/Decision Tables
- Combination Tests
- Keyword or Action-based Testing/ABT
- BDD
- Equivalent-class partitioning and Boundary Value Analysis

What is a Scenario?

Pronunciation: s&-'nar-E-'O, -'ner-

Etymology: Italian, from Latin scaenarium place for erecting stages, from scaena stage Date: 1878

1 a : an outline or synopsis of a play; especially : a plot outline used by actors of the commedia dell'arte b : the libretto of an opera

2 a : SCREENPLAY b : SHOOTING SCRIPT

3 : a sequence of events especially when imagined; especially : an account or synopsis of a possible course of action or events <his scenario for a settlement envisages... reunification -- Selig Harrison>

Merriam Webster Dictionary

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

33

33

Test Scenario

Scenario tests have three main components:

- Persona (called Actors in Use Cases)
- Sequence of steps
- Situation

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

34

34

Test Scenario

Persona (called Actors in Use Cases)

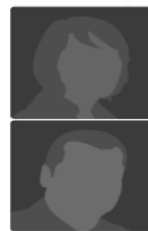
- Typical Users.
- Realistic- not generic.
- The more realistic, the better the test case.

Scenario-based Testing Persona

Personas

Modeling techniques to describe your target audience

- Behavior
- Patterns
- Goals
- Skills
- Attitudes
- Motivation
- Environment - useful when you don't have easy access to **real users**



Personas

- Help to **guide your decisions** about functionality and design
- Let you be in your user's shoes – think the way that your users would
- Give ideal for building test script relying

Scenario-based Testing Persona

The Greenhorn	The Casual User	The Teacher	The Business User	The Power User	The Hacker
JOHN	EMILY	AKIKO	STEPHAN	ROBERTO	RICKY
Profile: Probably the single biggest segment of mobile users. Want simple: turn on their mobile, dial a number and talk to their extended party. Don't care about anything other than the mobile being able to be used as a phone, and possible contacts.	Profile: Take advantage of most phone features, but not all. Use the phone to make calls, use the contacts, send text messages, and take pictures. Their mobile is always with them.	Profile: Teaching is far more popular than calling. Will use and receive thousands of text messages per month. Rarely use their phone for calling. Want a clean testing interface with the fastest possible input.	Profile: Wants a phone that is simple, but functions as an integrated smart device. Wants to read email and call back the sender with the least amount of effort. Needs "thunder" mail server integration, including Blackberry and Exchange.	Profile: Will use almost all of the built-in functionality. Will also extend their phone's functionality with additional software. Will flip through menu menus options and changing settings.	Profile: Care more about customization. Wants to make changes to every aspect of the phone. Likes to making lists and forums about hacking the phone. Contributes to the open source community.
Scenario: I didn't get my first phone until 2001. My daughter thought it was necessary but since then, I have it with me all the time and use it more than my home phone.	Scenario: My phone has to look cool - personalize it with shells, themes, and ring tones. Talk on it everywhere, so my phone style is everything. Of course, it has to work too. I usually talk on the phone, but recently started taking pictures and recording video. My phone is my favorite accessory.	Scenario: I prefer texting over calling because it's more fun and private. My friends and I probably text each other several times a day. We'll never text to order food. It's far more interesting and less expensive. I don't have to worry about disturbing people on the train with my talking. I love it.	Scenario: My mobile is my life. Without it my business would suffer. I take conference calls while driving down the hill. If someone text me, I need to ring them without taking my eyes off the road. And since I use my mobile everywhere, it needs to be reliable. The last thing I need is for it to break after one day.	Scenario: I'm addicted to new toys. I got the latest gadget as soon as they arrive on the market. I upgrade my phone every 6 months. I guess you can say this is almost a sport for me. Or an addiction. I just love to explore the latest and how it can make life fun.	Scenario: As soon as I found out about an open source phone, I jumped on it. I traded my laptop for the phone and am working on the testing solution. I probably should spend more time at my day job, but this is far more fun. Of course, I use my phone for calls and texting too.

Reference: Openmoko
<https://i.pinimg.com/originals/a4/cf/2d/a4cf2dabb7c772630947168fc0c302d8.jpg>

Test Case

Test Case

A specific set of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

(IEEE 729-1983)

A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

After [IEEE,do188b]

Test Case

Test Case: A test that (ideally) executes a single well defined test objective (i.e., a specific behavior of a feature under a specific condition).

Testing Computer Software
Kaner, Faulk, Nguyen

Test Script, Suite

Test Script: A particular set of step-by-step instructions describing how a test case is executed. A test script may contain one or more test cases.

Test Suite: A collection of test scripts or test cases used for validating bug fixes (or finding new bugs) within a logical or physical area of the tested product. For example, an acceptance test suite contains all the test cases used to validate that the software has met a certain predefined acceptance criteria; a regression suite contains all the test cases used to validate that all previously fixed bugs are still intact.

Test Module

Module

Similar to test suite

Its how the tests are organized

It contains test objectives and test cases to test them, all defined within a single scope.

Test modules provide a level of abstraction over test cases and make it possible to create well-defined test case flows. The top-down planning approach helps to create test cases that are free of unnecessary details and redundant checks. It also effectively separates the design of the tests from how they will be executed, allowing tests to be used for both manual and automated test execution.

Test Suites Modules Folders

Test Suites and Test Modules or even sometimes Folders seem to have overlap but there are a few clear differences.

Test Suites and more about running tests than designing tests.

The phrase Test Suite has been taken over by automation tools where suite is a set to be run for

Test Module is a cluster of tests with common attributes for organizations and lower maintenance.

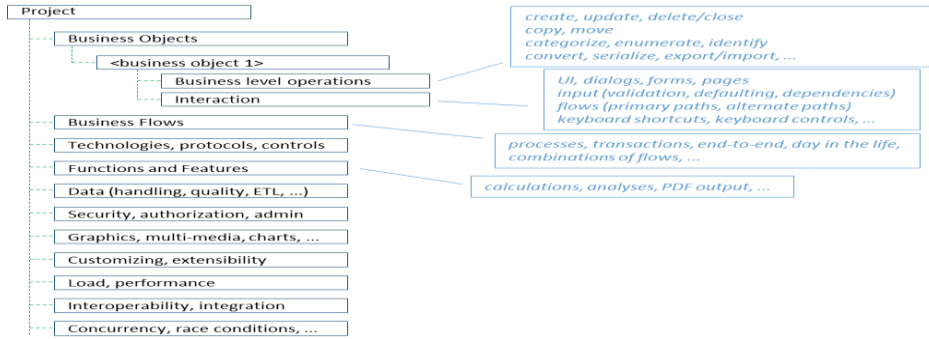
Some people organize tests into *folders*.

Smart Suite or Module design will help prevent over-checking.

Over-checking is step-by-step validation and very old, outdated test design.

This is particularly true of UI checking.

Modules/suites ABT structure



Test Type

Test Type: A specific logical testing objective in which a collection of tests attempts to expose errors or verify the correctness of the application's behavior.

Read "Test Types and Their Place in the Software Development Cycle".

Test Types

Test Types can be broad and often have sub-sets.

Installation:

- Clean Install
- Over-install
- Re-install
- Upgrade
- Uninstall

Some Test Types

Load	UI
Volume	Usability
Stress	Documentation
Performance	Help
Boundary/Limit	Collateral
Installation	3rd Party
Security	Internationalization
Failover	Localization
Database	Smoke
Data Integrity	Quick Look
Configuration	End-to-End
Compatibility	Real World
Platform	Memory
Out of Box	Functionality

Test Artifacts

Hierarchy: How this all fits together.

- Test Goal
 - Test Strategy
 - Test Plan
 - Test Types
 - Test Suite
 - Test Cases
 - Test Script

Test Case Design Intensive

Chapter 3

Details on Test Cases

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

49

49

Documenting Tests

Why write Tests?

