Simple Habits

Master
Test Plan

Version 1.4

Revision History

Date	Version	Description	Author
23/04/2017	1.0	Creation	Benedikt Bosshammer
24/04/2017	1.1	Basic Information	Benedikt Bosshammer
04/05/2017	1.2	Test Configuration	Benedikt Bosshammer
08/05/2017	1.3	Additions to Test Configuration	Benedikt Bosshammer
18/06/2017	1.4	UI-Tests linked	René Penkert

Table of Contents

Master Test Plan

1. Introduction

1.1 Purpose

The purpose of the Iteration Test Plan is to gather all of the information necessary to plan and control the test effort for a given iteration. It describes the approach to testing the software, and is the top-level plan generated and used by managers to direct the test effort.

This *Test Plan* for the Alarm Clock supports the following objectives:

- Decrease the number of bugs to ensure trouble-free usage
- Provide an easy and comfortable use to the users

1.2 Scope

Our scope of the testing consists of the following two parts:

- Integration Testing with Xcode Server and Xcode Unit-Tests
- Xcode User Interface-Tests

In addition, we want to the test the user interface with end users before the first release.

1.3 Intended Audience

This document is intended for advanced readers. It’s a technical document that doesn’t describe the application for the users and should only be read by developers.

1.4 Document Terminology and Acronyms

n/a

1.5 References

n/a

2. Evaluation Mission and Test Motivation

Testing is done to guarantee that the software is stable and furthermore stays stable over the development of new features and bug fixes.

2.1 Background

By using continuous tests, we can monitor the effects that changes to the source code combined with user interactions cause to the functionality and performance of the app.

As a result, we can:

1. Ensure that a change do what it’s supposed to do.

   Testing guarantees that new implementations work as intended and detects possible conflicts between the new and old code.

2. Catch all possible edge cases.

   No developer can ever think of all possible combination of user interactions possible in his system to still catch all possible bugs. User Interface-Test Scenarios cover all this cases.

2.2 Evaluation Mission

With Testing being one of the critical parts of a software project, it is necessary in order to exterminate technical bugs and give assurance of their absence to the stakeholders.
By covering not only Unit-Tests but also UI-Tests and Integration Test we cover as much possible issues as possible in an iOS Project.

2.3 Test Motivators

Due to the number of components and complexity the project has, we have to maintain and refactor our code base on a regular basis. It is important that we prepare tests for every new Usecase before starting the development process to:

- Reduce technical risks.
- Fulfill functional and no-functional requirements
- Realize the Usecase faster

3. Test Approach

3.1 User Interface Testing

Technique Objective:	Navigation through the user interface of the app
Technique:	Manually implement UI-Tests testing User Interactions (tabs, keyboard inputs, navigation, menu) executed by Xcode's UI-Test environment in the simulator
Oracles:	Manually executed with the help of Xcode
Required Tools:	Xcode's UI-Test environment
Success Criteria:	All test pass successfully.
Special Considerations:	n/a

3.1.1 Link to tests

Create Alarm
Weather API
Read Alarms
Complete CRUD
Select OS Calendar

3.2 Unit-Testing

Technique Objective:	Make sure that the new features work properly
Technique:	Manually implement tests for the new coding
Oracles:	Manually executed with the help of Xcode
Required Tools:	Xcode's Unit-Test environment
Success Criteria:	All test pass successfully. We strive for 40% tests coverage.
Special Considerations:	Not everything can be covered with unit tests!

3.2.1 Link to tests

Core Data
Travel API
Travel Calculation
Weather API

3.3 Testing with end user

Technique Objective:	Testing the user experience of the app.
Technique:	Work out test scenarios.
Oracles:	The test users love the experience of usage. Our app is easy to understand and self-explaining.

Required Tools:	A Device capable of navigating and interacting with the app (iPhone 5 - 8).
Success Criteria:	Users confirm our goals of efficiency and ease of use.
Special Considerations:	Is done manually by asking a different person to use and test the app.

4. Entry and Exit Criteria

4.1 Test Plan

4.1.1 Test Plan Entry Criteria

This Test Plan begins as soon as the development is setup correctly and all Use Cases are defined properly.

4.1.2 Test Plan Exit Criteria

If the work on the project "Simple Habits" ends, there is no further need for a Test Plan. This can either occur after a successful release or because the project got cancelled.

5. Deliverables

5.1 Test Evaluation Summaries

The Test Log is visible on our Mac OS Server Report.

5.2 Reporting on Test Coverage

The Test Coverage is visible on our Mac OS Server Report.

5.3 Perceived Quality Reports

For Source Code Quality-Measurement we are using the Codacy service to show the code quality as a badge in our GitHub project.

6. Testing Workflow

All tests can be run manually by every developer from out of the IDE.

7. Environmental Needs

7.1 Base System Hardware

The following table lists the system resources for the test effort presented in the Test Plan.

