Homework #1: StackApp

CPSC 544: Advanced Software Process, Section 50

Instructor: Dr. Chang-Hyun Jo

Team 03: Caliware

Date Submitted: 9/28/15

|  |  |  |
| --- | --- | --- |
| Timothy Cioffi-Dinkel | Anthony Farina | Joanna Hang |
| timothycioffi@csu.fullerton.edu | afarina@csu.fullerton.edu | johang@fullerton.edu |
| https://lh5.googleusercontent.com/PSroDgY8ZOUkmCV05kKayV-boTdpJSN4FXzGMZ1w6lwE5Br5w20a5pWMewmiTxHAznLEpv6jhuOyAIByZrQkCyPaceyEEcTEmPBQpFilW6976BlGvS6ZX0VZf9cAObv8=s1600 |  | sig.jpg |

|  |  |
| --- | --- |
| Lourdes Lopez | David Sullivan |
| lourdeslopez@csu.fullerton.edu | dsullivan@fullerton.edu |
| Untitled.png | https://lh5.googleusercontent.com/3FGNbflMbPEB_7oxSHs8FcMMr_Lp5KykcuOeEjncWoizP2v-jn12POyZmVra2fZCxjqKTL1FQzXUmPlmarUW9EyxEf-ii570A3UFlycFawZP6PRs9L7I6VX2bGtZIwEE8Ws=s1600 |

# **Revision History**

|  |  |  |
| --- | --- | --- |
| Date | Title | Team Member |
| 8/29/15 | Created document outline | Timothy Cioffi |
| 9/1/15 | Work Products | Anthony Farina |
| 9/5/15 | User stories | Lourdes Lopez |
| 9/5/15 | Project Process, Roles | David Sullivan |
| 9/6/15 | Work Practices | Anthony Farina |
| 9/7/15 | Roles | David Sullivan |
| 9/7/15 | Vision | Timothy Cioffi |
| 9/7/15 | Added to Vision | Joanna Hang |
| 9/8/15 | Requirements, Technology Preparation | Joanna Hang |
| 9/8/15 | Technology Preparation | Anthony Farina |
| 9/20/15 | Iteration I | Lourdes Lopez |
| 9/22/15 | Risk Management, Project Planning, References | Joanna Hang |
| 9/23/15 | Release Planning, Release | Anthony Farina |
| 9/23/15 | Configuration Management | Timothy Cioffi |
| 9/25/15 | Work Products, Roles, & Practices | David Sullivan |
| 9/25/15 | Process | Joanna Hang |
| 9/25/15 | Iteration II | Lourdes Lopez |
| 9/26/15 | Acceptance Testing, Iteration III, User Stories, Product Backlog and Estimates | Lourdes Lopez |
| 9/26/15 | Release | Anthony Farina |
| 9/26/15 | Project Monitoring and Control | Lourdes Lopez |
| 9/27/15 | Quality Assurance | Timothy Cioffi |

Table of Contents

[Revision History 2](#_Toc431236609)

[1 Project Process 4](#_Toc431236610)

[2 Definition of Work Products, Roles, and Practices 7](#_Toc431236615)

[3 Pre-Game: Planning and Staging With Vision 13](#_Toc431236644)

[4 Requirements 16](#_Toc431236652)

[5 User Stories 19](#_Toc431236658)

[6 Products Backlog and Estimates 20](#_Toc431236659)

[7 Architectural Spike 23](#_Toc431236663)

[8 Release Planning 24](#_Toc431236664)

[9 Technology Preparation 26](#_Toc431236670)

[10 Development 28](#_Toc431236671)

[11 Release 40](#_Toc431236709)

[12 Estimation 45](#_Toc431236712)

[13 Project Planning 46](#_Toc431236713)

[14 Project Monitoring and Control 48](#_Toc431236717)

[15 Configuration Management 49](#_Toc431236720)

[16 Quality Assurance 50](#_Toc431236725)

[17 Risk Management 52](#_Toc431236729)

[18 Lessons Learned 53](#_Toc431236737)

[19 References 55](#_Toc431236744)

[20 Team Charter 57](#_Toc431236745)

[21 Team Evaluation 61](#_Toc431236746)

[Appendix A - Sketches 62](#_Toc431236747)

[Appendix B - Timeline 63](#_Toc431236748)

[Appendix C - List of Tables and Figures 64](#_Toc431236749)

# Project Process

Our organization, Team Caliware, will develop an Android-based stack application utilizing the agile process by incorporating both Scrum and Extreme Programming (XP) methodologies. Our process is detailed below.