- Accountability
- Reproducibility
- Tracking
- Automation
- To find bugs
- To verify that tests are being executed correctly
- Use as a Training Tool for new Testers.
- For Compliance
- To measure Test Coverage.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

50

50

Good test cases vs Bad test cases

Good Test Design has:

- Effective test cases that find bugs and issues early
- Gives the ability to convey confidence in the test effort or measure coverage to team
- Tests that are easy to write, read, review and maintain
- Improved customer experience... (and if we aren't improving that, what are we doing?)

Failed test design has:

- Missed bugs and/or late bugs
- No way to give the team confidence in the project
- Verbose tests; too many details; take a long time to write; difficult to understand without deep product or technological knowledge
- High maintenance cost for test cases especially when the tests are maintained by an engineer who did not created them originally.
- Problems with automation, needing a separate project to automate badly designed tests

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

51

51

Good and Not so Good Tests- low level

A “Good Test”:

- Refers to and maintains tight control over specific test data
- Has detailed enough Test Design Steps so that any tester with basic knowledge of the system can execute the Test
- Is aware of the Tester's experience
- Has clear criteria for pass or fail

A “Not so good” Test:

- Leaves it up to the user to find test data
- Gives very high level instructions that leave too much room for “artistic interpretation”
- Does not consider the Tester's experience
- Leaves out follow-up verification steps which make it difficult to determine pass or fail criteria

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

52

52

Goal of a Good Test Design

It can describe to anyone the intent of the test

Is a reference tool to make sure the right tests are getting done.

The Test Design works like a Test Requirement, that is, it details what tests are required to satisfy the project's testing need.

Ability to Defend your testing position

Good test cases vs Bad test cases

Failed test design, especially for automation projects have:

- High maintenance costs for test author and future automation engineers
- Wastes time and money on poorly designed scripts, or results in a lack of designed test scripts
- Little to no value, giving a false sense of confidence
- Missed bugs
- Fragile tests—tests that break often or easily
- Unwieldy suites of too many tests lacking quick value

Test Case Essentials

Why you *may not* document ALL test cases...

- Experienced testers have the freedom for Exploratory testing.
- There may be no solid information on which to base the cases.
- There may be too many tasks and not enough time to document.

Test Case Essentials

Why you *may not* document ALL test cases...

- Scrum/Lean does not prescribe it.
- Your team may not care.
- No other use than your organization
- Low value tests.
- Only doc higher value or higher risk tests.

How to author test cases

What is a test case?

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

57

57

How to author test cases

Test Case

A specific set of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

(IEEE 729-1983)

A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

After [IEEE,do188b]

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

58

58

Test Case possible contents

What can tests often, *but not always*, contain:

- Test Title/Objective
- Steps/Script
- Pre-conditions (setup)
- Data
- Post conditions (cleanup)
- Environment
- Expected Results / Validation Point

How to author test cases

What are the other parts of a test case?

Accounting (**done by tool/repository**)

ID

Environment

Author

Test run

Build

Date

...etc.

This is not the focus today

Contents of a Good test cases

Test Objectives—These specify the intention and goal of the test; what do you hope to find from the test. In some cases, this is the most effective part of the test to review for sign-off. The objective is the test idea or goal: the reason why you are doing the test. These convey the real meaning of the test.

Test Steps—The step-by-step are the highest-maintenance part of documented tests: not of great value. Avoid documenting the steps until the latest point possible.

Contents of a Good test cases

Test Data—Often the most interesting part of a test. The most effective tests are boundary tests. Use Equivalence Class Partitioning for the most efficient tests. Interesting data gives interesting tests. Boring data gives boring tests.

Validation point—The pass/fail criteria, expected results, check, checkpoint. Use good and correct validation points. Make sure that what you *think* you're testing is what you are *actually* testing. Validation points need to be reviewed.

Basics in documenting tests

Each Test Case needs a *Test Idea* or *Test Objective*.

- It specifies what you are looking for and explains the intent of the test.
- In some cases, this is the most effective part of the test to review for sign-off.

Test Objective

The most important part of a test case is the 1-line title describing the objective of the test.

That 1-line title can be called:

- Test Title
- Test Name
- Test Case
- Test Objective
- Test Goal

Good Test Case Objective

Test Objective

The main goal is to convey the meaning of the test. Below are three sample syntax that are easy, common, easy to teach:

Action+ Function+ Operating condition

ABT: Cause and Effect (From Action Based Testing)

BDD: Given-When-Then (From Behavior Driven Development)

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

65

65

Test Objective

It is most important because:

- It gives the reader a description and idea of the test
- A good test name makes review easier
- Easier to pass to another person
- Easier to pass to automation team
- Describes intention of the test
- In many cases, may be the only part of a test case documented.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

66

66

My favorite style Test Objective

Test Objective suggested syntax

Action + Function + Operating Condition

Function may be function, feature, validation point
Condition may be data

Test Objective

Action:

- Test
- Validate
- Prove
- Execute
- Print
- Calculate
- Run
- ...any action verb

Test Case Title Syntax

Action	Function	Operating Condition
Run	annual report	from standard data (file location)
Run	annual report	on Day 1 of fiscal year
Run	annual report	from empty spreadsheet
Run	annual report	on last day of fiscal year

Bad test case names/objectives

TC_01_Online_Login_Success
 TC_02_Online_Valid_Case

If these are meant to be long term assets, names like this are a maintenance nightmare.
 If they are temporary value, why write them at all?

Bad Examples

Syntax:

verify << copy acceptance criteria >> is displayed in <>

Or

check if << copy acceptance criteria >> is displayed in << copy acceptance criteria>>

Example:

//Test case to verify forename, surname, postal code is displayed in accounts
Screen verify the fields forename, surname, post code is displayed in accounts
screen

If you already have defined acceptance criteria, additional test cases with the same
text are waste.

I sadly still find these style test cases around.... Please stop. Its 1990. Not 2018.

Unique “standard”

Verify <TC Expected Results> when <TC Execution Conditions>

Original Wording of Test Case Title:
Verify when updating Carrier Profile in OFS and change the City field, the profile
is updated properly with no errors generated on the screen.

Apply syntax:

TC Execution Condition	TC Expected Results
User changes Carrier City	OFS updates Carrier Profile with no errors

Revised Wording of Test Case Title:
Verify OFS updates Carrier Profile with no errors when user changes
Carrier City

Not credited

Weird “standard”

- Each Interface or Extension will generate 35 test cases**
- Each Report item will generate 15 test cases**
- Each Test Case will take 30 minute to write and 30 minute to execute**
- Each Tester can execute 75 test cases per week**
- Each Test Case will be executed 2 times during a testing event**
- Reviewing a test case takes 5 minutes/test case**
- UAT test cases usually cover about 20-25% of total test cases**

Not credited

“Business Readable”

- Business Readable DSL**
- Domain Specific Language**

DSL

The current move in testing today is domain specific, business readable test cases and documentation.

- A **domain-specific language (DSL)** is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains, and lacks specialized features for a particular domain.

Wikipedia.org

Test Steps or Script

Stop writing steps! That is the script.

They are time consuming, difficult to maintain,
not as needed as some people think!

Be Lean!

Test Steps or Script

The movement in testing these days is away from focus or even writing steps.

Especially with “over-checking” validation points.

Few people have time for step by step by step by step

If you *have to* write them- do it late/JIT. Cut maintenance.

Test Steps or Script

Are there some situations to write steps?

Use your brain! Write them only if you have to:

- Compliance
- Giving the tests to new people or from outside team.
- Someone who does not know the app will execute/automate.
- There are multiple ways to do the same task...but these details could be in the script.
- Training tool (product or people).
- Special case:
 - L10N/Translation testing
 - Contract

Setup and Tear Down

Many tests need setup: Initialize the test, set the exact or correct situation, add preconditions where needed.

Many need to be torn down and cleaned up after running.

Some people suggest tests can be *piggy-backed* to optimize setup and tear down. Use good judgment, as many people will suggest isolating tests to be completely independent. It would be too simplistic to make a blanket statement here.

