

Test Automation for Mobile Devices

A SIPTECH CASE STUDY

Version: 1.03

Created Date: 24.Dec.2004

Inside this document ...

01. Background
02. Problem / Challenges
03. Method / Intervention / Solution
04. Results Achieved
05. Lessons Learned / Conclusion



© 2005 All rights reserved. SIP Technologies and Exports Ltd. The information contained in this document is CONFIDENTIAL and PROPRIETARY in nature, and subject to the rights and ownership of SIPTECH. Any and all unauthorized copying or use of the contents hereof is prohibited.

1. Background

A large mobile handset manufacturer requested SIPTECH to automate tests for messaging applications targeted for its next generation mobile platform. The competition in the mobile devices segment is very high; there is an ever-growing need for different variant of the devices to fulfill the needs of the consumers. Hence most of the manufacturers develop and maintain a platform; based on the market requirements new variants of the mobile devices are released.

2. Problem / Challenge

The lifecycle used for developing the mobile platform was an iterative incremental lifecycle model, with each of the iterations spanning four weeks. Hence every four weeks new builds are promoted which need regression testing. The number of use case scenarios for messaging application was estimated to be 4750+. The messaging application would be developed spanning over 20+ iterations.

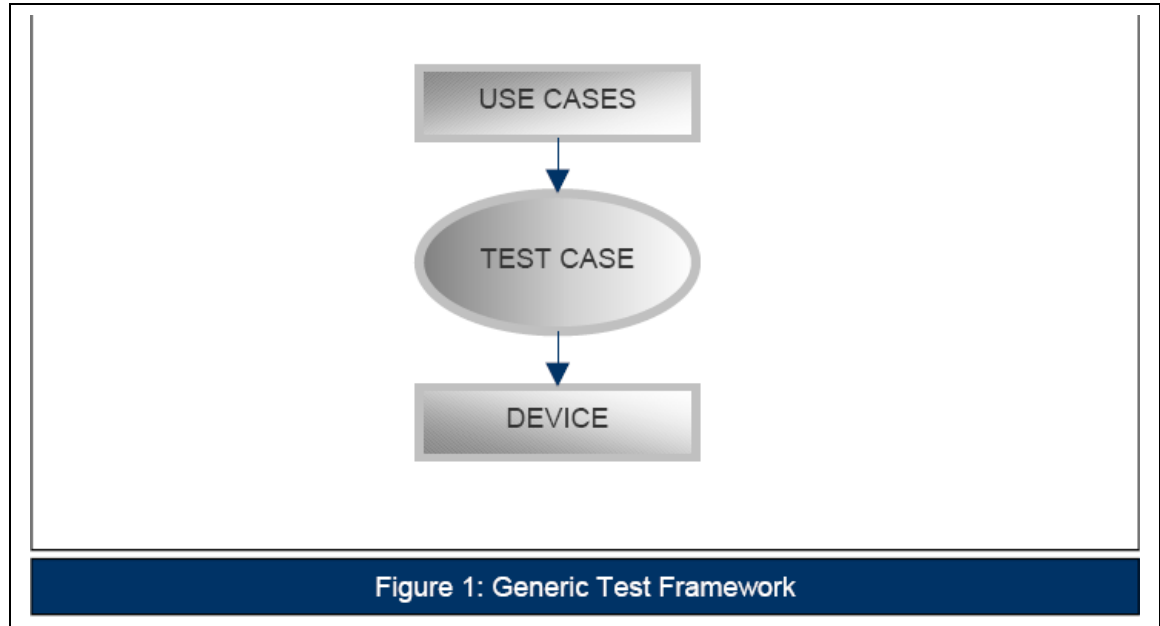
The effort required to test the messaging application manually would be enormous as regression tests needs to be performed for 20+ iterations across 4500+ test cases, the need for test automation is justified. More over there would be a number of models of mobile devices based on the platform and there would be differences in the UI of these devices. The automated test cases should support all these variances.

The goal of this task is to automate tests for messaging applications, and the automated tests should be easy to maintain and have the capability to test different variants of the mobile device.

3. Method / Intervention / Solution used

Use Case based Test Case

Most of the Test frameworks implement test cases as discrete programs. These test-cases are grouped to build a suite. There is a one to one mapping of the test-case to the use-case scenarios. Based on the customer requirements the Use cases and scenarios are identified. For each of the use case scenarios test cases are defined.



Example:

For each of the use case scenarios separate test cases are created, a sample scenario is as listed below.

