
nappydevelopment

**Nappy, the ingenious
Testplan**

Revision History

Date	Version	Description	Author
15/05/2016	1.0	First version	Mehmet Ali Incekara
23/05/2016	1.1	Fill document	Mehmet Ali Incekara and Marvin Zerulla
30/05/2016	1.2	Update document	Mehmet Ali Incekara
03/06/2016	1.3	Add references	Mehmet Ali Incekara
06/06/2016	1.3	Update design, add pictures	Mehmet Ali Incekara

Table of Contents

1.	Introduction	5
1.1	Purpose.....	5
1.2	Scope.....	5
1.3	Intended Audience.....	5
1.4	Document Terminology and Acronyms.....	5
1.5	References	5
1.6	Document Structure	5
2.	Evaluation Mission and Test Motivation	6
2.1	Background.....	6
2.2	Evaluation Mission.....	6
2.3	Test Motivators.....	6
3.	Target Test Items.....	6
4.	Outline of Planned Tests	6
4.1	Outline of Test Inclusions	6
4.2	Outline of Other Candidates for Potential Inclusion	6
4.3	Outline of Test Exclusions	6
5.	Test Approach	7
5.1	Initial Test-Idea Catalogs and Other Reference Sources	7
5.2	Testing Techniques and Types	7
5.2.1	Data and Database Integrity Testing	Fehler! Textmarke nicht definiert.
5.2.2	Function Testing	7
5.2.3	Business Cycle Testing	Fehler! Textmarke nicht definiert.
5.2.4	User Interface Testing	7
5.2.5	Performance Profiling	Fehler! Textmarke nicht definiert.
5.2.6	Load Testing	Fehler! Textmarke nicht definiert.
5.2.7	Stress Testing	Fehler! Textmarke nicht definiert.
5.2.8	Volume Testing	Fehler! Textmarke nicht definiert.
5.2.9	Security and Access Control Testing.....	Fehler! Textmarke nicht definiert.
5.2.10	Failover and Recovery Testing	Fehler! Textmarke nicht definiert.
5.2.11	Configuration Testing	Fehler! Textmarke nicht definiert.
5.2.12	Installation Testing	8
6.	Entry and Exit Criteria.....	9
6.1	Test Plan.....	9
6.1.1	Test Plan Entry Criteria.....	9
6.1.2	Test Plan Exit Criteria.....	9
6.1.3	Suspension and Resumption Criteria	9
6.2	Test Cycles	9

7.	Deliverables	9
7.1	Test Evaluation Summaries	9
7.2	Reporting on Test Coverage	9
7.3	Perceived Quality Reports.....	9
7.4	Incident Logs and Change Requests	9
7.5	Smoke Test Suite and Supporting Test Scripts	9
7.6	Additional Work Products	9
8.	Testing Workflow	10
9.	Environmental Needs.....	11
9.1	Base System Hardware	11
9.2	Base Software Elements in the Test Environment.....	11
9.3	Productivity and Support Tools	11
9.4	Test Environment Configurations	11
10.	Responsibilities, Staffing, and Training Needs	12
10.1	People and Roles	12
10.2	Staffing and Training Needs.....	13
11.	Iteration Milestones	13
12.	Risks, Dependencies, Assumptions, and Constraints	13
13.	Management Process and Procedures.....	13
13.1	Measuring and Assessing the Extent of Testing	13
13.2	Assessing the Deliverables of this Test Plan	13
13.3	Problem Reporting, Escalation, and Issue Resolution.....	13
13.4	Managing Test Cycles	13
13.5	Traceability Strategies	13
13.6	Approval and Signoff	13
14.	Appendix	Fehler! Textmarke nicht definiert.
14.1	Metrics	Fehler! Textmarke nicht definiert.
14.2	Installation tests.....	Fehler! Textmarke nicht definiert.

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 Nappy, the ingenious supports the following objectives:

- The tests are aligned with the interface and functionality.
- The motivation is to develop an application with the fewest number of bugs.