- Think about lowering or increasing maintenance costs.
- Think about the speed of execution.
- Think about avoiding cascading failures where one test fails and causes more tests to fail, not because there are issues, but only because a test is reliant on a previous test passing to be executed.

Basics in documenting tests

Tests need validation points.

- This is the expected result. It can also be stated in the test objective.
- Do not rely on “seeing” that a test passes or fails. Write it.
- Many times it is easier to define the test once you clearly state what behavior, result or point you are attempting to validate.

Basics in documenting tests

Tests need validation points.

Example:

Email sent

db value

Navigation result

Correct or Error

Correct calculation

UI value

Remote api setting

Session or state set

...

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

81

81

Test Case Reviews

Review!

QC your test cases

QC each other's tests

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

82

82

Effective Test Case Design

Test Case Development using
Spreadsheets
Data-driven Testing

Test Cases using Spreadsheets

Test Scripts.

Step-by-step text descriptions of exact procedures to execute a test, often with explicit data.

In some cases this style test is still needed.

Since the need for testing to be both more documented and transparent has grown there has been an effort to make the test documentation process, time to complete and maintenance easier.

Test Cases using Spreadsheets

Step-by-step with explicit data:

- Naïve (not much trust of testers)
- May need for automation
- May need for regression testing
- In certain cases the test case may need to be explicit
- May need for bug report
- May use as a training tool for new testers

But!

- These tests generally take longer to write
- Harder to maintain.

Effective Test Case Design

Data Driven Testing

Data Driven Testing

Data driven testing is a technique for developing test cases that keeps the test script separate from the test data.

The reasons for doing this are:

- It prolongs the life of both the data and the test script. They can be maintained separately and more easily.
- It provides for running a larger volume of tests
- It is a common use in writing test cases for automation.
- The tests are more easily readable and reviewable for test coverage.
- Clustering tests makes them easier to write.

Data Driven Testing

Contact Us > Send E-mail >

Visitor Interested in More Information

Contact Details

Title (Optional):

First Name:

Last Name:

E-mail Address:

Your Inquiry

Subject:


Comments:

Note: We are unable to act upon any trading requests to buy, sell, or exchange securities delivered through e-mail. E-mails to Fidelity are encrypted for security, and Fidelity keeps all information confidential. We do not give or sell this information to other companies. Read [Fidelity's Commitment to Privacy](#).

Do you have questions about opening an account?

Yes

No

 © Copyright 1998-2004 FMR Corp. All rights reserved. [Terms of Use](#)

Data Driven Testing

The script is simple:

- 1 Select a Title, or not.
- 2 Enter a first name
- 3 Enter a last name
- 4 Enter an email address
- 5 Enter a subject
- 6 Enter a Comment
- 7 Choose "have questions" or not.
- 8 Click "Send email".

Separately there can be spreadsheets with as much data and validation points as you have time to write!

Data Driven Testing

Just for the email text box how many test cases could we reasonably write?

name@domain.extension

mary@yahoo.com
 mary.engles@company.com
 Mary.Engles@Company.com
 mary.engles@company.org
 mary.engles1@company.com
 mary_engles@company.com

mary.engles@ucb.edu
 mary.engles1@ucb.edu
 mary.engles@eng.ucb.edu
 mary_engles@ucb.edu

mary.engles@chemco.biz
 mary.engles@nashville.doe.us.gov
 mary.o'engles@nashville.universityoftennessee.edu

...
@.net

Data Driven Testing

Example Spreadsheet for some data driven tests

Test Case	Title	First Name	Last Name	email	subject	comments	open?	Expected Result
1	Mr	blank	blank	._@_.net	blank	blank	on	request sent
2	Mrs	A.	A.	mary.engles@chemco.biz	space	space	off	error #1
3	Dr	A.Z.	O'Leary	mary.engles@company.com	?	?		error #2
4	none	A.B.a.b.	Jones-McCleary Watson	Mary.Engles@Company.com	>	>		error #3
5		Ed	...	mary.engles@company.org	+	+		error #4
6		José		mary.engles@eng.ucb.edu	*	*		...
7		Mary-Anne		mary.engles@nashville.doe.us.gov	</html?	</html?		
8		reallylong firstname		mary.engles@ucb.edu	a	a		
9		...		mary.engles1@company.com	really long subject	really long subject		
10				mary.engles1@ucb.edu	123-456-	123-456-		
				mary.o'engles@nashville.universityof tennessee.edu		
				mary@yahoo.com				
				mary_engles@company.com				
				mary_engles@ucb.edu				
...				...				
150								

Data Driven Testing

Because the documentation for data driven testing should be easily read it is also a great education tool for the development teams into what testers do- similar to a product spec.

Summary

What did you learn?

Summary

A glossary for test artifacts
Lean Software Development (LSD)
Test Case Basics
The importance of Test Objectives
An easy syntax for authoring Test Cases.

Test Case Design Intensive

BREAK CLASS

Basic and Advanced

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

95

95

Test Case Design Intensive
Advanced

Michael Hackett

STPCon 2018
Arlington, VA



Copyright (c) 95-18 LogiGear Corporation. All Rights Reserved.

96

97

Today's Workshop Objectives

This workshop focuses on a strategic and tactical Test Case Development. Our goal is to help you maximize test productivity while minimizing maintenance cost and maximizing communication.

Advanced Session Takeaways:

- Authoring excellent test cases in the *Agile age*.
- Action based testing/ABT and BDD test design.
- Test Case Best Practices for Automation.
- Test case design as a product design and specification activity.

98

Introductions

Recap from this morning:
What did you learn?

Test Case Design Intensive

ABT
Action-Based Testing
Higher level Test Design with
Keyword-based Testing

ABT

Keyword driven testing has been around for a few decades. It works very well.

ABT is a higher level methodology for organizing a test project.

Two main goals of ABT are scalability and low maintenance.

ABT

ABT is a top down, JIT approach to Test Development

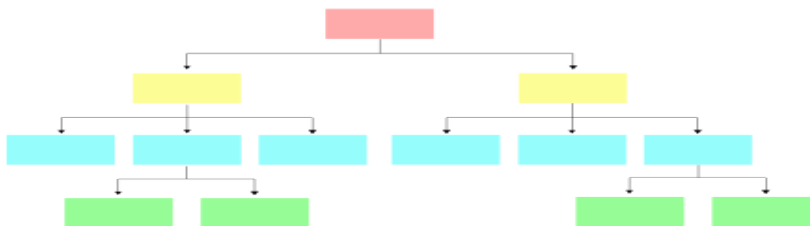
Test Suites Test modules

Start with the organization
Global and detailed organization
 Top down
 No low level details for a while...

Example global test design
Business versus interaction
 Differentiating interaction tests
 Business objects and business flows
Special tests

Top Down Approach

High level comes first, as details are needed and emerge, add them, JIT.