Use Case Scenario: Create Email

Test Case: The complete implementation of a use case scenario is in a discrete test case

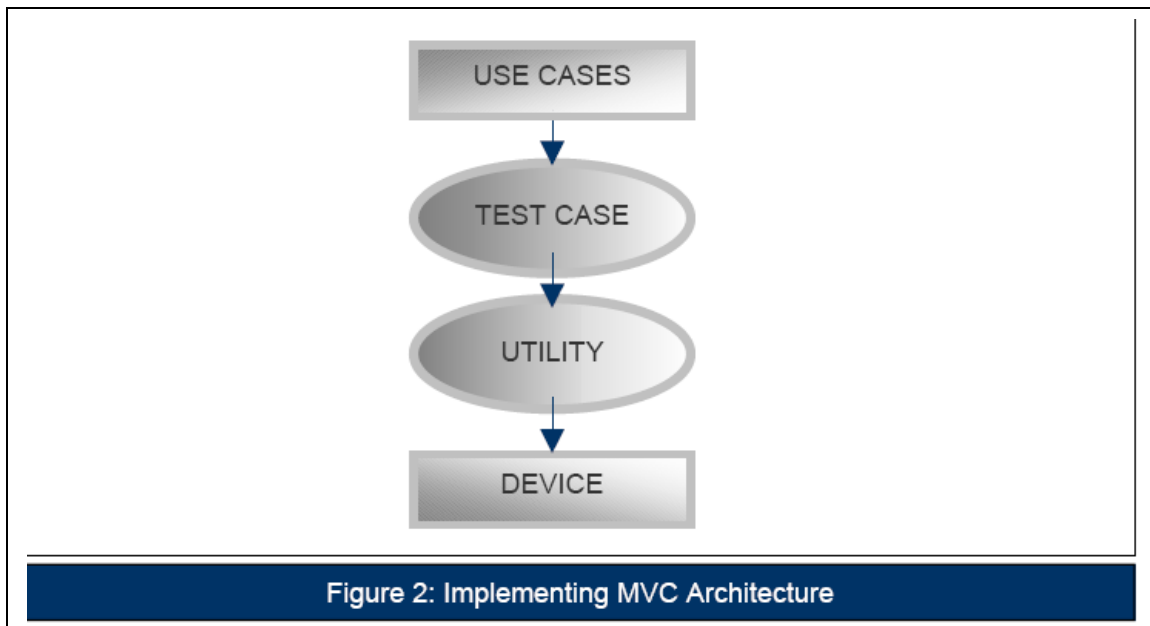
1. Select the Centre key
2. Select the Left soft key
3. Scroll down
4. Press Centre Key (to launch Email Application)
5. Press the Left soft key
6. Scroll and select "Create Email" option.
7. Check if " Create email" screen is displayed

Advantages:

By having discrete test-cases for use case scenarios it is easier to manage the test suite.

Implementing MVC Architecture

The architecture used is an extension of Model, View and Controller. Here the test cases act as Controllers and Utility Functions act as models. The test cases controllers invoke the multiple utility functions to perform a test. The Utility function contains the device specific source code.



Example

This test case acts as a controller and invokes appropriate utility functions to perform the task.

Use case Scenario: Create Email.

Test Case: Here the test case act as a controller and invokes the utility function where in the actual implementation is present.

1. Invoke utility function "Launch Application - Email"
2. Invoke utility function "Select Option - Create Email"
3. Invoke utility function "Check Screen - Create Email"

Utility Function 1: Launch Application – Email

1. Select the Centre key
2. Select the Left soft key
3. Scroll down
4. Press Centre Key

Utility Function 2: Select Option - Create Email

1. Press the Left soft key

2. . Scroll and select "Create Email" option.

Utility Function 3: Check Screen - Create Email

1. Check Screen Title for text "Create Email"

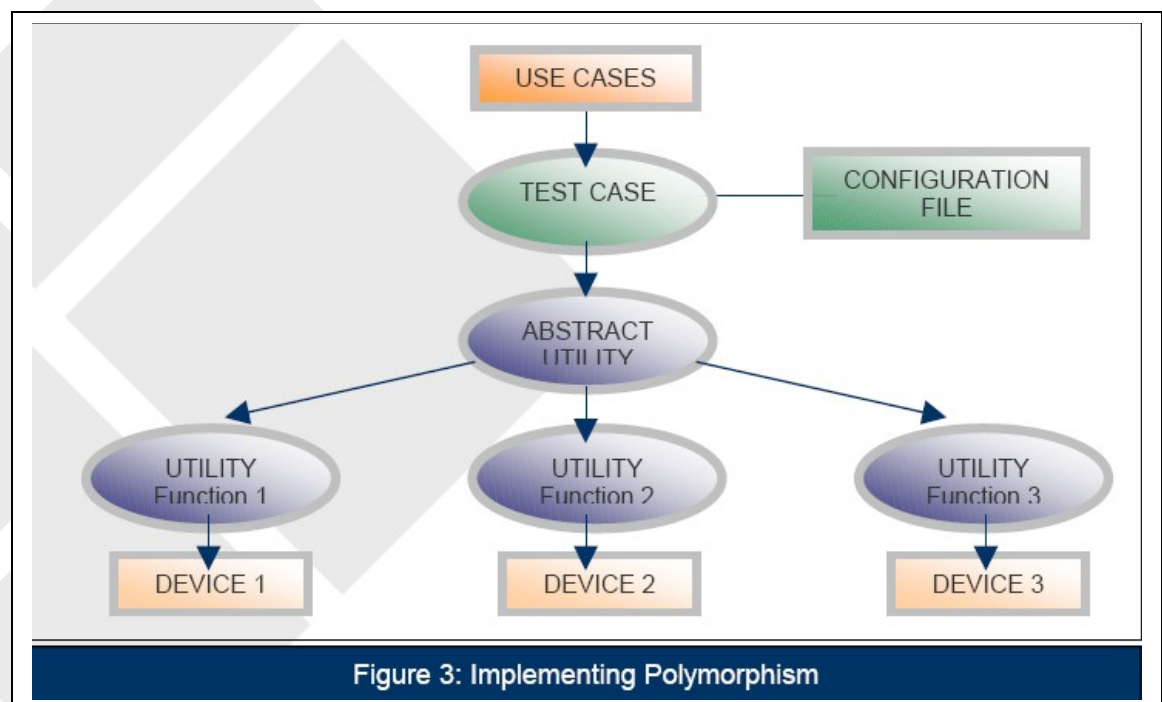
Advantages:

