**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.

**Serious Software Development Group**

****

**FreeCol Enhancement Project**

**Software Quality Assurance Plan**

**Version: (1.0) Date: (05/16/2018)**

**Document History and Distribution**

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision #** | **Revision Date** | **Description of Change** | **Author** |
| 1 | 3/30/18 | First commit - no changes | Ethan |
| 2 | 4/18/18 | Updated client package | Kyle |
| 3/4 | 4/20/18 | Refactored long methods in net.sf.freecol.common.io | Ethan |
| 5 | 4/27/18 | .common files refactored: long methods, type checking, other bad smells | Ethan |
| 6 | 4/27/18 | .common files refactored: CodePro audit | Ethan |
| 7 | 4/27/18 | new JUnit test run: build passed, all tests passed | Ethan |
| 8 | 4/28/18 | testing AutoRefactoring tool | Ethan |
| 9 | 4/29/18 | created .gitignore: added FreeCol.jar and test/ directory | Ethan |
| 10 | 4/30/18 | deleted deprecated/empty FreeCol-Master directory | Sam |
| 11 | 5/14/18 | removed dead code using AutoRefactor tool | Ethan |
| 12 | 5/15/18 | Spartanizer tool run on .common packages | Ethan |
| 13 | 5/16/18 | Additional client tests done | Kyle |
| 14 | 5/16/18 | Add files via upload | Ryan |

**TABLE OF CONTENTS**

**1.** **Introduction 1**

**2.** **Test Items 2**

**3. Features To Be Tested 3**

**4. Features Not To Be Tested 3**

**5. Approach 3**

**6. Pass / Fail Criteria 5**

7. **Testing Process 5**

**8. Environmental Requirements 6**

**9. Change Management Procedures 7**

# INTRODUCTION

(NOTE 1: THE SOFTWARE TEST PLAN GUIDELINES WERE DERIVED AND DEVELOPED FROM IEEE STANDARD FOR SOFTWARE TEST DOCUMENTATION (829-1998)).

*(Note 2: The ordering of Software Test Plan (STP) elements is not meant to imply that the sections or subsections must be developed or presented in that order. The order of presentation is intended for ease of use, not as a guide to preparing the various elements of the Software Test Plan. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content.)*

*The Introduction section of the Software Test Plan (STP) provides an overview of the project and the product test strategy, a list of testing deliverables, the plan for development and evolution of the STP, reference material, and agency definitions and acronyms used in the STP.*

**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**.

**1.1 Objectives**

*(Describe, at a high level, the scope, approach, resources, and schedule of the testing activities. Provide a concise summary of the test plan objectives, the products to be delivered, major work activities, major work products, major milestones, required resources, and master high-level schedules, budget, and effort requirements.)*

1. The scope of our project will be to improve the quality and code design of FreeCol, an open-source strategy-based game implemented in java.
2. We will be using a multitude of refactoring and testing tools provided by the Eclipse IDE to refactor and test the source code of FreeCol.
3. We are working in a group of four student developers. Each developer will be able to dedicate at least three hours of work time a week to project activities.
4. Each developer will utilize two refactoring plugins (that were not included in the course) to refactor and improve the source code.
5. We will submit our finished final project on the 16th of May, which gives us around one month to complete all tasks. A soft deadline for each task is as follows:
   1. By tuesday class on 4/24: complete all prior-possible content of this Quality Assurance template. Also, accomplish preliminary testing of source code.
   2. By tuesday class on 5/1: continue testing source code. Continue adding to QA template.
   3. By tuesday class on 5/9: continue source code testing and begin working on new. feature. Continue working on QA template.
   4. By tuesday class on 5/15: finish source code testing, new feature, and QA template.
   5. By 11:59 5/16: make pull request from repo, submit QA template.
6. The finished project needs to be compilable so the professor can grade it.

**1.2 Testing 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.** (*This may appear as a specific document (such as a Test Specification), or it may be part of the organization's standard test approach. For each level of testing, there should be a test plan and an appropriate set of deliverables. The test strategy should be clearly defined and the Software Test Plan acts as the high-level test plan. Specific testing activities will have their own test plan. Refer to section 5 of this document for a detailed list of specific test plans.)*

*Specific test plan components include:*

* *Purpose for this level of test,*
* *Items to be tested,*
* *Features to be tested,*
* *Features not to be tested,*
* *Management and technical approach,*
* *Pass / Fail criteria,*
* *Individual roles and responsibilities,*
* *Milestones,*
* *Schedules, and*
* *Risk assumptions and constraints.*

Each developer will go through each of their packets refactoring as necessary based on JDeoderant and Google Code Pro’s auditing. Once refactored, developers will create unit tests using JUnit. The newly created unit tests will be tested using Eclemma to make sure they have a sufficient code coverage. Lastly, developers will test their code for munition testing using PIT mutation testing software.