1.2 Scope

This document addresses the following types and levels of testing:

- JUnit tests
- Functional tests
- Blackbox TestFX tests
- JUnit like GUI tests with TestFX
- Installation test

1.3 Intended Audience

The intended audience of this test plan are project members and stakeholder.

1.4 Document Terminology and Acronyms

n/a

1.5 References

SikuliX – Directory	https://github.com/nappydevelopment/docs/tree/master/sikulix
JUnit + GUI – Directory	https://github.com/nappydevelopment/Nappy-the-ingenious/tree/master/src/test/java/nappydevelopment/nappyTheIngenious
Codacy	https://www.codacy.com/app/NappyDevelopment/Nappy-the-ingenious/dashboard
Coveralls	https://coveralls.io/github/nappydevelopment/Nappy-the-ingenious?branch=master
SonarQube	http://193.196.7.25/overview?id=5235
Installation Test – Jani	https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Jani.pdf
Installation Test – Tom	https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Tom.pdf
Installation Test – Samuel	https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Samuel.pdf
SAD	https://github.com/nappydevelopment/docs/blob/master/pdfs/Software%20Architecture%20Document.pdf
SRS	https://github.com/nappydevelopment/docs/blob/master/pdfs/Software%20Requirements%20Specification.pdf

1.6 Document Structure

n/a

2. Evaluation Mission and Test Motivation

2.1 Background

Testing supports all project members with feedback to their work. It ensures that use cases and functionalities are implemented in a correct manner and shows if changes infected the behaviour of the application in a bad way.

2.2 Evaluation Mission

In general, our mission is to improve our design and code quality.
This contains to find as many bugs as possible, find quality risks and so forth.

2.3 Test Motivators

Tests reduce bugs in new features and in existing features. Also tests are good documentation and reduce the cost of work if something needs to be changed.

3. Target Test Items

The listing below identifies those test items—software, hardware, and supporting product elements —that have been identified as targets for testing. This list represents what items will be tested.

- Desktop application “Nappy, the ingenious”
- Database connection

4. Outline of Planned Tests

4.1 Outline of Test Inclusions

- Unit tests for game logic and GUI
- Installation tests
- Functional tests

4.2 Outline of Other Candidates for Potential Inclusion

Database and Wiki stress-test. It is unlikely that we add more Characters.

4.3 Outline of Test Exclusions

Database (H2) itself
JavaFX library
Java System API (open Browser, Mail client, ...)

5. Test Approach

5.1 Initial Test-Idea Catalogs and Other Reference Sources

n/a

5.2 Testing Techniques and Types

5.2.1 Function Testing

Customers can define SikuliX tests in plain text with screenshots and developers implement these in Java.

Technique Objective:	Correct implementation of use-cases.
Technique:	Execute each use-case scenario's individual use-case flows or functions and features, using valid and invalid data, to verify that: <ul style="list-style-type: none">the expected results occur when valid data is used
Oracles:	n/a
Required Tools:	SikuliX
Success Criteria:	SikuliX tests don't fail.
Special Considerations:	Game mode 1 and 2 can't be tested easily. We created an extra version with no randomness (question selection) to test these two game modes.

5.2.2 JUnit Testing

JUnit is a popular and simple framework to write repeatable tests for Java.

Technique Objective:	Proving that all function work on a logical correct base.
Technique:	Writing for the functions tests to save the functionality of the application.
Oracles:	The tests run via Travis-Ci
Required Tools:	JUnit
Success Criteria:	Unit tests don't fail.
Special Considerations:	Not every functionality can be covered with Unit tests.

5.2.3 User Interface Testing

User Interface (UI) testing verifies a user's interaction with the software. The goal of UI testing is to ensure that the UI provides the user with the appropriate access and navigation through the functions of the target-of-test.

