Jarvis Emulator  
Project Management Plan  
COP 4331, Fall 2015

**Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Who** | **Comment** |
| v0.0 | 9/13/2015 | Robin Schiro | Created document |
|  |  |  |  |

**Team Members:**

* Jimmy Lam
* Julian Rojas
* Manuel Gonzalez
* Robin Schiro

1. **Project Overview**
   1. This application uses facial detection and recognition to respond to users as they enter or exit a room. A webcam is set up to point toward the room’s entrance. When a user enters, the program can respond with an action that has been configured that user. The GUI of the application will allows users to set up profiles that specify how the program should behave. Based on this configuration, the application will communicate back and forth with the user to perform desired actions and provide useful information. If more than one person is in the room at once, the application will perform actions based on who is talking at any given time.
2. **Applicable Standards**
   1. **Coding Standard**
      1. Generally, we should follow the standard defined by Microsoft for .NET programming here. However, there are a few exceptions and additions:
         1. Do not use the ‘var’ variable type. It is always best to be as unambiguous as possible when declaring variables. Using ‘var’ only increases the chance of potential confusion.
   2. **Document Standard**
      1. Normal Font: Calibri Body, 11
      2. Headings:
         1. Document Title: Calibri Body, 14, Centered at top of document
         2. Major Section: Calibri Body, 14, Bold
         3. Subsection: Calibri Body, 12, Bold
      3. Spacing: Place a line break between every major section of the document
      4. Table of Contents: Include a Table of Contents if a document is longer than 10 pages.
         1. Format: The heading of each major sections should be left aligned and the corresponding page of each major section should be right aligned. There should be a line a periods separating each heading from its corresponding page number.
      5. Modification history: A table with columns labeled Version, Date, Who, and Comment will be placed at the top of every document to display the document’s modification history. Because the table has a ‘Who’ column, we do not need to maintain a separate list of authors’ names.
   3. **Artifact Size Metric Standard**
      1. Our team will use Size Points to determine the difficulty/amount of work involved in accomplishing a task. A Size Point (SP) corresponds to a realistic amount of time that any one of us might spend on the project in one day: 2 hours. 1 Size Point can be equivalent to 2 hours of work by a person with our average amount of experience. For example, a class that would take the average person on our team 8 hours to write would be measured at 4 SP.
3. **Software Life Cycle Process**
   1. Our group will adhere to a form of the Agile development process over the course of this project. This process divides development into a series of ‘sprints’ during which developers complete tasks that were assigned to them at the beginning of each sprint. Agile is very flexible due to ability of customers and developers to report bugs and create user stories during each sprint. We would like to gain experience using this process because of its ubiquity in the industry of software engineering and its proven success.
   2. We will have two-week sprints. At the beginning of each sprint, we will create several user stories and assign them to each team member.
   3. Weekly Status meetings- Describe what you did during the week, problems you had, what you plan to work on during the next week.
   4. Diagram:

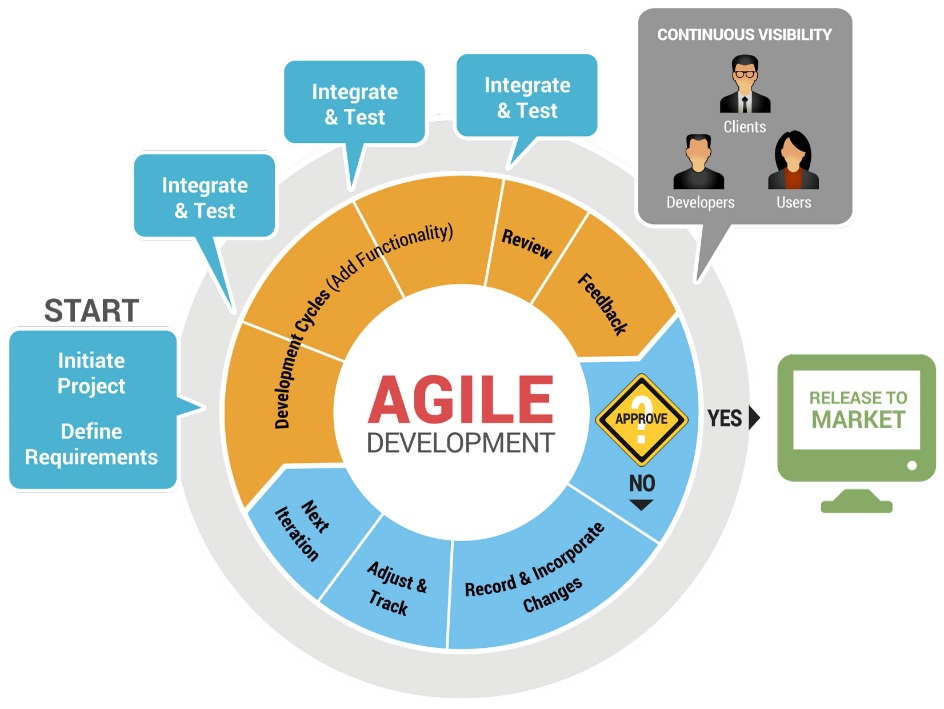


Figure : Representation of the Agile development cycle. Source: [STAGroup](http://www.stagrp.com/technology/application-solutions/)

1. **Configuration Management**
   1. **Version Control**
      1. We will use Microsoft Visual Studio’s Team Foundation Server as a repository for our code and documentation. This repository will be used in conjunction with Git to manage checkouts and commits.
      2. Each member of the team will be responsible for minimizing conflict between pushes to the repository. Communication with the other members of the team is key in order to guarantee the stability of the repository.
   2. **Commit Process**
      1. Before working on an issue, the developer must create a branch for that issue. This is done to prevent conflict when multiple developers are working on the project simultaneously.
      2. As the developer works on an issue, he should make several commits (depending on the size of the issue) to document the history of changes he makes to the relevant code files.
      3. Once the developer has resolved the issue, he should merge his local branch with the main branch. He must be sure to communicate with the rest of the team before performing a merge.