**AGE - Android Game Engine**

**Software Quality Assurance (SQA) Plan**

**Date:** 01/14/2013

**Team Members:**

Alex Riedl

Rebecca Vance

Max Witte

Austin Binish

Kyle Gustafson

Kris Kata

**Table of Contents**

[1.0 Purpose](#h.81iy4rv3vp5i)

[1.1 Scope](#h.jeuvyt6sqysf)

[1.2 Project Summary](#h.w9fdd7ffz9eh)

[2.0 Reference Documents](#h.epvjrzpge0te)

[3.0 Quality Goals and Expectations](#h.74fxfzqx7bw4)

[4.0 Software Reviews](#h.89xl3849hxfg)

[4.1 Purpose](#h.jecb8a7pjn1)

[4.2 Review Schedule](#h.envuodem7ilk)

[4.2.1 Code & Documentation](#h.qp1mdpgr8tsp)

[5.0 Test](#h.fwpnbc88ytz4)

[5.1 Purpose](#h.frrm3oj3afeg)

[5.1.1 Proof of Concept Testing](#h.4o8a6owurwog)

[5.1.1.1 Scope](#h.jcuq6lkcx6m2)

[5.1.1.2 Testing Description](#h.9mv56jsg2fgl)

[5.1.1.3 Responsible Party](#h.t5r52g7iq1fk)

[6.0 Problem Reporting and Corrective Action](#h.ngu4upkej2nx)

[6.1 Discovery](#h.sq4c4kxx8d6e)

[6.2 Action](#h.jm3q66xl697i)

[6.2.1 Immediate Correction](#h.fxg2j3muqsqm)

[6.2.2 Future Correction](#h.il7pbidvyij2)

[7.0 Tools, Techniques and Methodologies](#h.z97fzquyw1f4)

[8.0 Configuration Management](#h.axuyizaydcqk)

[8.1 Scope Creep](#h.cuuxzcduugtb)

[8.2 Software Release](#h.qnjnq7y56q00)

[9.0 Revision History](#h.w48ol67xnylp)

# 

# 

# 1.0 Purpose

The purpose of this Software Quality Assurance (SQA) Plan is to establish the goals, processes, and responsibilities required to implement effective quality assurance functions for the AGE project. The AGE Software Quality Assurance Plan provides the framework necessary to ensure a consistent approach to software quality assurance throughout the project life cycle. It defines the approach that will be used by the Software Quality personnel to monitor and assess software development processes and products to provide objective insight into the maturity and quality of the software.

## 1.1 Scope

This plan covers SQA activities throughout the implementation phases of the AGE.

## 1.2 Project Summary

The Android Game Engine (AGE) is a game engine made to be used on Android. It is used to help make the creation of games easier on that system. AGE will control game systems including but not limited to; graphics, collisions, state management, and audio. The reason this engine is being developed is to allow the easy creation of games on Android, and help developers focus on their game instead of potentially complicated extra code.

# 

# 

# 2.0 Reference Documents

The following documents were used or referenced in the development of this plan:

* IEEE STD 730-2002, IEEE Standard for Software Quality Assurance Plans
* Project Proposal, <https://msoe.fogbugz.com/default.asp?W1538>
* Standards, <https://msoe.fogbugz.com/default.asp?W1613>

# 

# 

# 3.0 Quality Goals and Expectations

Goal 1: Proper documentation of publicly visible classes.

Description: AGE is a tool that will be used by a variety of different people with different knowledge bases. As such, the documentation on how to use every class is needed and must be clear in its intent.

Goal 2: Internal code and documentation must follow their respective standards.

Description: Our team has created internal standards for the quality of code and documentation. It is expected that these will be followed to ensure high quality and consistency throughout the project.

Goal 3: Every AGE feature works on Android 2.3

Description: At the beginning of the project the team looked at which Android version would define our base. Android 2.3 was chosen because of its saturation in the market.

Goal 4: Tutorials and Walkthroughs will accompany the release.

Description: One of the easiest ways to start using a new tool is to follow the walkthrough or a tutorial that goes into detail on how to use it. A game engine is no different and the users of this tool will be expecting documentation on how to use it.

Expectation 1: The tool will be continued after its initial release.

Description: Game Engines are an evolving piece of technology. It cannot be expected that the initial release will include every possible feature that would be requested. Additional features will be added after the release.

Expectation 2: The tool will be continually documented.

Description: Since game engines are continuously updated and modified, the documentation and tutorials will need to be kept update date along with the code. This will ensure better understanding for the users of the game engine.