Next levels down

Developing the test modules:

1. Creating a Test Module
2. Defining Test Objectives
3. Writing tests
4. Checks, errors, warnings
5. Commenting, documentation
6. Variables and expressions
7. Data sets

Example High level organization

Some of the main categories in this template are:

- Business Objects
- Business Flows
- Features
- Interoperability
- Components
- Graphics, multi-media
- Technologies
- Data
- Administration
- Customizability
- Concurrency, race criteria
- Non-functional tests

ABT Test Objectives

ABT Syntax:

Cause & effect

- Refunded items should be returned to stock
- Replaced items should be returned to stock
- Clicking submit empties all fields
- If all fields are populated, ok is enabled
- OK becomes enabled if both first name and last name are specified
- In the case of a sports car: the screen specifies seconds to reach 60MPH
- In the case of a sports car: no lease price is available

Last, Low level

Actions

- System level, application level, navigation, utility
- Built-in, user defined, scripted
- Variations
- Recommendations for actions
- Using a high level action
- Hiding navigation
- Over-Checking

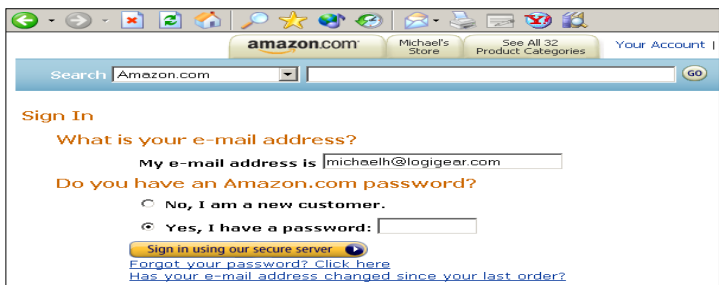
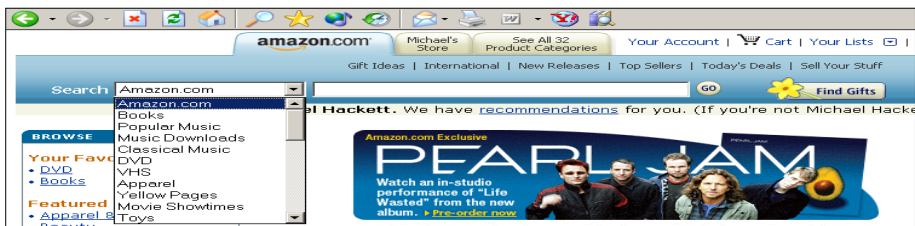
Test Module and Cases

TEST MODULE		Restaurant Reservations			
OBJECTIVES					
test objective	TO 01	Reservation complete will send confirmation email			
test objective	TO 02	User can change reservation details			
test objective	TO 03	User can cancel the reservation			
INITIAL Setting up					
username	password				
sign in	john	j@logout23			
SECTION Pick some dates to use in the tests					
days from now	result				
pick date	1	>> tomorrow			
pick date	7	>> next week			
pick date	30	>> next month			
TEST CASE TC 01 Reserve a restaurant					
test objective	TO 01	Reservation complete will send confirmation email			
restaurant	Evvia	party size	date	time	
make reservation		2	# tomorrow	9 PM	
username	password				
check confirmation email	john	Your reservation is confirmed for Evvia			
TEST CASE TC 02 Change reservation multiple times					
test objective	TO 02	User can change reservation details			
restaurant	Evvia	party size			
change reservation		4			
username	password				
check confirmation email	john	Evvia will be ready for your party of 4			
restaurant	Evvia	date	time		
change reservation		# next week	8 PM		
username	password				
check confirmation email	john	# "at 8:00 PM on " & next week			
TEST CASE TC 03 Change multiple details of one reservation					
test objective	TO 02	User can change reservation details			
restaurant	Evvia	party size	date	time	
change reservation		10	# next month	6 PM	
username	password				
check confirmation email	john	# "party of 10 at 6:00 PM on " & next month			
TEST CASE TC 04 Cancel reservation					
cancel reservation	Evvia				
username	password				
check confirmation email	john	You've successfully canceled your reservation at Evvia			
FINAL Cleaning up					
sign out					
close all browsers					

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

109

Action-based Testing



Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

110

110

Action Word example Amazon.com

Test Objective: Verify an order with two "ship-to" addresses.

- 1 Go to Amazon.com
- 2 Click on Search product type drop-down arrow.
- 3 In Product drop down box choose "Books"
- 4 Click in search box
- 5 Enter "Italian Cooking"
- 6 Click "GO" button
- 7 Choose any book from results by clicking on book title.
- 8 Enter (1) in Quantity and Add to shopping cart
- 9 Click on Search product type drop-down arrow.
- 10 In Product drop down box choose "DVD"
- 11 Click in search box
- 12 Enter "101 Dalmatians"
- 13 Click "GO" button
- 14 choose the DVD by clicking on title.
- 15 Enter (2) in Quantity and Add to shopping cart
- 16 click "proceed to checkout" to get to sign-in page
- 17 click in the email address field.
- 18 Enter "testaccount.mycompany.com"
- 19 Tab twice to the password field.
- 20 Enter the testing password "test1234"
- 21 Select (X) "show gift options" during checkout
- 22 Submit

111

Action Word: First, separate the data.

Test Objective: Verify an order with two "ship-to" addresses.

Steps	Data
1 Go to Amazon.com	
2 Click on Search product type drop-down arrow.	
3 In Product drop down box choose "DATA 1"	Books
4 Click in search box	
5 Enter "DATA 2"	Italian Cooking
6 Click "GO" button	
7 Choose any book from results by clicking on book title.	
8 Enter (DATA 3) in Quantity and Add to shopping cart	1
9 Click on Search product type drop-down arrow.	
10 In Product drop down box choose "DATA 1"	DVD
11 Click in search box	
12 Enter "DATA 2"	101 Dalmatians
13 Click "GO" button	
14 choose the DVD by clicking on title.	
15 Enter (DATA 3) in Quantity and Add to shopping cart	2
16 click "proceed to checkout" to get to sign-in page	
18 click in the email address field.	
18 Enter DATA 4	testaccount@mycompany.com
19 Tab twice to the password field.	
20 Enter the testing password "DATA 5"	test1234
21 Select (X) "show gift options" during checkout	ON
22 Submit	

112

Make an Action Glossary

SEARCH Action

- | | | |
|---|---|--------|
| 1 | Click on Search product type drop-down arrow. | |
| 2 | In Product drop down box choose "DATA 1" | DATA 1 |
| 3 | Click in search box | |
| 4 | Enter "DATA 2" | DATA 2 |
| 5 | Click "GO" button | |
| 6 | Choose any book from results by clicking on book title. | |

The test case becomes:

STEPS

- | | | | |
|---|----------------------|------|-----------------|
| 1 | Go to Amazon.com | | |
| 2 | Search | Book | Italian Cooking |
| 3 | Add to Cart | 1 | |
| 4 | Search | DVD | 101 Dalmations |
| 5 | Add to Cart | 2 | |
| 6 | Procees to Checkouot | | |
| 7 | Login | | |
| 8 | ... | | |

But then we see another pattern!

We make a new action. A higher level action called Fill Cart.

FILL-CART Action

- | | | | |
|---|--|--------|---------|
| 1 | Search | DATA 1 | DATA 2 |
| 2 | Enter (DATA 3) in Quantity and Add to Cart | | DATA 3# |

Action Word

Add LOGIN Action

- | | | |
|---|--|---------------------------|
| 1 | click "proceed to checkout" to get to sign-in page | |
| 2 | click in the email address field. | |
| 3 | Enter DATA 4 | testaccount@mycompany.com |
| 4 | Tab twice to the password field. | |
| 5 | Enter the testing password "DATA 5" | test1234 |
| 6 | Select (X) "show gift options" during checkout | ON |
| 7 | Submit | |

The Final optimized steps:

- | | | | |
|---|---------------------|---------------------------|-------------------|
| 1 | Go to Amazon.com | | |
| 2 | FILL-CART | Book | Italian Cooking 1 |
| 3 | FILL-CART | DVD | 101 Dalmations 2 |
| 4 | Proceed to Checkout | | |
| 5 | Login | testaccount@mycompany.com | test1234 |
| 6 | Submit | ON | |

Final Test Case and Glossary

STEPS					
1	Go to Amazon.com				
2	FILL-CART	Book	Italian Cooking	1	
3	FILL-CART	DVD	101 Dalmations	2	
4	Procees to Checkout				
5	Login	"testaccount.mycompany.com"	test1234	ON	
6	...				

SEARCH Action

- 1 Click on Search product type drop-down arrow.
- 2 In Product drop down box choose "DATA 1" DATA 1
- 3 Click in search box
- 4 Enter "DATA 2" DATA 2
- 5 Click "GO" button
- 6 Choose any book from results by clicking on book title.