**1.3 Scope**

*(Specify the plans for producing both scheduled and unscheduled updates to the Software Test Plan (change management). Methods for distribution of updates shall be specified along with version control and configuration management requirements must be defined.)*

The scope of the work will be the refactoring and improvement of the FreeCol project and all of the tasks this involves. The developers have agreed to abandon intensive testing of multiplayer features but instead focus on the functionality of the game which involves the GUI, client, and server packages.

**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.**

**1.5 Definitions and Acronyms**

*(Specify definitions of all terms and agency acronyms required to properly interpret the Software Test Plan. Reference may be made to the Glossary of Terms on the IRMC web page.)*

Scope- mostly used in regards to the project scope, which is the agreed upon area of work expected from the project.

Refactor- improving the functional performance of a program by altering the source code. This should not, however, affect the outward behavior of the code.

IDE- Integrated Developement Environment. Provides an environment for programmers to generate software through coding.

Soft Deadline- a presumptive deadline established before the exact deadline is known.

Developer- refers (in this document) to the four student programmers working on this project.

QA- Quality Assurance. A term referring to (in the “coding” industry) the act of analyzing the behavior of a program in order to improve its function.

Repo- Repository. Electronic storage of files.

Unit Test- level of testing where specific units of code are tested.

# TEST ITEMS

*(Specify the test items included in the plan. Supply references to the following item documentation:*

* *Requirements specification,*
* *Design specification,*
* *Users guide,*
* *Operations guide,*
* *Installation guide,*
* *Features (availability, response time),*
* *Defect removal procedures, and*
* *Verification and validation plans.)*

**2.1 Program Modules**

*(Outline testing to be performed by the developer for each module being built.)*

Program modules will first be refactored using JDeodorant and Google CodePro’s auditing when necessary. Each (4) developer will then use two outside refactoring plugins to improve packages. The modules are broken down as follows:

1. .common packages
2. .client packages.
3. .server packages
4. .metaserver package
5. .tools package

**2.2 User Procedures**

*(Describe the testing to be performed on all user documentation to ensure that it is correct, complete, and comprehensive.)*

# 3. FEATURES TO BE TESTED

*(Identify all software features and combinations of software features to be tested. Identify the test design specifications associated with each feature and each combination of features.)*

.client Packages- will be tested and refactored using Google CodePro and Eclipse’s Bad Smells utilities. The two extra refactoring utilities used on these packages will be FindBugs and Unnesessary Code Detector.

.common Packages- will be tested and refactored using Google CodePro and Eclipse’s Bad Smells utilities. The two extra refactoring utilities used on these packages will be SpotBugs and AutoRefactor.

# 4. FEATURES NOT TO BE TESTED

*(Identify all features and specific combinations of features that will not be tested along with the reasons.)*

Features that will not be tested include code for multiplayer parts of the game and sections that rely to heavily on other packages. The multiplayer will not be tested simply because there is too much code and too little time to be tested and that is something that is not necessary for the game to work. The packages that rely on other packages have been left out of testing in order to avoid issues when committing to the repository.

The multiplayer features rely on the .server and .metaserver packages.

# 5. APPROACH

*(Describe the overall approaches to testing. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each task. Identify the types of testing to be performed along with the methods and criteria to be used in performing test activities. Describe the specific methods and procedures for each type of testing. Define the detailed criteria for evaluating the test results.)*

*(For each level of testing there should be a test plan and the appropriate set of deliverables. Identify the inputs required for each type of test. Specify the source of the input. Also, identify the outputs from each type of testing and specify the purpose and format for each test output. Specify the minimum degree of comprehensiveness desired. Identify the techniques that will be used to judge the comprehensiveness of the testing effort. Specify any additional completion criteria (e.g., error frequency). The techniques to be used to trace requirements should also be specified.)*

**5.1 Component Testing**

*(Testing conducted to verify the implementation of the design for one software element (e.g., unit, module) or a collection of software elements. Sometimes called unit testing. The purpose of component testing is to ensure that the program logic is complete and correct and ensuring that the component works as designed.)*

Unit and component testing will be done using Eclipse’s JUnit testing feature. JUnit tests have already been created for the program by the creators, and we will be using these to test the changes we make before every repository commit. JUnit tests will be run through the ANT build and manipulated by the build.xml file.