Agile is an iterative process that many organizations practice in software development projects. Agile was created in response to evolving and rapidly changing requirements from customers. This process differentiates from the traditional waterfall process, an assumed linear and sequential methodology, which relies too heavily on having a full set of complete and accurate requirements from the beginning of the software development process (Jo, n.d.-e).

Two methodologies of the agile process are Scrum and XP. To develop our Stack application, Caliware has selectively chosen some practices from Scrum and XP that best represents the agile process and that best fits the needs of our organization. Both methodologies complement each other well in our customized software process model as Scrum focuses on team and project management whereas XP focuses on the technical aspects of software development (Mar, 2015).

By combining both Scrum and XP practices, our software process best represents the agile process: a collaborative, client-driven process that is adaptive to change and produces working software at the end of each iteration. With this process, we will deliver a high quality product that satisfies our customers. The Caliware software process is detailed further below.

## Caliware Software Process Model

The Caliware Software Process Model consists of three phases that was derived from Scrum and XP: pre-game, development, and release. These three phases will be furthered detailed later.

The following table on the next page identifies the lifecycle of our model, including the phases, purpose of each phase, and the necessary activities for each phase (Jo, n.d.-3 & Jo, n.d.-f):

|  |  |  |  |
| --- | --- | --- | --- |
| **Pre-Game** | | **Development** | **Release** |
| **Planning**  Purpose:  -establish the vision and set expectations (Scrum)  -enough well-estimated user stories for first release (Scrum)  -feasibility ensured  (XP)  Activities:  -write vision, initial product backlog and estimate items  (Scrum)  -exploratory design and prototypes  (Scrum)  -user stories and estimating  (Scrum)  -define/establish roles | **Staging**  Purpose:  -identify more requirements and prioritize enough for the first iteration  (Scrum)  -agree on date and stories for first release (XP)    Activities:  -planning  (Scrum) | Purpose:  -implement a system ready for release of a 1 week iteration (Sprints)  (Scrum)  -implement a tested system ready for release (XP)    Activities:  -sprint planning meeting each iteration, defining the Sprint Backlog and estimates  (Scrum)  -daily Scrum meetings  (Scrum)  -Sprint Review  (Scrum)  -testing and programming (XP) | Purpose:  -operational deployment (Scrum and XP)    Activities:  -documentation (Scrum and XP)  -training (Scrum and XP) |

Table Caliware Software Process Model

### Pre-Game

For the Pre-Game phase of our Process Model, we divided this area into two parts: planning and staging.

#### Planning

In the planning section of the Pre-Game phase, the purpose is to establish the vision and set expectations for developing the product. The Scrum Master and Product Owner complete these. Once completed, the Scrum Master informs the team about the vision. The vision is crucial, and all team members must the know vision in order to understand the product that needs to be built. This phase also includes ensuring feasibility, meaning the team members must understand that they are capable of completing the product. User stories are also created in this phase, which set the initial requirements for the product. Thus, the activities in this section include writing the vision, creating user stories, creating the initial Product Backlog, and estimating items. Exploratory designs and prototypes are also created.

#### Staging

In the staging section of the Pre-Game phase, the purpose is to identify more requirements and prioritize enough requirements for the first iteration. This means the user stories are ranked and chosen for the first iteration. With these requirements in hand, the team can agree on the date of the first release. These are the planning activities from Scrum.