# 

# 

# 4.0 Software Reviews

## 4.1 Purpose

This section identifies the number and type of reviews and engineering peer reviews that will be performed. It describes the artifact types to be reviewed as well as the format of the reviews that will be conducted. These reviews have been scheduled on the WBS and accounted for in project planning.

## 4.2 Review Schedule

### 4.2.1 Code & Documentation

Every PBI that adds or modifies code in the project must have at least one code review from someone else. The code review is performed in the Kiln website interface. The most efficient way for the developer to be able to create the code review is to ensure to put “Case x:” as a prefix when pushing their code to the Kiln server. This will automatically link a changeset to the ticket in FogBugz, which will make it simpler on the developer when they need to create the code review. When the developer is believed to have completed their task, they will create a code review in Kiln and add the assigned code reviewer on the team. The reviewer then code reviews the module and inputs their comments. Upon completion of the review, the original author must review the code review and make changes as necessary. If there is confusion or disagreement on a comment, then the comment must be discussed by the two members or amongst the team for further explanation before the code review is marked as complete.

# 

# 

# 5.0 Test

## 5.1 Purpose

This section defines the types of testing and the scope of testing activities for this software development. Testing shall include both developmental testing as well as acceptance level testing. For each type, the scope of testing shall be defined as well as the responsible parties.

### 5.1.1 Proof of Concept Testing

#### **5.1.1.1** **Scope**

This is applicable to pieces of code seen as crucial for demonstrating a piece of functionality, but it is not encompassing of all new code added.

#### **5.1.1.2** **Testing Description**

Whenever functionality is added, a proof of concept should be made showing it in action. This serves as a functional test for its intended implementation and also as a regression test going forward since these should continue to work as new functionality gets added.

The proof of concept should test as many aspects of the additional functionality as is possible, including failure conditions if not done so elsewhere.

#### **5.1.1.3** **Responsible Party**

Whoever added the functionality is responsible for defining what goes into a proof of concept. They are responsible to add the needed functionality to the proof of concept to demonstrate that they fulfilled the requirements for the task they are assigned to. This means that the acceptance criteria of the task is fully met. It is the responsibility of all team members to check if the proof of concepts continue to work after adding new functionality.

# 

# 

# 6.0 Problem Reporting and Corrective Action

# 6.1 Discovery

Bugs can be identified at any point due to the nature of the agile process. Ideally, a focus on bug discovery is made during the review and testing process. Upon discovery of a bug the finding party is responsible for creating a new task within FogBugz and notifying the team of its discovery. The description of the new bug task should be made so that the bug can be reproduced if possible, and identified if it’s location is known.

# 6.2 Action

There are two actions that can be taken once a bug is known and which is used will be determined based on when and where the bug is found.

### 6.2.1 Immediate Correction

If the bug is found during testing or review of a backlog item that has not yet been completed yet or was tasked for completion during the sprint in which the bug was found, then the bug should be added as a subtask of the owning backlog item and must be completed in order for the backlog item to be considered complete. Product owners can use discretion here as the circumstance may be that the defect is not a high priority and the backlog item is needed in the next release.

### 6.2.2 Future Correction

If the bug is found after the owning feature was completed then the bug shall be added as a new backlog item and given a priority based on its severity. Discussion is advised for determining which sprint the bug shall be a part of.

# 

# 

# 7.0 Tools, Techniques and Methodologies

SQ personnel will require access to the following:

|  |  |  |
| --- | --- | --- |
| **Tool Name** | **Version** | **Purpose** |
| Eclipse | Juno | Development |
| FogBugz | N/A | Wiki, task management, time management |
| Kiln/Mercurial | N/A | Code repository, code reviews |
| Android SDK Tools | Latest | Testing |
| Java | 1.6 | Code |

# 

# 8.0 Configuration Management

## 8.1 Scope Creep

To avoid scope creep, every new backlog item must have a defined acceptance criteria stating what must be done in order for the feature to be complete. Every backlog item also undergoes a planning poker phase in which the team estimates how much effort and/or value the backlog item will need/contain. Using this information backlog items may be separated into smaller items if deemed too large, or removed altogether if they do not fit with the project.

## 8.2 Software Release

In Scrum, the end of each sprint is effectively considered a release. In order to accompany such a methodology we will be tagging the final result of each sprint as a release in Kiln. This will allow us to create a jar file using the tagged trunk for distribution to customers or publishing on the web.

# 

# 9.0 Revision History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Change Log** |
| 1.0 | 1/14/2012 | Initial Revision |