**Software Test Plan (STP) Template**

Items that are intended to stay in as part of your document are in **bold**; explanatory comments are in *italic* text. Plain text is used where you might insert wording about your project.

This document is an annotated outline for a Software Test Plan, adapted from the IEEE Standard for Software Test Documentation (Std 829-1998).

Tailor as appropriate. Where you decide to omit a section, you might keep the header, but insert a comment saying why you omit the element.

**The Three Musketeers**

****

**FreeCol - the Colonization of America**

**Software Quality Assurance Plan**

**Version: 1.07 Date: 05/16/2016**

**Document History and Distribution**

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Revision Date | **Description of Change** | **Author** |
| 1.00 | 4/25/2016 | Created document | Obed |
| 1.01 | 4/28/2016 | Added introduction and testing approach | Kevat |
| 1.02 | 4/30/2016 | Added pass/fail criteria and environment requirements | Nahom |
| 1.03 | 4/30/2016 | Added change management approach | Obed |
| 1.04 | 4/30/2016 | Added information about EuropeWas.java, TypeCountMap.java |  |
| 1.05 | 4/30/2016 | Added information about Nation.java and Modifier.java | Nahom |
| 1.06 | 4/30/2016 | Added additional information about ResourceType.java, FreeColSpec.Java | Kevat |
| 1.07 | 5/13/2016 | Added information about LanguageOption.java and TextOption.java | Nahom |
| 1.08 | 5/16/2016 | Final revision | Obed |

**Table of Contents**

[1. Introduction 1](#_Toc451200710)

[￼￼ 2](#_Toc451200711)

[￼ Strategy 2](#_Toc451200712)

[￼Definitions and Acronyms 4](#_Toc451200713)

[2. Test Items 4](#_Toc451200714)

[2.1 Program Modules 5](#_Toc451200715)

[3. Features To Be Tested 5](#_Toc451200716)

[4. Features Not To Be Tested 6](#_Toc451200717)

[5. Approach 7](#_Toc451200718)

[5.1 Component Testing 7](#_Toc451200719)

[6. Pass / Fail Criteria 8](#_Toc451200720)

[7. Testing Process 9](#_Toc451200721)

[8. Environmental Requirements 10](#_Toc451200722)

[8.1 Hardware 10](#_Toc451200723)

[8.2 Software 10](#_Toc451200724)

[8.3 Security 11](#_Toc451200725)

[8.4 Tools 11](#_Toc451200726)

[8.5 Risks and Assumptions 12](#_Toc451200727)

[9. Change Management Procedures 12](#_Toc451200728)

[9.1 Issues Encountered 13](#_Toc451200729)

[9.2 Change Suggestions 13](#_Toc451200730)

[9.3 Authorized Changes 14](#_Toc451200731)

# Introduction

The game FreeCol is an open source turn-based game similar to Civilization. The source code for this project is written in Java and is available online. This document is a Software Test Plan (STP) intended for FreeCol source. FreeCol is split into various interdependent packages. The classes in these packages are grouped based on their core functionality. The most important packages include *net.sf.freecol.common.model* and *net.sf.freecol.common.io*. These packages are important since they provide many of the base classes used extensively throughout the code. As a part of this testing cycle, the team will define a testing plan (tracked and maintained via this STP) and implement testing within the FreeCol project based on that plan.

This STP will be an evolving document as the team continues to analyze the codebase and add testing artifacts. The test strategy consists of 3 major phases: analysis phase, implementation phase and retrospective phase. In the analysis phase, the team analyzed the code to determine which classes to test first.

The following factors were considered when ranking classes:

1. Small classes were preferred over larger ones (based on number of lines of code)
2. More referenced classes preferred over less referenced
3. Classes with less dependencies preferred over classes with more dependencies

The Eclipse plug-in CodeCity was used to get a graphical image of all the classes in the project based on lines of code and logical lines of code. An example of such an image is given below:



### Figure 1 Colony Image

CodeCity was utilized during the analysis phase to determine class size. The Eclipse plug-in ObjectAid was used to then map the dependencies of the smaller classes. The map of dependencies for each of the classes in the STP can be found in the /reqs/DependencyMaps in the project. Finally, references for each class were determined by using in build Eclipse functionality. After thorough consideration of these factors, it was decided that the following classes should be tested as a part of this STP:

Testing Deliverables:

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Package | LOC | Developer |
| EuropeWas.java | net.sf.freecol.common.model | 85 | Kevat |
| FreeColSpecObjectType.java | net.sf.freecol.common.model | 277 | Kevat |
| FreeColSpecObject.java | net.sf.freecol.common.model | 89 | Kevat |
| ResourceType.java | net.sf.freecol.common.model | 131 | Kevat |
| Nation.java | net.sf.freecol.common.model | 352 | Nahom |
| AbstractGood.java | net.sf.freecol.common.model | 306 | Nahom |
| Modifier.java | net.sf.freecol.common.model | 689 | Nahom |
| LanguageOption.java | net.sf.freecol.common.option | 329 | Nahom |
| TextOption.java | net.sf.freecol.common.option | 150 | Nahom |
| Turn.java | net.sf.freecol.common.model | 292 | Obed |

### Table 1 Testing Deliverables

**The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan**.

## ￼￼

Due to time constraints and the size of the codebase, it is not possible to test every class within the project. Additionally, the initial project had minimal testing; this software test plan was the first attempt and introducing testing to the project. Due to these factors, this STP focuses on unit testing of frequently used classes rather than integration, acceptance or non-functional testing.

As most of the FreeCol game is already developed, most of the new development would be maintenance development, such as correcting existing bugs or perfecting the code. Many of the classes in the packages model and io are frequently referenced throughout the code. Making additional changes to these classes could be problematic; it would be very time consuming to test each reference or dependency upon these base classes. Therefore, the main objective for this STP is introducing unit testing for small but frequently used classes. Unit testing for such classes would enable the developers to catch any changes they might introduce when changing these classes. Implementing these unit tests would decrease the time required for new development as well as the risk of introducing new bugs to existing functionality.

## ￼ Strategy

**Testing is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item.**

The main objective of this testing cycle will be to implement unit tests for the classes listed as Testing Deliverables in the Introduction section of this document. Developers will attempt to achieve at least 75% code coverage via unit testing for each of these classes. This cycle will consist of 25 days (April 25th - May 16th) with 3 developers. The cycle will be split into 3 major phases as shown below:

**Analysis phase:** April 25th - May 2nd

**Implementation Phase:** May 2nd - May 13th

**Retrospective Phase:** May 13th - May 16th

The following table outlines the activities for each developer and the due date by which each activity should be completed:

|  |  |  |
| --- | --- | --- |
| Activity | Developer | Due date |
| Analysis Phase |  |  |
| Create STP | Obed | 4/25/2016 |
| Add STP introduction and testing approach | Kevat | 4/30/2016 |
| Add STP Change management procedures | Obed | 4/30/2016 |
| Add STP Pass/Fail criteria and environment requirements | Nahom | 4/30/2016 |
| Analyze project and decide on list of classes | All | 5/2/2016 |
| Implementation Phase |  |  |
| Add STP sections 2, 4, and 7 based on list of classes | All | 5/2/2016 |
| Create repository for project | Obed/Nahom | 5/3/2016 |
| Add comment, unit testing to assigned classes | All | 5/13/2016 |
| Retrospective Phase |  |  |
| Update STP based on progress | All | 5/16/2016 |
| Create test suite to all the JUnit test cases. | Obed | 5/16/2016 |
| Add any additional comments to created JUnit test cases using the standards for writing Doc comments for Javadoc tool. | Obed | 5/16/2016 |

**1.3 Scope**

The STP will be updated as needed. There is a list of required updates as a part of the planned activities. These updates are listed in the activity table above. Apart from the scheduled updates, the STP can be updated by any developer as needed. Any time an unplanned update is made to the document, the developer making the update shall inform the other developers via email. All updates to the document (planned and unplanned) will require a change in version of the document which is tracked in the Revision History section of this STP.

**Testing will be performed at several points in the life cycle as the product is constructed. Testing is a very 'dependent' activity. As a result, test planning is a continuing activity performed throughout the system development life cycle. Test plans must be developed for each level of product testing.**

## ￼Definitions and Acronyms

**STP** – Software Testing Plan

**LOC** – Lines of code

# Test Items

# 

This section describes the test items in the plan. It shows the specifications that need to be followed while creating tests and for documentation.

* *Requirements specification,*
* Create unit tests for each method until Code Coverage reaches 75%
* Follow Javadoc documentation standard guidelines and practices for the tests created
* Create coverage report
* *Design specification,*
* Create before and after functions as well as variables for the test class
* Identify dependencies upon other classes for each method
* Create stubs for dependencies
* *Users guide,*
* Create dependency map
* *Operations guide,*
* Create a test class according to the naming convention
* Add methods to the exclusion list if analysis or creation of stubs takes too long
* *Installation guide,*
* Updated version of Unix, Linux and Windows
* 512 MB of memory recommended
* Screen Resolution 1024 x 768
* *Features (availability, response time),*
* Test cases must go through with no error.
* *Defect removal procedures, and*
* *Verification and validation plans.*
* Verification and validation will be done as peer review of each other’s work.