Technique Objective:	Navigation through the target-of-test reflecting business functions and requirements, including window-to-window, field-to-field, and use of access methods (tab keys, mouse movements, accelerator keys). Window objects and characteristics can be exercised—such as menus, size, position, state, and focus.
Technique:	Create or modify tests for each window to verify proper navigation and object states for each application window and object.
Oracles:	Maven Test or build fails.
Required Tools:	Java Maven TestFX (loaded by maven automatically) JUnit (loaded by maven automatically)
Success Criteria:	Every Unit tests passes without assert fails or unexpected exceptions.
Special Considerations:	n/a

5.2.4 Installation Testing

Technique Objective:	new installation: a new machine, never installed previously with Nappy, the ingenious update: a machine previously installed Nappy, the ingenious, older version
Technique:	Install desktop application on a linux, mac or windows.
Oracles:	n/a
Required Tools:	Java 8 (JRE or JDK)
Success Criteria:	Installation Testing Document should be marked successfully
Special Considerations:	n/a

We told 3 developers from other groups with different operation systems to download our application and complete our “installation test document”.

Installation Test from Tom Wolske:

https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Tom.pdf

Installation Test from Samuel Phillip:

https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Samuel.pdf

Installation Test from Jan-Eric Gaidusch:

https://github.com/nappydevelopment/docs/blob/master/pdfs/Installation%20Testing%20Document_Jani.pdf

6. Entry and Exit Criteria

6.1 Test Plan

6.1.1 Test Plan Entry Criteria

The entry of the Test Plan is to take immediately after the first version.

6.1.2 Test Plan Exit Criteria

The Test Plan is completed once all tests have been successfully completed.

6.1.3 Suspension and Resumption Criteria

The Test Plan cannot be canceled.

6.2 Test Cycles

n/a

7. Deliverables

7.1 Test Evaluation Summaries

Our summary of our tests are always up-to-date in Travis:

<https://travis-ci.org/nappydevelopment/Nappy-the-ingenuous>

The summary will produce with every push on GitHub. If the build fails, it will automatically create a Jira ticket.

7.2 Reporting on Test Coverage

Test Coverage is reported in:

- SonarQube: <http://193.196.7.25/overview?id=5235>
- Coveralls: <https://coveralls.io/github/nappydevelopment/Nappy-the-ingenuous?branch=master>
- Codacy: <https://www.codacy.com/app/NappyDevelopment/Nappy-the-ingenuous/dashboard>

7.3 Perceived Quality Reports

Quality is graded and reported in:

- Codacy: <https://www.codacy.com/app/NappyDevelopment/Nappy-the-ingenuous/dashboard>
- also in SonarQube!

More information about quality and metrics see chapter 14.