### Development

The purpose of the development phase is to start implementing a system that is ready for release within a one-week iteration. Although the suggested practice for iteration length is 30 days, our team shortened the iteration to one week in order to reach the project’s deadline. The following activities take place during the development phase.

#### Sprint Planning

Before the first iteration can begin, the Sprint Planning must be completed. First, the Product Owner describes the objectives of the upcoming Sprint. The Product Owner picks the user stories that need to be implemented by refining and reprioritizing the Product Backlog and Release Backlog (Jo, n.d.-f). These selections turn into the Sprint Backlog. Then, the developers break down the user stories into short, estimated development tasks. The development team members discuss the time they need to work on the items selected from the Sprint Backlog. The Scrum Team to discuss *what* can be delivered in the upcoming Sprint and *how* the incremental work will be achieved.

#### Sprint

During the one week Sprint, the development occurs. The Scrum Master will hold a daily Scrum meeting only with the Development Team. This is a 15-minute stand-up meeting for the development team to discuss the progress of the work in the Sprint Backlog. The team discusses the work that was completed, the work that will be completed, and if there are any impediments or blocks to development.

During the Scrum, each member answers these questions:

* What have you done since the last Scrum?
* What will you do between now and the next Scrum?
* What is getting in the way (blocks) the iteration goals?
* Is there any tasks relevant to the current iteration that should be changed or needs to be added?

Further discussions can continue after the Scrum meeting to go into more detail after the 15-minute meeting.

#### Sprint Review

At the end of each Sprint, there is a Sprint Review, a review meeting for the Scrum Team, in which there is a demo of the product to all the stakeholders (Jo, n.d.-f). During this Sprint Review, the developers listen to feedback only, and no further commitments made. Making commitments would also contradict the XP principle of not adding any additional work to the iteration. Instead, commitments are discussed at the next Sprint Planning meeting.

#### Testing and Programming

During the actual development of the product, the Development Team begins programming product and extensively testing the product. During the development, the team follows the XP principle of “do the simplest thing that could possibly work" (Wells, 1999). This ensures the development team will deliver a working product later in the release.

### Release

The purpose of the release phase is to deliver a working product to the customer. The release can be either internal or external. Upon release, there is documentation and training about the product.

# Definition of Work Products, Roles, and Practices

Work Products, Roles, and Practices are essential to the software process. The combination of work products, roles, and practices define the performance and the outcome of the software process. Caliware will define each work product, role, and practice of the software process along with the reasoning behind the inclusion of each.

## Work Products

Work Products are non-software products that are used and created during all stages and categories of the software process. These products help the team manage the project during the gathering of requirements, design, and project management. Below are the work products that Caliware will produce throughout the project. Because our team meets remotely, some of the work products are documents that are shared online.

### User Stories (Scrum)

User stories are short and simple statements from the user that describe the functionality/feature that they expect the system or product to perform. User stories are written in a template:

As a <who>, I want <what (feature)>, so that <why (value)>

From this user story, the team can talk to the user to get a more refined story and then proceed with creating acceptance tests for the user story. These stories will help the team extract requirements. User stories will be collected during the pre-game stage of the process. Additionally, user stories can be added at the beginning of each iteration.

### Product and Release Backlog (Scrum)

The Product and Release Backlog (spreadsheet) is a list of the all the requirements, features, enhancements, defects, and features that need to be developed and are gathered from the user stories. The Product and Release Backlog will be filled in during the Pre-Game - Planning stage. The backlog items may be refined when additional information is found during building the product in the Sprint is based on what the customer discovers after they review the product after each Sprint. The items at the top of the backlog or more refined that items at the bottom of the backlog and are placed in the order of priority.

The advantages of utilizing the Product Backlog are brief meetings, increased user involvement from all participants, and increased focus on the goals. This ensures the requirements are in line with what the customer needs.