## 2.1 Program Modules

Not applicable to current STP

2.2 User Procedures

At the end of all testing activities, the developers will review each other’s tests and make sure that documentation standards where followed. A tester will volunteer to be the quality assurance reviewer. Obed volunteered for this role. During the review phase of each other’s tests the developer will create a list of fixes if they found any and send an email to Obed, so he can make those changes. Obed will take the suggested corrections and update the tests accordantly. Additionally, Obed will follow the style guide convections for documenting comments for Java programs described in the article “How to Write Doc Comments for the Javadoc Tool” on the Oracle website.

# Features To Be Tested

# 

Developers are expected to test each method within the agreed upon class. The only exceptions to this rule are the methods listed under the section "Features Not To Be Tested" below. Some classes will have tests that will be created by one developer, and the test will be commented by another developer. Each developer will periodically check another’s developers work and suggest changes if applicable following the change management procedures.

# 4. Features Not To Be Tested

This section lists all the methods which will not be tested as a part of this STP and the reason why they will not be tested.

|  |  |  |
| --- | --- | --- |
| Class Name | Method Name | Reason for exclusion |
| EuropeWas | getNewUnit | Requires setting up additional parameters. Given the lack of documentation, this task requires more time than allotted to this STP. |
| FreeColSpecObjectType.java | readChild | Requires setting up additional parameters. Given the lack of documentation, this task requires more time than allotted to this STP. |
| FreeColSpecObject.java | newInstance | Requires setting up additional parameters. Given the lack of documentation, this task requires more time than allotted to this STP. |
| Nation.java | getRandomNonPlayerNationNameKey | Requires setting up additional parameters; requires random pseudo source. |
| Modifier.java | hashcode | Requires setting up additional parameters in order to calculate the parameter |

### Table 2 Features Not to Be Tested

### 

# 5. Approach

This section will discuss its corresponding subsection if it applies to this STP. Any subsection that does not apply to this STP will be noted with a message “Not applicable to current STP.”

## 5.1 Component Testing

The major requirement for unit testing of the classes listed is to achieve 75% code coverage for the class. This includes the constructor methods in the class as well as other methods. The tests for each class should reside in the same package as the class and should follow the naming convention <ClassName>Test.java. The tests should all be written in JUnit. There are no plans to create a test suite containing all the unit tests. Developers are expected to run each unit test separately.

**NOTE:** Many of the classes are dependent upon the FreeColXMLWriter and FreeColXMLReader classes. However, no enough documentation is provided regarding these classes. They expect files written in certain formats or object with certain attributes. Due to the lack of documentation as well as an attempt to independently test agreed upon classes without dependencies on other classes, we have created stubs for any calls to FreeColXMLWriter and FreeColXMLReader. Any references to these classes should instead use the classes StubXMLReader and StubXMLWriter. These stub classes were recently created and can be found in the *net.sf.freecol.common.io* package. Developers should validate the variable outputString found in each of these classes. The stubbed methods within these classes log the input given to a method every time the method is called. This input is added to the outputString variable. This allows the developer to validate the expected input values for these methods with the actual input values.

**5.2 Integration Testing**

Not Applicable to current STP

**5.****3 Interface Testing** ￼

Not Applicable to current STP

**5.4 Security Testing**

Not Applicable to current STP

**5.5 Performance Testing**

Not Applicable to current STP

**5.6 Regression Testing**

Not Applicable to current STP

**5.7 Acceptance Testing**

Not Applicable to current STP

**5.8 Beta Testing**

Not Applicable to current STP

# 6. Pass / Fail Criteria

This section will cover pass fail criteria in its respective subsection. If any of the subsections do not apply to the STP it will have a message “Not applicable to current STP.”

**6.1 Suspension Criteria**

* + Multiple methods require additional set up of variables
  + Lack of documentation specifying the purpose and functionality of methods
  + Build application repeatedly crashes.
  + The application gives output that is not identical to the expected output.
  + knowledge of initial expected value will not equal the actual value.

**6.2 Resumption Criteria**

* + Testing will continue when the expected value is equal to actual value
  + Testing can be resumed when additional variables/components required can be stubbed
    - In this case, testing variables which are instances of the stubbed class must be created

**6.3 Approval Criteria**

The approval criteria for the tests is bulleted bellow. The table show the actual code coverage reached compared to the agreed code coverage criteria. The classes that have lower coverage criteria was not accomplished due to reason mentioned on “Table 2 Features Not to Be Tested.”