FILL-CART Action

- 1 Search DATA 1 DATA 2
- 2 Enter (DATA 3) in Quantity and Add to Cart DATA 3#

LOGIN Action

- 1 click "proceed to checkout" to get to sign-in page
- 2 click in the email address field.
- 3 Enter DATA 4 "testaccount.mycompany.com"
- 4 Tab twice to the password field.
- 5 Enter the testing password "DATA 5" test1234
- 6 Select (X) "show gift options" during checkout ON
- 7 Submit

Actions

Action Word

The first step into keyword driven test design is generalizing or abstracting the functionality of the software, typically around a submit. Each of these actions, or keywords is a submit.

Some companies using the RUP/use case development have an action for each use case. They share the same name.

Rather than step-by-step how to accomplish an action, trust the tester- and make test documentation easier- by leaving out the step-by-step and only documenting what action is taking place.

Actions

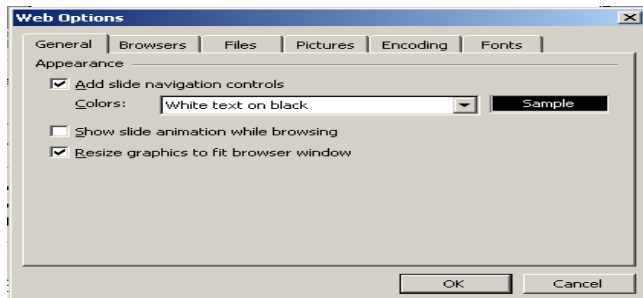
Notion of an Action, in this specific case- data with a submit.

- Login: UID & PWD
- Search in Amazon: Product type & search string
- Checkout: Product code & quantity & price & size/weight constraints for shipping.

117

Action-based Testing

Notion of an Action, in this specific case- data with a submit.



General Web Options

In this PowerPoint Web Options settings function the Submit is an "OK".

The data is:

- Add slide (enabled/disabled),
- Color (5 dropdown choices),
- Show slide (enabled/disabled),
- Resize graphics (enabled/disabled).

118

Action Based Testing Approach

- Top Down, High level organization
- Meaningful Module Design
- Test Cases, Objectives clustered in Modules

- Action Design last
 - Separation of action, logic and data
 - Test data can go in separate file or spreadsheet
 - Test logic can go in separate file or spreadsheet

Action-based Testing

- Low maintenance sharing of modules and actions
- Basis for building automation framework

Low level Tests are defined using parameterized, reusable *actions*

- Non-technical testers and business analysts can easily create and maintain *automated* tests using actions
- Automation engineers focus on implementing and maintaining *actions*, not *tests*.

ABT Anti-patterns

Wikipedia describes an anti-pattern as “a common response to a recurring problem that is usually ineffective and risks being highly counterproductive.” The term was coined by Andrew Koenig, based on the well-known notion of “design patterns.”

- *Enter Enter Click Click*: Test steps are too detailed. Many tests, even automated ones, have been designed on a detailed step-by-step basis. This makes it difficult to manage and maintain those tests. For example, it will be hard to factor the impact of changes in the application under test into the tests. Identifying common sequences of steps and putting them in actions or functions will relieve this problem.
- *Interaction Heavy*: Not having many business tests. A main distinction I recommend that testers make is between “business tests” that focus on business objects, rules and processes on one hand, and “interaction tests” that focus on the interaction with the application. However, I've seen in many projects that testers focus only on the interaction. This makes the tests shallow and misses potential business level problems.
- *Lifeless*: Missing life cycle steps of business objects. Most applications work on “business objects,” like orders, invoices, products, customers, etc. These objects have their life-cycles in the application, like creation, update, retrieval and closure, and also operations like copy, move, export, import, etc. However, the tests for such basic operations in applications are often scattered and as a result, hard to find and incomplete.
- *Lame*: No depth or variety, no testing techniques used. Time pressure and other factors often result in shallow test cases that don't challenge the application much. Try to think as a tester, somebody who wants to break things. Applying testing techniques and interaction with various stakeholders can help you in this process.
- *Clueless*: No clear scope for the tests. A very common situation is lack of scope for tests. The tests then are hard to find and assess, and may do work that is also done in other tests.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

121

121

ABT Anti-patterns

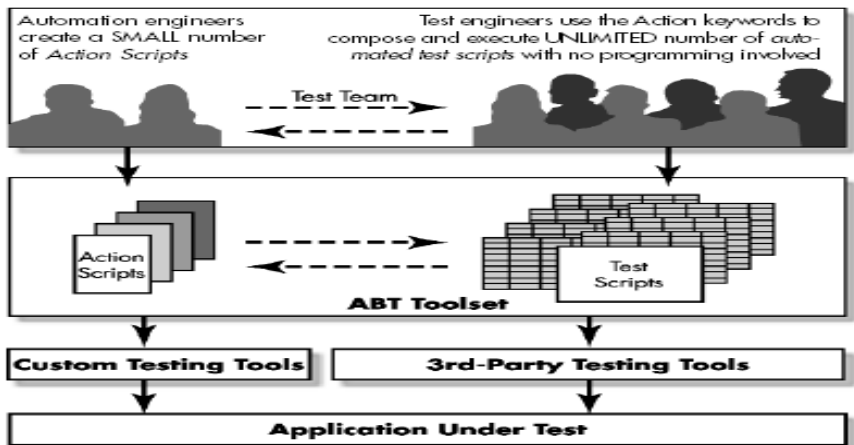
- *Over-Checking*: Checks not relevant for the scope. Since test designers often follow an approach of steps with an expected result for each step, tests do many checks that do not fit the scope of such tests. Such checks are unnecessary and probably over-lapping similar checks elsewhere. They then clutter up result statistics (e.g. they create too many “passes”), and aggravate the impact of changes in the application under test.
- *Cocktail*: Interaction tests mixed with business tests. Even if tests are testing business functionalities, like business object life cycles and business rules, calculations and processes, they are often mixed with tests on interaction details, resulting in a convoluted and hard to maintain mix. A common example is to describe a log-in process in detail in all tests that start with a log in.
- *Sneaky Checking*: Checks hidden in actions. Even though it is good to have business level actions that hide unneeded details for many of the tests, try to avoid hiding too much. In particular, checks should be explicit and visible in the main test (the test modules), at the appropriate level of detail. An outsider should be able to understand what is being tested by just looking at the test module, without a need to inspect how actions are implemented.
- *Action Explosion*: Many actions, with little re-use. Some testers may have interpreted a statement like “actions are good” too literally. In tests one then sees actions for every little step in a test. The result is many (thousands) of actions, and even though actions are meant to ease maintenance, they themselves become hard to manage.
- *Mystery Actions*: Actions should have clear names, representing their function. In some projects one can find actions like “verify transaction compliance”, without clarity what that actually means.
- *Techno*: Actions and tests that look like code, using camel case or underlines, and are therefore often `_NOts0EasY_2REad`, in particular for non-technical users.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

122

122

ABT In Practice



Test Case Design Intensive

BDD Behavior Driven Development

Behavior Driven Development (BDD)

What is BDD?

Developed by Dan North as a way programmers can better understand and spec work to be done.

<https://dannorth.net/introducing-bdd/>

Dan developed BDD as a method to do TDD.

It is not meant for test cases. Gherkins are not meant to be written only by testers as a method for test automation. But- probably the most common use of Gherkins is just that. Testers writing UI automation using Gherkins and calling it BDD.

That aside- we are going to look at documenting test cases using this method.

Behavior Driven Development (BDD)

We need to understand the words commonly used here before we proceed
TDD –test driven development. An XP practice similar to unit testing to drive the development of software thru writing tests specifying what the software is expected to do.

BDD – behavior driven development. Dan North’s practice to do TDD with syntax and tooling.

Behavior Driven Development (BDD)

Cucumber is a software tool that computer programmers use for testing other software. It runs automated acceptance tests written in a behavior-driven development (BDD) style. **Cucumber** is written in the Ruby programming **language**.