### Task Board (Scrum)

A task board can be in the form of a whiteboard, a section of a wall, or in Caliware’s case, an online-shared spreadsheet since we are working remotely. The task board consists of four columns, “Story”, “To Do”, “In Progress”, and “Done”. The rows consist of the user story and the tasks that need to be complete the user stories for the iteration. Each column also includes the complexity of the task, hours estimated, hours remaining, and the assigned developer. The task board is filled in at the beginning of each iteration, and the tasks are moved from left to right as when the tasks are currently being worked on or completed.

The Task Board is a useful, visual, and interactive way to present the tasks that need to be done in the current sprint. During the Daily Scrum, the Scrum Master and the Development Team will bring up the Task Board for the discussion of work that has been completed and work yet to be started or work that could still be in progress.

### Task list (XP)

A task list contains is a list for all the tasks that need to completed within the iteration. Our task list has been combined with the task board that is used in Scrum. The Scrum Master and the developers at the beginning of each iteration fill out the task list.

### Sketches (XP)

As part of the Exploration Phase of XP development, sketches are a good way to create an initial design so the development team can start programming more quickly. For this process, sketches will be either basic hand drawn sketches of the user interface or user flow.

Sketches can help developers model the system design in a visual way before development in order to help see how the requirements can be implemented. XP discourages more than 10 - 20 minutes in design to begin coding. The development team along with input from the Scrum Master will create these sketches. A few sketches of the ‘StackApp’ can be seen in Appendix A.

### Sprint Backlog with Volunteering (Scrum & XP)

A Sprint Backlog (spreadsheet) lists all of the tasks for the current iteration. It also allows the team to keep track of the estimated work remaining for each task and all tasks overall. The team will limit the effort estimate in ideal engineering hours. The designated person will update it daily. A volunteer instead of being assigned will choose each task. This leads to a higher degree of commitment and responsibility.

### Sprint Burndown Chart (Scrum)

A Burndown Chart is a visual summary of the days in the Sprint compared to remaining tasks that are stated in the Sprint Backlog.

X axis = outstanding work

Y axis= time (days)

This collection of metrics allows the team to see trends and adjust accordingly. The Sprint Backlog graph is set up in the Sprint Planning stage and the beginning of each iteration/sprint. It is updated throughout the iteration so the team can see the trends. The Development Team and Scrum Master are responsible for the backlog graph.

## Roles

The team will volunteer for roles in the software development process. The roles will differ and interchange during different iterations. Each role is defined as follows:

### Scrum Master (Scrum)

The Scrum Master is the servant-leader of the Scrum Team (Scrum Alliance, n.d.). This Scrum master does not assign work items to the team members. The Scrum Master facilitates the meetings and ensures Scrum procedures are correctly followed by the Scrum Team for agility. The Scrum Master acts as a firewall between management and the development team and removes any impediments that are brought up by the Development Team during Daily Scrum meetings. This important service to the team will handle any issues that the developers are stuck on and provide any resources as needed by the developers. All parties will accept this authority of this role and abide by the Scrum Master’s procedures. For example, the Scrum Master will not allow people outside the Developer circle to overwhelm the team with additional requirements during the Sprint. In addition, the Scrum Master will provide the Development Team a knowledge expert to seek any clarifications on requirements to eliminate misunderstood requirements. The Scrum Master will also take on the role of the **Tracker** who collects all of the metrics, tracks progress, and provides feedback on poor estimates.

### Development Team (Scrum & XP)

The Development Team is responsible for performing the work in the Spring Backlog. There are no other titles on the Development Team other than “Developer”. The members are responsible for the Sprint result as a team and make all decisions about how the work will be accomplished. After the development team has the backlog, the programmers volunteer to select the tasks to code. Scrum does not cover development best practices or procedures. The assumption is that developers can determine for themselves best how the actual development is done based on their own experience and procedures (self-directed teams). This means that management must trust the Scrum process, and self-management within the Development Team is encouraged.

### Programmer(s) (XP)

The programmer(s) is a part of the development team. The programmer will design and code the application. The programmer will use the practice of pair programming while programming.

### Tester(s) (XP)

The tester(s) is a part of the development team. The tester will help write and develop tests for each release. The tester will assure that the product has high quality. The tester will document bugs or inconsistencies in the product and will make sure that they are addressed.