7.4 Incident Logs and Change Requests

n/a

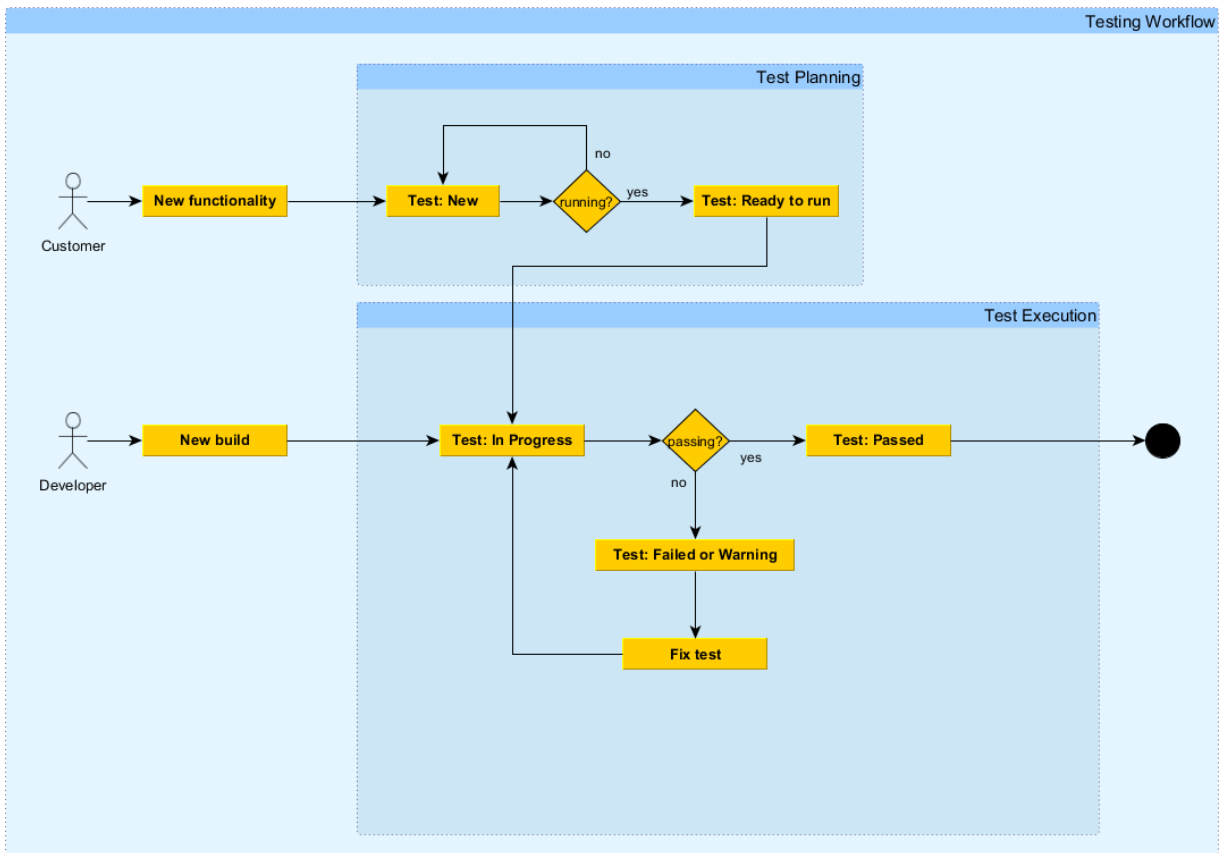
7.5 Smoke Test Suite and Supporting Test Scripts

n/a

7.6 Additional Work Products

n/a

8. Testing Workflow



9. Environmental Needs

9.1 Base System Hardware

The following table sets forth the system resources for the test effort presented in this *Test Plan*.

System Resources		
Resource	Quantity	Name and Type
Client Test PCs	1	Mac OS X 10.11.3 with Java 8
	3	Windows 7 with Java 8
	2	Linux with Java 8
	3	Windows 10 with Java 8

9.2 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	Version	Type and Other Notes
Windows	7, 8, 8.1, 10	Operating System
Linux	Ubuntu 14.04, 15.10	Operating System
Mac OS	10.11.3	Operating System
Firefox	45.x.x, 46.x.x	Internet Browser
Google Chrome	49.x.x	Internet Browser
MS Outlook	2010, 365	eMail Client software
Thunderbird	38.x.x	eMail Client software
Java	1.8.x	Programming language

9.3 Productivity and Support Tools

Tool Category or Type	Tool Brand Name	Version
Metrics	MetricsReloaded	1.7
Metrics and Coverage	SonarQube	4.5.7
Metrics and Coverage	Codacy	n/a
Coverage	Coveralls	n/a
Testing	JUnit	4.12
Testing	TestFx	4.0.4
Testing	SikuliX	1.1.0
IDE	IntelliJ IDEA	2016.1.1
IDE	Eclipse	Mars
Project management	JIRA	7.0.0