* + 75% Code Coverage criteria
    - Does not include excluded methods

|  |  |
| --- | --- |
| **CLASS NAME** | **CODE COVERAGE %** |
| EuropeWas.java | 16% |
| FreeColSpecObject.java | 57% |
| ResourceType.java | 78.8% |
| Nation.java | 79.7% |
| AbstractGood.java | 79.5% |
| Modifier.java | 80.0% |
| LanguageOption.java | 81.8% |
| TextOption.java | 94.8% |
| Turn.java | 95.8% |

### Table 3 Code Coverage Final Results

# 7. Testing Process

This section covers the testing process to be follow by each developer during the implementation phase and during the retrospective phase. Each subsection has a bullet list for the required information. If a subsection does not apply to the STP is will have the message “Not applicable to current STP.”

**7.1 Test Deliverables**

* + Analysis of dependencies for each class agreed upon, saved as image files in the /reqs/DependencyMaps folder.
  + Unit tests for each class, stored in a class with the following naming convention: <ClassName>Test.java
  + Coverage report for each class stored in the /reqs/ folder

**7.2 Testing Tasks**

* + Create dependency map
  + Create a test class according to the naming convention
  + Create before and after functions as well as variables for the test class
  + Identify dependencies upon other classes for each method
  + Create stubs for dependencies
  + Add methods to the exclusion list if analysis or creation of stubs takes too long
  + Create unit tests for each method until Code Coverage reaches 75%
  + Follow Javadoc documentation standard guidelines and practices for the tests created
  + Create coverage report

**7.3 Responsibilities**

Each developer is responsible for all of the testing tasks listed above for each class they are assigned. Some testing tasks will be done by the two or all the developers on the same class. This was done on purpose to take advantage of each developers’ weaknesses and strengths for developing tests.

**7.4 Resources**

Not Applicable to current STP

**7.5 Schedule**

The following schedule gives an approximate number of hours the developers should spend on each task

|  |  |
| --- | --- |
| Task | Hours |
| Create Dependency Map | < 1 hour |
| Create Test Class | < 1 hour |
| Create before and after functions, testing variables | 1 hour |
| Identify dependencies for each method | 3 hours |
| Create stubs | 1 hour |
| Add methods to exclusion list | < 1 hour |
| Create unit test | 1 hour / 20 Lines of Code (LOC) |
| Create Coverage report | < 1 hour |
| Create test suite | < 1 hour |

### Table 3 Task Schedule

# 8. Environmental Requirements

This section will discuss the environmental requirements in their respective subsection. If the subsection does not apply it will have a message “Not applicable to current STP.”

## 8.1 Hardware

* + The following requirements are Free Col computer hardware needed to compile and run
    - Updated version of Unix, Linux, and Windows
    - FreeCol is very large software (256MB) memory so for smoother interaction 512MB of memory recommended
    - Screen resolution: 1024 x768 pixels

## 8.2 Software

* + Written in Java Programming Language
  + Ant build system to run the tests
  + Compatible with Sun Java8
  + IDE: Eclipse (for developer)
  + To test software, it requires jUnit 4.12jar and hamcrest-core 1.3 jar
  + Compiler compliance level 1.8

## 8.3 Security

Not applicable to current STP.

## 8.4 Tools

The following table describes tools and software used during our testing phase. The tools listed below are in no particular order. Most tools mentioned here are tools that have been used during the semester and its assumed that acquisition, training, and support is already know by each developer. Any additional tool that was not used the entire semester will be learned, practiced, and used by individual bases of each developer.

|  |  |
| --- | --- |
| **TOOLS** | **DESCRIPTION** |
| Find Bugs | One of the tools we used for our project is called Find Bugs. It is a code analyzer that detects for potential error. Error have to be fixed before submitting. It classifies them from scariest, scary, troubling, and concern. Applied on: Modifier.java |
| Javadocs | The Oracle website describes Javadocs as “The **Javadoc**™ tool parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages describing (by default) the public and protected classes, nested classes (but not anonymous inner classes), interfaces, constructors, methods, and fields.” |
| JUnit | JUnit is an eclipse tool used to create unit test cases. |
| Googel CodePro | Google CodePro tools helps developers run different features like code metrics, audit code, generate test cases, generate factory classes, and find dead code to name a few. |
| CodeCity | CodeCity is an eclipse plug-in used to get a graphical image of all classes in the project based on lines of code and logical lines of code. |

### Table 4 Tools Used

## 8.5 Risks and Assumptions

FreeCol is a very large application. To maintain it, significant amount of time for planning and executing is required. Understanding that, one of the constraints that we faced as a group when testing this application was the value of time. There are numerous factors that contribute to this issue:

1. software being too large (many methods and classes)
2. Poor test planning
3. Time span given to do the project (low)

Nevertheless, since it was impossible for us to test every aspect of this application, our approach was selecting as many classes as we can have based on the size of the class or the one we thought were important. We selected classes that needed to be tested and improved. We used various tools such as Google Code Pro to generate test cases for these classes.