Wikipedia.org

Behavior Driven Development (BDD)

What is a gherkin cucumber?

- A pickled **cucumber** (commonly known as a **pickle** in the United States and Canada or generically as **gherkins** in the United Kingdom) is a **cucumber** that has been pickled in a brine, vinegar, or other solution and left to ferment for a period of time, by either immersing the **cucumbers** in an acidic solution or through souring ...

Gherkin is the language that **Cucumber** understands. It is a Business Readable, Domain Specific Language that lets you describe software's behaviour without detailing how that behaviour is implemented. **Gherkin** serves two purposes — documentation and automated tests.

<https://github.com/cucumber/cucumber/wiki/Gherkin>

Behavior Driven Development (BDD)

The things people call "tests" in BDD are usually called
Gherkins
Or
GWT
Given-When-Then

Behavior Driven Development (BDD)

BDD is organized into Feature then Scenario.

Feature:

- Some terse yet descriptive text of what is desired.
- Textual description of the business value of this feature
- Business rules that govern the scope of the feature
- Any additional information that will make the feature easier to understand

Scenario:

- Some determinable business situation

<https://github.com/cucumber/cucumber/wiki/Gherkin>

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

131

131

Behavior Driven Development Given When Then

Given a context

And additional information of context (optional)

When a specific action is performed

Then an expected result

And additional information or expected result (optional)

<https://github.com/cucumber/cucumber/wiki/Gherkin>

The essential idea is to break down writing a scenario (or test) into three sections:

- The **given** part describes the state of the world before you begin the behavior you're specifying in this scenario. You can think of it as the pre-conditions to the test.
- The **when** section is that behavior that you're specifying.
- Finally the **then** section describes the changes you expect due to the specified behavior.

<http://martinfowler.com/bliki/GivenWhenThen.html>

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

132

132

Behavior Driven Development (BDD)

The overall structure looks like:

Feature

 Scenario

 GWT

 Scenario

 GWT

Many teams are using a Parser to divide up these statements and have a tool, like cucumber make Step definitions to automate tests.

Behavior Driven Development (BDD)

Title (one line describing the story)

(User Story/Feature) Narrative:

 As a [role]

 I want [feature]

 So that [benefit]

Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title

 Given [context]

 And [some more context]...

 When [event]

 Then [outcome]

 And [another outcome]...

Scenario 2: ...

<https://dannorth.net/whats-in-a-story/>

Behavior Driven Development (BDD)

Feature: Login

- As a user I can login with my valid username password
- As a user I will see error message with my invalid username password

Scenario: Login with valid username password

- **Given** Login page is navigated
- **When** Type valid username password
- **Then** Homepage is navigated

Scenario: Login with invalid username password

- **Given** Login page is navigated
- **When** Type invalid username password
- **Then** Error message display

Behavior Driven Development And But

If you have several givens, whens or thens you can write

Or you can make it read more fluently by writing

Scenario: Multiple Givens

- Given one thing
- Given another thing
- Given yet another thing
- When I open my eyes
- Then I see something
- Then I don't see something else

• **Scenario:** Multiple Givens

- Given one thing
 And another thing
 And yet another thing
- When I open my eyes
- Then I see something
 But I don't see something else

Behavior Driven Development Background- for a shared condition

Background:

Given some setup
And some condition

Scenario: Some scenario

When a first trigger occurs
Then something good happens

Scenario: Some other scenario

When another trigger occurs
Then something else happens

*these are not supposed to be used for 'set up or 'tear down

<http://morelia.readthedocs.io/en/latest/gherkin.html>

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

137

137

Behavior Driven Development (BDD)

Story: Account Holder withdraws cash

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario 1: Account has sufficient funds

Given the account balance is \ \$100
And the card is valid
And the machine contains enough money
When the Account Holder requests \ \$20
Then the ATM should dispense \ \$20
And the account balance should be \ \$80
And the card should be returned

Scenario 2: Account has insufficient funds

Given the account balance is \ \$10
And the card is valid
And the machine contains enough money
When the Account Holder requests \ \$20
Then the ATM should not dispense any money
And the ATM should say there are insufficient funds
And the account balance should be \ \$20
And the card should be returned

Scenario 3: Card has been disabled

Given the card is disabled
When the Account Holder requests \ \$20
Then the ATM should retain the card
And the ATM should say the card has been retained

<https://dannorth.net/whats-in-a-story/>

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

138

138

Behavior Driven Development (BDD)

Data tables are common and useful.

Given the following users exist:

name	email	twitter
Aslak	aslak@cucumber.io	@aslak_hellesoy
Julien	julien@cucumber.io	@jbpros
Matt	matt@cucumber.io	@mattwynne

<https://cucumber.io/docs/reference>

ABT & BDD Best Uses

ABT and BDD are different. They solve different problems and have different *best uses*. Here are some examples.

- Situation 1 – The scrum team does not fully understand the user stories or product. The user stories are not very detailed or the product has complicated domain specific tasks where the dev team may have only a high level knowledge of the workflow and domain. Use BDD to show behaviors of the system and get Product Owner buy-in. Big bonus for the team.
- Situation 2 – Programmers do no unit tests, testers do all automation and regression. Programmers can write Given-When-Then statements and use a common unit test harness, such as JBehave for automated unit tests. Big bonus for the team.
- Situation 3 – Test team lacks programming skill. Domain knowledgeable team (testers, POs, subject matter experts, business analysts) can write the various test objectives and conditions without concern for how the automation will get executed. An automation engineer using an action based automation tool, like TestArchitect, can automate and support about a dozen domain knowledge people. Big bonus for the team.

ABT & BDD Best Uses

- Situation 4 – A highly skilled, fully Agile team. Programmers can use BDD to better specify the system behavior. Action based test modules and test objectives can be developed at the same time. Once coding starts and you have product to be tested, the BDD will be turned into TDD/unit tests. The ABT tests (bigger, longer, more complex tests) can be automated. Big bonus for the team.
- Situation 5 – Your product has multiple platforms or devices and you need a lot of *high volume* automation. Use ABT to design tests. Since ABT focuses on test design separate from the execution, the same tests can be executed across various platforms. An automation engineer, using a cross-platform tool, can change the implementation of the automation and the bulk of the work-developing the tests- might not change at all. Big bonus for the team.
- Situation 6 – High maintenance cost for test automation. If you have a product that is very often in a state of change and/or constant redesign, your tests are probably high maintenance. Building a reusable, high-volume automated regression suite using a test method focused on low maintenance and high reusability is a must. ABT which focuses on small action level maintenance is best for this. Big bonus for the team.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

141

141

Summary

- Parts of a test case
- Objective is most important
- Example methods:
 - Best Generic
 - ABT
 - BDD

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

142

142

143

Test Case Design Intensive

Chapter 4

More topics on test cases

- Maintenance, Cost
- Traceability
- Automation

Summary

144

Other issues with test case

Maintenance, Cost

- When to write them – JIT, focus on Objective, not “how” to execute yet.
- User stories- acceptance criteria

Automation

- Data driven
- ABT
- JIT

Traceability

- Trace
- Tagging
- Priority

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

145

145

When to write

Increasing levels of details

Detail the actions, steps, procedures and call scripts.

Beware these do not become the early focus of the tests. Keep it low maintenance. Only add low-level details as needed. Practice the *Lean* principle of Just-in-Time, JIT, writing the steps later or not at all is a great cost savings.

A common question about test design is *when*. When should we write our tests? The answer is layered.

Very early in the process, you must define and write the test objectives, that is, establish what it is that you are going to test. This will help you size the testing task, as well as provide a non-traditional type of TDD (test driven development) in that defining what you will test early in the cycle will help developers ensure those tests will pass.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

146

146

When to write

Increasing levels of details

Test design is an ongoing process.

At the beginning of a sprint, the test objectives can be authored and detailed.