9.4 Test Environment Configurations

n/a

10. Responsibilities, Staffing, and Training Needs

10.1 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	Mehmet Ali Incekara Marvin Zerulla	Provides management oversight. Responsibilities include: <ul style="list-style-type: none">• planning• agree mission• identify motivators• present management reporting
Test Analyst	Mehmet Ali Incekara Marvin Zerulla	Identifies and defines the specific tests to be conducted. Responsibilities include: <ul style="list-style-type: none">• define test details• evaluate effectiveness of test effort• document change requests• evaluate product quality
Test Designer	Marvin Zerulla	Defines the technical approach to the implementation of the test effort. Responsibilities include: <ul style="list-style-type: none">• define test approach• define test automation architecture• verify test techniques• define testability elements• structure test implementation
Tester	Marvin Zerulla Mehmet Ali Incekara Marc Mahler Manuel Bothner	Implements and executes the tests. Responsibilities include: <ul style="list-style-type: none">• implement tests• execute test suites• analyze and recover from test failures
Implementer	Marvin Zerulla Mehmet Ali Incekara Marc Mahler Manuel Bothner	Implements and unit tests the test classes and test packages. Responsibilities include: <ul style="list-style-type: none">• creates the test components required to support testability requirements as defined by the designer

10.2 Staffing and Training Needs

n/a

11. Iteration Milestones

Milestone Planned	Start Date	End Date
20% coverage	Project start	22.05.2016
50%+ coverage	23.05.2016	12.06.2016

12. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Implementation specific tests	<Tester> should implement one test for each functionality. They should be generic.	Restructure tests

13. Management Process and Procedures

13.1 Measuring and Assessing the Extent of Testing

“

The *Coverage* view shows all analyzed Java elements within the common Java hierarchy. Individual columns contain the following numbers for the active session, always summarizing the child elements of the respective Java element:

- Coverage ratio
- Items covered
- Items not covered
- Total items

“

Source: <http://eclemma.org/userdoc/coverageview.html>

13.2 Assessing the Deliverables of this Test Plan

n/a

13.3 Problem Reporting, Escalation, and Issue Resolution

n/a

13.4 Managing Test Cycles

n/a

13.5 Traceability Strategies

n/a

13.6 Approval and Signoff

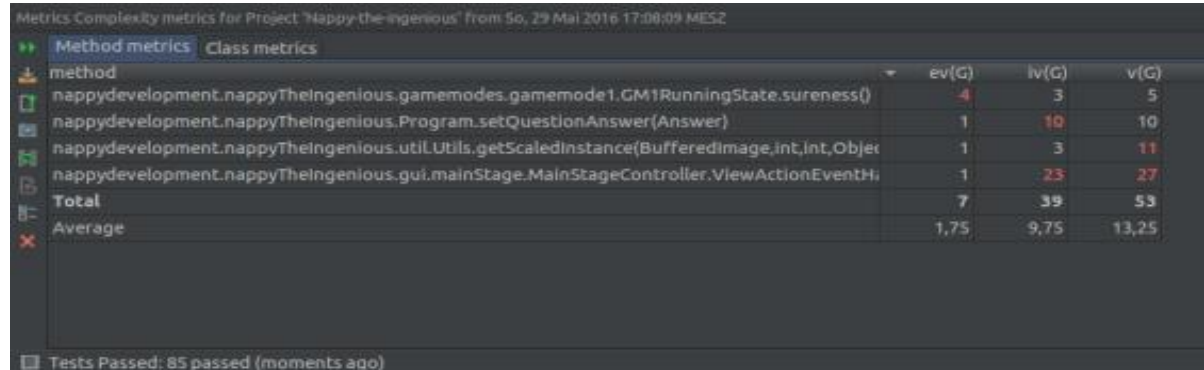
n/a

14. Metrics

We are using the metrics plug-in for IntelliJ: MetricsReloaded.