**5.2 Integration Testing**

*(Testing conducted in which software elements, hardware elements, or both are combined and tested until the entire system has been integrated. The purpose of integration testing is to ensure that design objectives are met and ensures that the software, as a complete entity, complies with operational requirements. Integration testing is also called System Testing.)*

Once all unit tests of a package are created, the entire test package will be tested as a whole to ensure those units are properly working together.

**5.3 Interface Testing**

*(Testing done to ensure that the application operates efficiently and effectively outside the application boundary with all interface systems.)*

The interface will be tested similar to the rest of the code. The interface will be refactored where necessary, and unit tests will be created. Unit tests will then be checked for code coverage and mutation testing.

**5.4 Security Testing**

*(Testing done to ensure that the application systems control and auditability features of the application are functional.)*

CodePro’s audit tool will be used to look for security risks. Any high severity violations will be fixed and removed.

**5.5 Performance Testing**

*(Testing done to ensure that that the application performs to customer expectations (response time, availability, portability, and scalability)).*

Isolation testing will be used when performance testing. When a system problems occur, the test execution will be repeated in hopes of isolating and fixing the fault in the code.

**5.6 Regression Testing**

*(Testing done to ensure that that applied changes to the application have not adversely affected previously tested functionality.)*

Whenever new tests are added to the project, previously tested units will be retested to make sure no changes were made to these units. If changes were made and cause new problems to previous units, the developer is not to move on until all problems are fixed.

**5.7 Acceptance Testing**

*(Testing conducted to determine whether or not a system satisfies the acceptance criteria and to enable the customer to determine whether or not to accept the system. Acceptance testing ensures that customer requirements' objectives are met and that all components are correctly included in a customer package.)*

The two major tests that we are worried about for acceptance testing is our code coverage and mutation testing. Before moving on to the next test, developers will ensure that both of these tests pass with a minimum coverage of 80%.

**5.8 Beta Testing**

*(Testing, done by the customer, using a pre-release version of the product to verify and validate that the system meets business functional requirements. The purpose of beta testing is to detect application faults, failures, and defects.)*

Beta testing will be done by the testing developers before and after testing the product. We will try to find and problems with the product before testing to see if there is an area we should focus on, and after ensure that we did not create any new problems.

# 6. PASS / FAIL CRITERIA

*(Specify the criteria to be used to determine whether each item has passed or failed testing.)*

**6.1 Suspension Criteria**

(*Specify the criteria used to suspend all or a portion of the testing activity on test items associated with the plan.)*

Tests items will be suspended if their tests create a new problem for another test item.

**6.2 Resumption Criteria**

*(Specify the conditions that need to be met to resume testing activities after suspension. Specify the test items that must be repeated when testing is resumed.)*

Once the newly created problem is fixed, the testing on the original test item may resume as planned until another problem is created leading to another suspension.

**6.3 Approval Criteria**

*(Specify the conditions that need to be met to approve test results. Define the formal testing approval process.)*

For this product, our main test focuses will be on code coverage and mutation coverage. For both of these tests, test approval will not be accepted until a minimum of 80% coverage is reached.

# 7. TESTING PROCESS

*(Identify the methods and criteria used in performing test activities. Define the specific methods and procedures for each type of test. Define the detailed criteria for evaluating test results.)*

**7.1 Test Deliverables**

*(Identify the deliverable documents from the test process. Test input and output data should be identified as deliverables. Testing report logs, test incident reports, test summary reports, and metrics' reports must be considered testing deliverables.)*

1. Test Reports
   1. All developers will submit coverage reports for their respective tests
   2. Any code errors or faults will be identified and added to a bug tracker and ranked based on actual or potential severity
   3. Developers will compile and submit Google CodePro reports identifying potential refactoring points and code smells.
2. Test Inputs
   1. The input domain and test case will be documented and loggeds

**7.2 Testing Tasks**

*(Identify the set of tasks necessary to prepare for and perform testing activities. Identify all intertask dependencies and any specific skills required.)*

1. Environment configuration using up-to-date Eclipse (Oxygen+) and JRE (SE 9+)
2. Identify testing priorities for the codebase
3. Coordinate individual vs. collective responsibilities
   1. Designate assigned packages and classes for individuals to prioritize work on

**7.3 Responsibilities**

*(Identify the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving test activities. These groups may include the developers, testers, operations staff, technical support staff, data administration staff, and the user staff.)*

Our group has decided to break the project up into three separate section; client, common, and metaserver/server tools. Kyle will be developing tests on the client section, Ethan will test the common section, Sam will test the metaserver/server tools, and Ryan will work on documentation. Once tests are completed, the group will work together to create the new feature and tests for the feature.