The Utility functions can be re-used. If there is a change in the navigation path or the type of UI component, only the utility function has to be updated. This makes the test suite easy to maintain. Please note the number of test cases for messaging application is estimated to be 4500+ and for the complete platform to be ~40,000 test cases.

Implementing Polymorphism

As mentioned earlier a mobile platform will be implemented across different models of mobile handset. And these would be variances in UI such as resolution, navigation path, language, etc. To test all these variants one of the options is to have discrete set of test case and utility functions for each model of the handset. This is a redundant and cumbersome approach. By utilizing polymorphism and bringing in a layer of abstraction the source codes could be re-used.

The implementation to have a test case per use case scenario; a number of utility function implementations address the UI variations. The framework can be built with sufficient flexibility to get user input to use the appropriate utility functions while executing the test-cases against a specific platform or the device.



Example

There is single Test Case, which is used for testing use case scenarios across the devices. However there exists multiple Utility Functions depending on the UI variations in the devices. The test case under execution refers a configuration file where in the Utility Function implementation to be used for the particular device is defined.

Based on which the test case utilizes appropriate Utility Function while testing.

Use case Scenario: Create Email.

Test Case: Here there is no change as compared to the previous example in page 8

1. Invoke utility function "LaunchApplication - Email"
2. Invoke utility function "Select Option - Create Email"
3. Invoke utility function "Check Screen - Create Email"

Utility Function: As per the variation in UI of the hand sets a number of Utility functions would be implemented. The framework has sufficient flexibility to configure and select the appropriate implementation of utility function as per the devices on which the test cases are executed.

Utility Functions - Device 1	Utility Functions - Device 2
<p>Launch Application - Email</p> <ul style="list-style-type: none"> ❖ Select the Centre key ❖ Select the Left soft key ❖ Scroll down ❖ Press Centre Key <p>Select Option - Create Email</p> <ul style="list-style-type: none"> ❖ Press the Left soft key ❖ Scroll and select "Create Email" option. <p>Check Screen - Create Email</p> <ul style="list-style-type: none"> ❖ Check Screen Title for text "Create Email" 	<p>Launch Application - Email</p> <ul style="list-style-type: none"> ❖ Select the Centre key ❖ Select the Left soft key ❖ Scroll down ❖ Scroll down ❖ Scroll down ❖ Press Centre Key <p>Select Option - Create Email</p> <ul style="list-style-type: none"> ❖ Press the Left soft key ❖ Scroll and select "Create Email" option. <p>Check Screen - Create Email</p> <ul style="list-style-type: none"> ❖ Check Screen Title for text "Create Email"

Advantages:

The advantage of this architecture is a single set of test-case is required to test various devices for a given scenarios. As and when the new devices need to be tested only the Utility Function has to be developed / upgraded so as to build a test suite for the same.

4. Results Achieved

In the Test-cases developed which contain 25000 lines of code, there are 40 utility functions which contain 5000 lines of code. When Test-cases for new model of device need to be developed using this architecture it is necessary only to modify / update the Utility functions and not the test-cases.

By modifying the utility functions the test cases were running on newer version of the platform. When there was a need to test newer version of the platform, the utility functions were created / updated to support 150 Test Cases. The actual effort was only 25% of the effort otherwise would have been spent.

5. Lessons Learned / Conclusion

By implementing the above frameworks while developing suites for subsequent mobile platform or devices ~75% of the effort could be reduced. This architecture is more suitable for product / platform testing when different versions are released in quick succession with changes in GUI while the functionality is retained.