# 9. Change Management Procedures

The change management process for this STP is to communicate with developers via email or phone when issues arise during the implementation phase. If an issue arises during implementation the developer should email the other developers about said issue. If the issue is something that can be resolved by the developer that encountered the issue no initiation or change review process will be initiated. If the issue is something that cannot be resolved by the developer one approach to fixing the issue is to reach out to Dr. Dehlinger for assistance on the issue. In the event that the issue has no solution a change initiation process will begin. If issues are encountered during the implementation phase they will be discussed on a subsection named “Issues Encountered.”

The change initiation process will begin after all the options mentioned above have been exhausted. An email will be sent out by the developer to communicate the issue and suggest changes to the testing approach. When the change initiation process is put into motion it will result for a change review to occur. The change review will consist of analyzing the change initiation to make decisions on the changes. Some of change initiation processes will be discussed in its subsection.

The change authorization process will begin after the change initiation and change review have been completed. The change authorization will be decided by all three developers. The change authorization process will be done via conference call. During the conference call developers will agree upon how to processed as a result of whatever the issue might be. A subsection of authorized changes will also be shared in its subsection.

## 9.1 Issues Encountered

Issues that the team encountered before and during the course of this final project will be discussed here. The first issue that the team encountered was when one of the team members created the GitHub repository and the other members tried to import the project into eclipse. Two of the team members could not import the project into get hub. There was no change suggestion, change suggestions, or authorized changes process for this particular problem. The proposed solution for this issue was to reach out to Dr. Dehlinger to help the team resolve the issue.

Another issue that was found was for an individual team member that had problems getting the project to build correctly. This issue was also solved when seeking help from Dr. Dehlinger for the import issue. The solution for this particular problem was an eclipse version problem. No change suggestion or authorization process was required for this problem.

One major problem the team encountered was on a commit that created issues with the repository. The developer followed the change management procedure when trying to solve the problem. The individual emailed the team members and regularly updated as he tried to resolve the issue. The developer that encountered this problem then, reached out to Dr. Dehlinger for assistance with the commit problem. As a result of this problem the entire change management procedure was implemented. The developer followed all the steps, first by emailing the team mates and reaching out to the professor. The change suggestion and authorized change for this particular issue will be mentioned on their respective subsections. No other major issues happened that required for a complete change management procedure to be put into action.

## 9.2 Change Suggestions

This subsection will discuss any change suggestions that happened as a result from any issue mentioned on the issues encountered subsection. The first change suggestion was implemented as a result of the import issue mentioned above. This suggestion had no alternative and it had to be implemented. The change suggestion was done with the help of Dr. Dehlinger to resolve the import problem the team encountered. The solution was for one of the developers to fork the project from a working repository that the professor provided for the team. The authorization procedure for this problem was a given because of the nature of the problem.

A second change suggestion happened as a result of the major commit problem into the repository. All the solution options where explored, but no real solution was found. The change suggestion was then communicated via email and further discussed via a conference call with all team members, because of the nature of the change suggestion. The suggestion was to start from scratch meaning that any previous work done to the project would have to be redone. Unfortunately, this change suggestion had no other alternative an it had to be authorized.

One last change suggestion was initiated during the retrospective phase. This change suggestion was to create a test suite that was not planned to be included in this project. The change suggestion was brought forth because the way the HTML coverage report for the JUnit test rendered the report. Each individual report was affected when another report was exported and it took away the coverage from the previous report. To fix this issue the team decided to create a test suite and then run the coverage report from the suite that way all the Junit test cases and classes that had their test where under one coverage report. This change suggestion process was authorized via email communication between all team members.

## 9.3 Authorized Changes

This subsection will discuss authorized changes as a result of the change review and change initiation. As mentioned in previous subsection most of the authorized changes had to happened because there was no other alternative. The authorized changes followed all the change management procedures. Communication via email was sent when issues were discovered, solutions were suggested and reviewed by all team members, and all the team members agreed on the proposed solutions.