We are using following metrics profiles: Complexity and Dependency

metrics before:



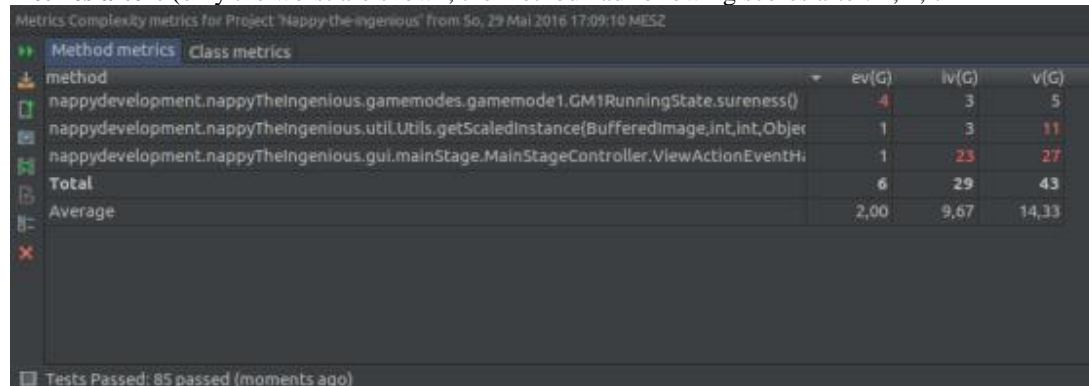
Method metrics	Class metrics
method	class
nappydevelopment.nappyTheingenious.gamemodes.gamemode1.GM1RunningState.sureness()	nappydevelopment.nappyTheingenious.gamemodes.gamemode1.GM1RunningState
nappydevelopment.nappyTheingenious.Program.setQuestionAnswer(Answer)	nappydevelopment.nappyTheingenious.Program
nappydevelopment.nappyTheingenious.util.Utils.getScaledInstance(BufferedImage,int,int,Object)	nappydevelopment.nappyTheingenious.util.Utils
nappydevelopment.nappyTheingenious.gui.mainStage.MainStageController.ViewActionEventHi	nappydevelopment.nappyTheingenious.gui.mainStage.MainStageController
Total	
Average	

ev(G) Calculates the essential complexity of each non-abstract method. Essential complexity is a graph-theoretic measure of just how ill-structured a method's control flow is.

iv(G) Calculates the design complexity of a method. The design complexity is related to how interlinked a method's control flow is with calls to other methods.

v(G) Calculates the cyclomatic complexity of each non-abstract method. Cyclomatic complexity is a measure of the number of distinct execution paths through each method. In practice, this is 1 + the number of if's etc. in the method.

metrics after: (only the worst are shown, the method had following scores after: 2, 4, 7)



Method metrics	Class metrics
method	class
nappydevelopment.nappyTheingenious.gamemodes.gamemode1.GM1RunningState.sureness()	nappydevelopment.nappyTheingenious.gamemodes.gamemode1.GM1RunningState
nappydevelopment.nappyTheingenious.util.Utils.getScaledInstance(BufferedImage,int,int,Object)	nappydevelopment.nappyTheingenious.util.Utils
nappydevelopment.nappyTheingenious.gui.mainStage.MainStageController.ViewActionEventHi	nappydevelopment.nappyTheingenious.gui.mainStage.MainStageController
Total	
Average	

The method we tried to fix was `Program.setQuestionAnswer` ([code before](#)) ([code after](#)). This was rather simple as this method has grown lately, it really only needed one try-catch. Also the counting of don't knows was moved into `gamemode1`.

Another spot in our code where metrics suggest that should change is our method "ViewAction-EventHandler" in our [MainStageController](#) but we have decided not to. There are so many if's because if someone starts the game the MainStage will "changed" to the GameState. We don't have an extra stage for both game modes because they are part of the MainStage and the transition is better. We can't and won't change this method because we don't have enough time.

The tool isn't part of our deployment process because it wasn't possible. We have Codacy, Coveralls and Sonarqube in our deployment process to check our code quality. But we thought we should present "MetricsReloaded" because it is a really powerful tool. Our deployment process is explained in our SAD.

The results of the metrics are exported as CSV:

<https://github.com/nappydevelopment/docs/tree/master/metrics>