### Product Owner/Customer (Scrum & XP)

The product owner, or customer, is the responsible person writing stories, picking stories, defining the priorities of the work items of Product Backlog, but does not directly manage the activities of the Development Team. In terms of product development, this is the customer role who will provide the goal and objectives of the current sprint and will review the work after an Sprint. The customer initiates the project and drives the project. The customer decides what needs to be developed for the next iteration.

## Practices

The software process consists of many practices from Scrum and XP. These practices help the organization produce the desired work products. All stages and categories of the software process use practices. The Scrum Master facilitates all of the practices. Below are the practices that Caliware will be performing will be listed.

### Pre-Game Planning/Release Planning Game (Scrum & XP)

In this practice, all stakeholders will meet and gather a list of features, enhancements, defects, etc. which are all recorded in the Product and Release Backlog. The Product Owner/Customer will be responsible for filling out the backlog in the form of user stories. The product owner determines the business value of each item and prioritizes the items on the list. The Developers provide story points to estimate the amount of work in the backlog. Then the Scrum Team will divide and negotiate the work based on what can be done in each Sprint Release. This is also the practice where user roles will be defined (e.g. Product Owner).

### Sprint Planning/Iteration Planning Game (Scrum & XP)

In this practice, the planning of the actual iteration will occur. The Scrum Team will meet to determine the priorities and the level of importance for each requirement. The Product Owner will clarify the goals of the iteration. User stories will selected for the current iteration and will be broken down into small tasks by the development team. The Product and Release Backlog will be refined and updated with new tasks. The Sprint Backlog will be created, and the team will volunteer to do certain tasks. This practice starts in the Pre-Game stages of the process. The Scrum Master will take the lead with the development team providing input as well.

### System Metaphor and Simple Design (XP)

This practice will help define how the organization perceives the entire system to work after development. This is also the practice where the team will describe how they intend to build the system. From this system metaphor, simple design will occur with basic models/sketches being created. This occurs in the beginning of the project in the Pre-Game stages. Everyone on the team will provide input for this practice.

### Frequent Refactoring (XP)

Frequent refactoring helps the product become less bloated and more efficient increasing its quality. Frequent refactoring means removing redundancies, eliminating unused code, and replacing less efficient code with a more efficient design. The programmers are the main contributors to frequent refactoring. This practice is done during the development stage.

### Pair Programming (XP)

Two programmers work together with one computer (or shared a screen if remote). One programmer will write the code while the other observes and reviews the code. This allows less bugs/problems to occur since there is another set of eyes on the code. All programming will occur as pair programming. This also allows the observer to learn something that they may not have known. This is down by the programmers/development team during the development stage.

### Unit Testing/Testing (XP)

Unit tests are essential to the development process. They enable refactoring and frequent integration and they provide a safety or tests. This will allow the team to save a lot of time throughout development by finding bugs and incompatibilities before being implemented. Unit tests are the first things created by the programmer during the development stage.

### Acceptance tests (*Scrum*)

Since Caliware’s process has user stories, acceptance tests will also be used. Acceptance tests are directly attached to a user story. These tests represent a certain outcome that the user story expects. If the system has the desired output, then the user story will be complete. These tests are done throughout the development stage by the development team as well as by the customer as part of each release.

### Sprint Review (Scrum)

The Scrum Master will host a meeting, which will review the product at the end of the iteration. The meeting attendees (i.e. team, product owner, and stakeholders) will provide feedback and brainstorm on future decisions. The Sprint Review is done at the end of the development stage, during the release stage.

### Coding Standards (XP)

Coding standards help eliminate confusion when multiple people are programming in the same project. If one programmer needs to work on a module that another programmer was working on, there should not be any trouble understanding what the other programmer was doing. Coding standards will be agreed up during the Pre-Game stages of the process and will be practiced throughout development.

### Team Code Ownership (XP)

Every programmer will be working all parts of the project. This allows every programmer to understand all parts of the system, which will elevate the team.