Add more, lower-level details *just-in-time* to lower maintenance.

The actions can be defined, but at this point it is less effective to get low level step-by-step details defined since the function is not yet built, and the UI is not defined.

This refocusing away from test scripts is a big shift for some testers.

Low-level details for tests should not be documented during the initial development when it stands a better chance of needing to be redesigned, thereby raising the maintenance costs.

More ideas on Maintenance

Low Maintenance/High reusability

– just like with code

What is design for low maintainability?

Make the tests easy to write and read. Use a natural, domain-specific language.

Start high-level and build detail over time as the functions get delivered.

Define actions or keywords, but not execution steps to exercise the task or workflow.

Don't get lost in details early.

Agile

Agile

In this agile era it is sadly, too common a practice that test teams under pressure for time focus on low-level tests. They don't take a step back to look at the bigger picture and see test design from a global perspective. Agile testing is still in development. It does not focus solely on low-level user story acceptance criteria. These tests are too granular to get a great customer experience for the user doing end-to-end scenarios and workflows. It is important to remember that testing early in a sprint is not the same as testing later in a sprint, or in future sprints. The tests have to change, increasing in complexity and size.

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

149

149

Agile

Agile is...

- Iterative
- Emergent
- Collaborative
 - Relies on constant and open communication
- Based on feedback

Not every detail will be known before the Sprint starts!

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

150

150

Acceptance Criteria: The key

Acceptance criteria

1. Test cases
2. Verification points
3. Test conditions

- Don't get low level early.
- Wait on details as late as possible

Agile Artifacts

When are Acceptance Criteria Written?	By whom?
Creating the User Story	PO
Release Planning & Sprint Planning	PO & Team
During Sprint	Team (Test Team, Programmers, PO)
Writing Test Cases	Testers

Requirement Traceability

For many companies the goal of doing Requirements-based Testing (including User Stories and any documentation to validate against) is to have an assessment of test or functional coverage, for regulatory compliance, or to focus the test team on a narrow slice of the testing effort.

This is not as easy as it seems. To not dwell on the subject, with this as our goal we can only test incrementally better than the requirements as they are written.

Requirement Traceability

Traces the linkage between the functional requirements to the software products including design and code and reports on the percentage requirements traced from the function to the test case.

Excel Example

Requirement	Functional Specification	Design Specification	Code section	Test Condition	Defect ID
1	1.1	3.1	Line 300-350	12	D2
				13	-
				14	D6
	1.2	3.2	Line 450-600	21	-
				22	-
				23	-
Requirement	Functional Specification	Test Condition	Test Case	Test Script	Defect ID
1	1.1	12	12.1/12.2	T5	D2
		13	13.1/13.2/13.3	T6	-
		14	14.1	T6	D6
	1.2	21	21.1	T8	-
		22	22.1/22.2	T8	-
		23	23.1/23.2/23.3	T8	-

Automating Traceability of Requirements to Test Case & Defect Logs - a case study
 Krishna Rajan, T.A.Indira

Automation

We have been talking about automation throughout the workshop.

A few things are clear:

- Test design is a different practice than modern test automation done by different people.
- Separating test design from test automation enables teams to get better tests and scale more.
- Test design is clearly more important.

Leverage modern tools.

Share across the organization

Use Lean principles

Summary

Most important: design good tests. And keep it Lean.

It makes sharing, maintenance and automating easier.

Excellent and simple test design is essential for confidence in your testing job, meaningful coverage, making sure tests are run correctly and finding bugs!

About *LogiGear* Corporation

LogiGear Corporation provides testing expertise and resources to software development organizations. Our partners benefit from our seasoned testing staff and facilities, practical training programs, and test support products. We help development teams deliver high quality software, improve time-to-market, and optimize development productivity.

Founded in 1994 as *softGear* technology, *LogiGear* has built a reputation on partnering with software development organizations to help make the most of outsourcing and staff training solutions. We assist our clients in delivering the best possible quality products while juggling limited resources and schedule constraints.

Copyright © 95-19 *LogiGear* Corporation. All Rights Reserved.

159

159

About Michael Hackett

MICHAEL HACKETT, Director, Training and Publications, has over 20 years of experience in software engineering and the testing of shrink-wrap and Internet based applications. He has developed for Windows, Macintosh and UNIX operating systems. Michael has helped well-known companies including Palm Computing, Electronics for Imaging, Adobe Systems, CNET, The Learning Company, Power Up Software, Oracle, PC World, ADP, The GAP and The Well produce, test and release applications ranging from business productivity to educational multimedia titles in English as well as a multitude of other languages.

Michael is a founding partner of *LogiGear* Corporation. Prior to joining *LogiGear*, he served as Director of Quality Assurance at The Well, an online service that is renowned for its electronic conferencing system. Michael has developed professional training courses dealing in engineering, business communication and computer training. He has also written many instructional manuals used by professional trainers.

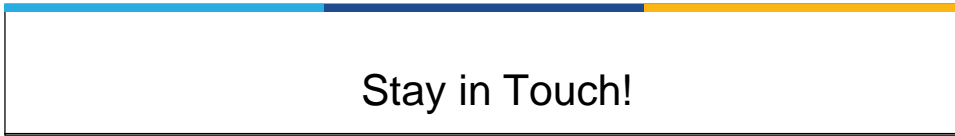
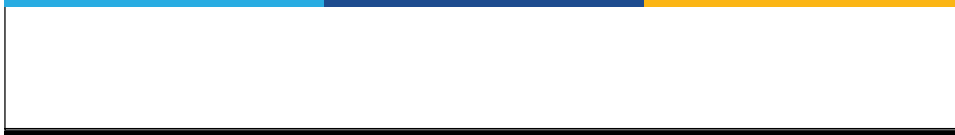
He is the co-author of *Testing Applications on the Web* published by Wiley. Michael holds a Bachelor of Science in Engineering from Carnegie-Mellon University.

michaelh@logigear.com

Copyright © 95-19 *LogiGear* Corporation. All Rights Reserved.

160

160

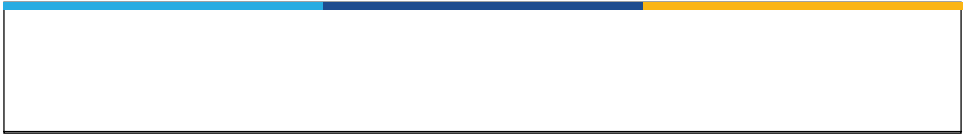


Stay in Touch!

- Email me at MichaelH@logigear.com
- Connect with me on LinkedIn:
<https://www.linkedin.com/in/michael-hackett-a0575b2>

Thank You!

- Thank you for joining us today. To learn more about LogiGear and our continuous testing services visit:
<http://www.logigear.com/solutions/continuous-testing-solution.html>
- *Visit LogiGear Magazine!*



24 Years of TESTING & DEVELOPMENT EXCELLENCE

LogiGear USA - Headquarters
4100 E 3rd Ave, Suite 150
Foster City, CA 94403
Tel: +1 650 572 1400
Fax: +1 650 572 2822

LogiGear USA - Houston
10777 Westheimer Suite 1000
Houston, TX 77042
Tel: +1 650 572 1400 ext. 380
Fax: +1 281 288 9088

LogiGear USA - Seattle
2225 N 56th St
Seattle, WA 98103
Tel: +1 650 572 1400
Fax: +1 650 572 2822

LogiGear Viet Nam - Headquarters
1A Phan Xich Long, Ward 2
Phu Nhuan District
Ho Chi Minh City
Tel: +84 839 954 072
Fax: +84 839 954 076

LogiGear Viet Nam - Da Nang City
346 Street 2/9
Hai Chau District
Da Nang City
Tel: +84 511 365 533
Fax: +84 839 954 076

Exercise

Validating a Credit Card Number, Expiration Date, Code

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

165

165

Credit Card Validation Example

Background

App processes credit card.
Visa/MasterCard only
Valid credit card has 16 digits, expiration date, security code

User Story

As a customer
I want to enter my credit card data
So I can make a purchase.