**7.4 Resources**

*(Identify the resources allocated for the performance of testing tasks. Identify the organizational elements or individuals responsible for performing testing activities. Assign specific responsibilities. Specify resources by category. If automated tools are to be used in testing, specify the source of the tools, availability, and the usage requirements.)*

This project is going to require several testing tools throughout the testing process. During the refactoring stage, developers will use JDeodorant to check for code smells, and Google CodePro’s audit tool to look for code violations. Unit tests will be created using JUnit with the option of using CodePro to auto create test cases. Lastly, to check mutation testing, developers will use PIT Mutation to make sure an acceptable amount of mutants are killed off.

The team members will also use two refactoring and code improvement plugins per developer. These plugins will include SpotBugs, AutoRefactor, FindBugs, Spartanizer, among others.

**7.5 Schedule**

*(Identify the high level schedule for each testing task. Establish specific milestones for initiating and completing each type of test activity, for the development of a comprehensive plan, for the receipt of each test input, and for the delivery of test output. Estimate the time required to do each test activity.)*

*(When planning and scheduling testing activities, it must be recognized that the testing process is iterative based on the testing task dependencies.)*

By April 24th, developers should have completed refactoring their sections of code using Jdeodorant and CodePro’s auditing tools. JUnit tests should be created and passing the required code coverage by May 3rd. By May 10th, developers should finish their mutation tests using Pit Mutation and begin working on the new feature. By May 15th, developers should have completed creating the new feature and supplied the required tests. Before the end of May 16th, a final pull request will be make from the repo.

# 8. ENVIRONMENTAL REQUIREMENTS

(Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.)

**8.1 Hardware**

*(Identify the computer hardware and network requirements needed to complete test activities.)*

All developers will need access to:

1. either:
   1. computer with at least
      1. 2Ghz processing speed per core
      2. 4GB memory
      3. 5GB free storage space

ii. regular access to a Towson University desktop computer

b. A reliable internet connection

**8.2 Software**

*(Identify the software requirements needed to complete testing activities.)*

All developers will need:

1. Eclipse version 4.7 or later
2. Git Bash for PC or Github Desktop

**8.3 Security**

*(Identify the testing environment security and asset protection requirements.)*

All developers will need:

1. Password protection for their personal computers (if they use one)
2. The team Github account needs to be invite-edit only and each developer will have edit access. The only developer with access to the hosting Github account is the owner of the account.

**8.4 Tools**

*(Identify the special software tools, techniques, and methodologies employed in the testing efforts. The purpose and use of each tool shall be described. Plans for the acquisition, training, support, and qualification for each tool or technique.)*

All developers will need :

1. JDeoderant
2. JAutodoc
3. JUnit
4. Eclemma
5. Google Codepro
6. PIT mutation testing software

(these are all Eclipse Plugins and can be downloaded from the Eclipse Marketplace)

**8.5 Risks and Assumptions**

*(Identify significant constraints on testing such as test item availability, test resource availability, and time constraints. Identify the risks and assumptions associated with testing tasks including schedule, resources, approach and documentation. Specify a contingency plan for each risk factor.)*

1. Risk: A developer is not able to download or receives errors when using a testing tool.

Contingency Plan: Other team members will help troubleshoot the issue and if it cannot be resolved: another developer will take over responsibility of the affected work.

2. Risk: A team member is not putting forth the agreed upon amount work/effort

Contingency Plan: The team member will be reminded that their performance will be honestly depicted on the peer report.

3. Risk: Team is not able to meet due to conflicting schedules.

Contingency Plan: If location of members is the only obstacle, we will meet meet via a video-conferencing tool.

4. Risk: Changes to source code from two developers overlap which creates a commit issue.

Contingency Plan: The two team members will work together to fix the overlapping changes issue.

5. Risk: A deliverable is not completed in time.

Contingency Plan: The entire group will communicate with each other to come up with an updated plan that accounts for remaining tasks.

# 9. CHANGE MANAGEMENT PROCEDURES

*(Identify the software test plan change management process. Define the change initiation, change review, and change authorization process.)*

Since we will be using a version control system (GitHub), the change management process will be simplified. However, to help prevent change collisions, each programmer will be responsible for a certain, non-overlapping segment of source code. Each segment will ideally have as little dependencies on other segments as possible.

If a change is required when working on the project, developers will have to budget their time. Developers will have to use their best judgement to see what segments of code are not as important to test as time becomes more limited. If by the end of the project, the developer has enough time to go back and work on that section, they will be expected to do so. If another developer is able, they may help with the section that got put off.