### Self-Directed and Self-Organizing Teams (Scrum)

This practice empowers the team to use their own authority and resource to guide their own work. This also relates to the volunteering since these self-directed teams will volunteer to do tasks that they think they can do the best. This team is formed during the Pre-Game stages of the process.

### Daily Scrum/Scrum Meeting (Scrum)

This practice will be held at the same time and place every day. Since Caliware is geographically dispersed team, the “place” will be Google Hangouts instead of a physical room. All the rules of the Scrum meeting are still in place. In these meetings, each member of the team answers the same five questions. These five questions are:

1. What have you done since the last Scrum?
2. What will you do between now and the next Scrum?
3. What is getting in the way (blocks) of meeting iteration goals?
   1. If there are any blocks, these blocks must be fixed within a day and decisions on the blocks must be made within one hour. No decisions are worse than a bad decision.
4. Any tasks to add to Task List? Any missed tasks?
5. Have you learned or decoded anything new, of relevance to some of the team members?

This is good practice to keep track of the progress of the project and to address any lingering problems.

Rationale

Although daily scrums are to be held every working day in our case, the daily scrum meetings are held every other calendar days. For our team, two calendar days are the equivalent of one working day.

### Continuous Integration (XP)

The team will embrace change so the team can quickly respond to change. Since requirements will have unforeseen elements and changes, the team must be ready for it and embrace it. GitHub is the chosen Configuration Management tool for continuous integration. Anyone developers can access and modify the code. They can be alerted when there new changes that are added. They can “clone” the latest version to their computer.

Caliware will create a test project inside Android Studio. The tests will built alongside new code and features and are checked-in together. The test project will be built frequently to insure that the functional tests are still working with the addition of new code.

# Pre-Game: Planning and Staging With Vision

The pre-game for this application is divided into segments in order to establish the groundwork for a product to ensure the elimination of chaotic production and provide a customer demanded guideline for developers to follow to produce an application within the timeline requirements.

These segments for pre-production include: A vision to establish what we see the product becoming by release; Requirements demanded by the customer, both functional and non-functional; User-Stories to keep iterations on track and specify development requirements at that time, Product Backlog and Estimates to provide a timeline for expected development and actual development estimates; Architectural Spikes, which identifies any potential risks and halts in the development process; Release planning in order to provide development with a general idea of when the release should take place; and Technology preparation in order to allow for development an idea of what technologies they will need to understand before delving into the engineering of the application.

## Vision

The application created by our team will be named ‘StackApp’. The customer ordered an application, which creates a stack from the user’s inputted variables and outputs the created stack to the interface. This will be achieved by creating a box for the user to enter the desired variable with a button to input the entered variable into the system. Once inputted, the application will display the created Stack.

### Goals of the Product

The goals for this project have three parts: One is to create a working Android application, which allows for user input and displays correct stack output. Second, along with a working algorithm, a user interface must be created which provides a simple, easy to use inputting area for the user’s variables, as well as, an easy to read output, clearly displaying the stack for the user. Thirdly, a strict quality assurance testing must be created to verify that the developed application fulfills the parameters set by the customer, as well as, checks for bugs within the system.

### The Goal of Project

The overall goals of the building the product is to learn about the software development process, learn and practice how to define a process, learn agile processes utilizing Scrum and Extreme Programming (XP) practices, and to build a mobile application for a mobile device using the process defined.

### Goals of our Company

Our company must follow two principles: One, always develop a product that is in line with the customer’s specifications. In the end, the customer is paying for the product and will expect the product to be developed to their requirements. Two, develop an application that is near perfection. A product should be as bug-less as possible as to continue having business with the customer and to develop a great reputation within the industry for which our company is trying to produce.

### System Features

The application, and the development process, will have certain features that will ensure a quality product for the customer. There will be five features of the application, these include: the application must work on Android Mobile Systems, an easy to use user interface, basic user input, output produced after an input, and a testing process.

First, the application must work on Android Mobile Systems. Our customer has ordered an application that must run on Android devices successfully, and without this basic feature, we will lose business.