Acceptance Criteria- Credit card Number

A valid credit card has 16 digits
An invalid number has 16 digits but is not a correct card number. Example, all 0s
Less than or greater than 16 digits throws an error
Blank throws a "mandatory field" error
Alpha or other non-numeric characters do not work.
Copy & Paste is disabled

Sample Data

1234 5678 1234 5678 (16 digit valid)
0000 0000 0000 0000(16 digit invalid)
ABCD 1234 5678 1234 (contains alpha char)
**&^% 5678 1234 5678 (contain special char)*
1234 5678 1234 567 (15 digits)
Blank (mandatory field)

Copyright © 95-19 LogiGear Corporation. All Rights Reserved.

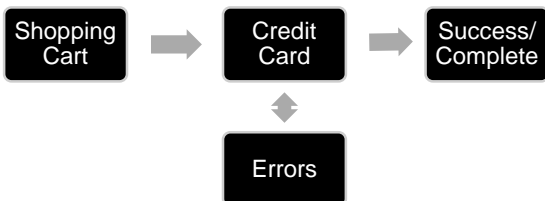
166

166

Credit Card Example

Credit Card validation

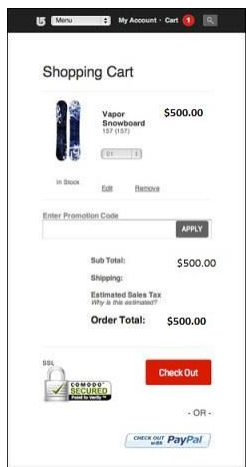
Focus only on the card number validation form.



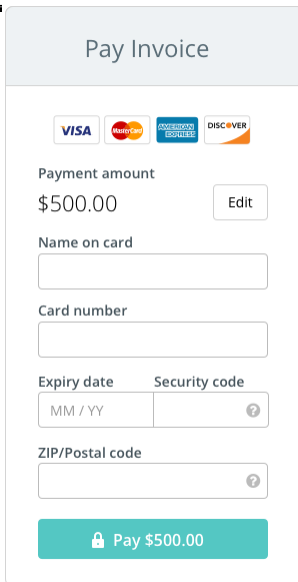
167

Credit Card Example

For Navigation only




168




Credit Card Example

Pay invoice



Payment amount
\$500.00 Edit

Name on card


Card number
 

Expiry date Security code
 ?

ZIP/Postal code
 ?

🔒 Sending...

✕



\$500.00

Your payment is complete.

Verification code:
LJ986JJG18656JKS3509GKCQIU

View receipt


Copyright © 95-19 LogiGear Corporation. All Rights Reserved. 169

169

Credit Card Example

Errors show on form.

Pay invoice



Payment amount
\$500.00 Edit

Name on card

Card number
 Enter a 16-digit card number

Expiry date Security code
 ?

ZIP/Postal code
 ?

🔒 Pay \$500.00

Copyright © 95-19 LogiGear Corporation. All Rights Reserved. 170

170

Exercise: Modules

First exercise: Define modules

How would you begin to organize your tests? Start High level.

Credit Card Test Objective

Test Objective Style 1

Action+ Function+ Operating Condition

Process card number with valid 16 digit number

Example Test Cases: Tear Down

Tear-down- post conditions- clean up

Example Test Cases: Scenarios

What are some possible scenarios?

Example Test Cases: Scenarios

What are some possible scenarios?

- Black Friday
- Over credit limit
- Card not yet authorized
- First time use
- After a failed attempt
- Time out
- Lost connection

Remember, in ABT, how these would get categorized is important.

Test Objective Ideas

Test Case Style 1

Action+Function+Operating Condition

Process card number with valid 16 digit number

Generate invalid card number error message with invalid 16 digit number

Generate valid card characters between 0 – 9 only message by entering alpha, special characters and any other non-numeric characters.

Generate enter a valid card number error with fewer than 16 digit number

Generate please enter a valid card number message by leaving number field blank

Test Case Style 2

Test Module

Cause Effect

Credit Card Number Field Validation

Enter correct card number to process

Enter invalid card number to generate error message #1

Enter Alpha characters to generate error message #2

Enter less than 16 char generates error message #2

Leave field blank generates error message #3

Test Objective Ideas

Test Case Style 3

Scenario: Enter Credit card Number
GWT

Valid

Given the user has a valid credit card
And the user is on the credit card screen
When they enter a valid credit card number
Then the card processes

Invalid

Given the user has a valid credit card
And the user is on the credit card screen
When they enter an invalid credit card number
Then an error message #1 appears

Alpha or special characters

Given the user has a valid credit card
And the user is on the credit card screen
When they enter alpha or special characters
Then an error message #2 appears

15 digits

Given the user has a valid credit card
And the user is on the credit card screen
When they enter less than 15 digits
Then error message #2 appears

Blank

Given the user has a valid credit card
And the user is on the credit card screen
When they leave the field blank
Then an error message #3 appears

181

Test Objective Example

<p>Background App processes Visa/MasterCard credit card. Valid credit card has:</p> <ul style="list-style-type: none"> •16 digits •Expiration date •Security code <p>Acceptance Criteria—Credit Card Number</p> <ul style="list-style-type: none"> •A valid credit card has 16 digits •An invalid number has 16 digits but is not a correct card number. Example, all 0s •Less than or greater than 16 digits throws an error •Blank throws a "mandatory field" error •Alpha or other non-numeric characters do not work. •Copy & Paste is disabled 	<p>User Story</p> <p>As a customer I want to enter my credit card data So I can make a purchase.</p> <p>Sample Data</p> <ul style="list-style-type: none"> •1234 5678 1234 5678 (16 digit valid) •0000 0000 0000 0000(16 digit invalid) •ABCD 1234 5678 1234 (contains alpha char) •*%& 5678 1234 5678 (contain special char) •1234 5678 1234 567 (15 digits) •Blank (mandatory field)
--	--

<p>Test Objective Style 1 Action + Function + Operating Condition</p> <p>Test Case Suite: Credit Card Number Field Validation</p> <ul style="list-style-type: none"> • Process card number with valid 16 digit number • Generate invalid card number error message with invalid 16 digit number • Generate valid card characters between 0 - 9 only message by entering alpha, special characters and any other non-numeric characters • Generate enter a valid card number error with fewer than 16 digit number • Generate please enter a valid card number message by leaving number field blank. 	<p>Test Objective Style 3 Behavior Driven Development: Given-When-Then</p> <p>Scenario—Enter Credit Card Number</p> <div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> <p>Valid</p> <ul style="list-style-type: none"> •Given the user has a valid credit card •And the user is on the credit card screen •When they enter a valid credit card number •Then the card processes </div> <div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> <p>Invalid</p> <ul style="list-style-type: none"> •Given the user has a valid credit card •And the user is on the credit card screen •When they enter an invalid credit card number •Then an error message #1 appears </div> <div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> <p>Alpha or special characters</p> <ul style="list-style-type: none"> •Given the user has a valid credit card •And the user is on the credit card screen •When they enter alpha or special characters •Then an error message #2 appears </div> <div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> <p>15 digits</p> <ul style="list-style-type: none"> •Given the user has a valid credit card •And the user is on the credit card screen •When they enter less than 15 digits •Then error message #2 appears </div> <div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> <p>Blank</p> <ul style="list-style-type: none"> •Given the user has a valid credit card •And the user is on the credit card screen •When they leave the field blank •Then an error message #3 appears </div>
--	--

<p>Test Objective Style 2 Action Based Testing: Cause and Effect</p> <p>Module: Credit Card Number Field Validation</p> <ul style="list-style-type: none"> • Enter correct card number to process • Enter invalid card number to generate error message #1 • Enter Alpha characters to generate error message #2 • Enter less than 16 char generates error message #2 • Leave field blank generates error message #3 	
--	--

182