Resource	Quantity	Name and Type
iPhone	2	Test deployment device
iMac / MacBook	2	Local test Macs running latest Mac OS X
Mac	1	Running Mac OS Server and Xcode for integration testing

7.2 Base Software Elements in the Test Environment

The following base software elements are required in the test environment for this *Test Plan*.

Software Element Name	Type and Other Notes
Apple iOS	Operating System
Xcode	Local Test Runner / IDE
Mac OS Server	Test Server OS

7.3 Productivity and Support Tools

The following tools support the test process for this Test Plan.

Tool Category or Type	Tool Brand Name	Vendor or In-house
Test Management	Mac OS Server	Vendor
Issue Tracking	JIRA	Vendor
Source Code Management	GitHub	Vendor
Metrics	Taylor, TaylorParser	Vendor, Inhouse

8. Responsibilities, Staffing, and Training Needs

8.1 People and Roles

This table shows the staffing assumptions for the test effort.

Human Resources		
Role	**Minimum Resources Recommended** (number of full-time roles allocated)	**Specific Responsibilities or Comments**
Test Manager		Provides management oversight. Responsibilities include: • planning and logistics • agree mission • identify motivators • acquire appropriate resources • present management reporting • advocate the interests of test • evaluate effectiveness of test effort
Test Analyst		Identifies and defines the specific tests to be conducted. Responsibilities include: • identify test ideas • define test details • determine test results • document change requests • evaluate product quality

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
Test Designer		Defines the technical approach to the implementation of the test effort. Responsibilities include: • define test approach • define test automation architecture • verify test techniques • define testability elements • structure test implementation
Tester		Implements and executes the tests. Responsibilities include: • implement tests and test suites • execute test suites • log results • analyze and recover from test failures • document incidents
Test System Administrator		Ensures test environment and assets are managed and maintained. Responsibilities include: • administer test management system • install and support access to, and recovery of, test environment configurations and test labs
Database Administrator, Database Manager		Ensures test data (database) environment and assets are managed and maintained. Responsibilities include: • support the administration of test data and test beds (database).
Designer		Identifies and defines the operations, attributes, and associations of the test classes. Responsibilities include: • defines the test classes required to support testability requirements as defined by the test team

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
Implementer		Implements and unit tests the test classes and test packages. Responsibilities include: • creates the test components required to support testability requirements as defined by the designer

8.2 Staffing and Training Needs

This section outlines how to approach staffing and training the test roles for the project.

9. Iteration Milestones

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
> 40% Test Coverage	(Semester 2, Week 4) 30.04.2017	30.04.2017	(Semester 2, Week 8) 10.06.2017	13.06.2017
Unit test implemented			(Semester 2, Week 4) 30.04.2017	30.04.2017
User Interface Tests implemented			(Semester 1, Week 6) 30.04.2017	(Semester 1, Week 6)
End User Tests prepared and executed				

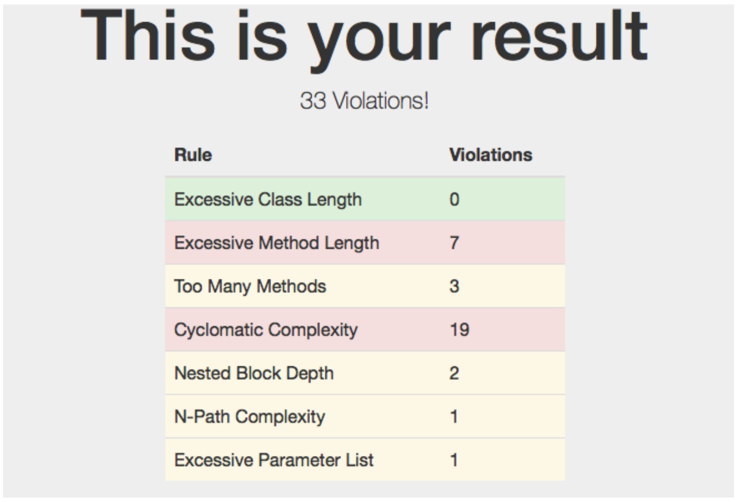
10. Metrics Analysis

As we already described before we use a combination of a vendor tool (Taylor) and our own tool (TaylorParser) which is available on npm for metrics measurement.
Our measurement looks for violations of the following metrics:

- Excessive Class Length
- Excessive Method Length
- Too Many Methods
- Cyclomatic Complexity
- Nested Block Depth
- N-Path Complexity
- Excessive Parameter List

We don’t use automated deployment due to the fact that deployment means publishing to the app store in our case. That means that our too doesn’t not is a part of the deployment process.

On May 18th we ran our tools and got the following result:



This is your result

33 Violations!

Rule	Violations
Excessive Class Length	0
Excessive Method Length	7
Too Many Methods	3
Cyclomatic Complexity	19
Nested Block Depth	2
N-Path Complexity	1
Excessive Parameter List	1

As a reaction (HW: Semester 2, Week 7) we changed the number of methods of the Class *AlarmListViewController*. (details)
More details information can be gathered from our blog post concerning this topic.