Secondly, a user interface must be created that will be easy for the user to operate and will display the desired outputs in a way that is both correct and easy for the user to see. For the purposes of simplicity, as requested by the customer, the GUI will consist of a basic input text box and a number pad underneath for user inputs. Besides that, a basic banner will be created with the title of the application as well as a button, which allows for the exiting of the program.

Thirdly, the application must have basic user input functionality. The customers has required that the application prompt the user for an input, and with certain keywords entered by the user, produce a certain output. The inputs include: “push”, “pop” and “exit,” which will push the entered variable to the stack, pull out the integer from the stack, or exit the application.

Fourthly, the outputs produced after a user input are also specified by the customer. These include: A pushed message and display the stack with the new variable listed first, a popped message, which displays the stack with the old first variable in the list removed, a full message if the user attempts to push to a stack of three items, an empty message if the user attempts to pop a stack with zero items, and an exit message when the user enters “exit.”

Fifthly, a testing process, which will ensure the application’s functionality. Unit tests should be created to test for all combinations inputs in the customer’s desired range (0-9). These unit tests can be utilized to find issues with both the system’s list that is being created, and the displayed list which is shown to the user. Another unit test, which should be created, will test the bounds of the application. The customer has required that the stack follow two specifications: One, the stack must only be able to hold three items. Two, the stack must only hold values between zero and nine. The test should show that the application both does not push the integer if out of bounds or if the stack is full, and if the proper error message is being displayed. There will also be manual GUI testing which will go through all of the available UI options provided to the customer, and test for working functionality and ensure a bug-free interface.

### System Context Diagram

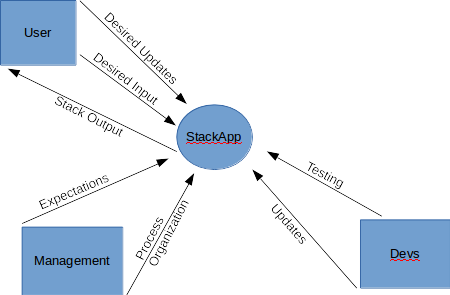


Figure System Context Diagram

### Stakeholders

There are three main stakeholders with the development of this application; the first one is the customer. The customer is monetarily investing into our business and if a working application is not being produces, the customer will be dissatisfied by having wasted their money and their time investing in other companies.

The second stakeholder is our company. If we cannot produce a well-developed application within a timely manner, the customer will no longer do business with our company. This means that we will lose money but also this means our business will have a poor reputation for other customers within the industry.

The third stakeholder is the entire development team. If a working product is not produced, or if the product development goes beyond the length of the time-boxed iterations, those on the team may lose their jobs entirely or may never be provided with an opportunity to develop a critical project.

# Requirements

## Definition of a Stack

The “StackApp” target system, in layman terms, is to have an Android application which simulates a stack. To better understand the target system, a description of a stack needs to be examined or reviewed.

A stack is a data structure that implements “Last-In-First-Out” (LIFO). There are a few operations that can be performed on a stack. These operations are called “Push” and “Pop”. To “Push” a stack, is to add an element to the end of the stack. To “Pop” is to remove the most recently added element from the stack. For example, the table below shows operations being performed on the stack in the left column, while showing the stack in the right column:

|  |  |
| --- | --- |
| Operation | Stack |
| - | [ ] |
| Push ‘3’ | [3] |
| Push ‘ 5’ | [3,5] |
| Pop | [3] |

Table Operation and Stack

## Target System

With the definition of a stack known, along with the user stories provided by the customer/product owner, the team can give a detailed description of what the target system of the ‘StackApp’.

The ‘StackApp’ has four main functions: Pop, Push, Clear, and Exit. The Pop function will remove the element while applying the LIFO principle. If the stack is empty, then the user will be notified that the stack is empty and cannot be popped. The Push function will push/add an element to the stack while applying the LIFO principle. Via user stories, there are a few restrictions to the Push functions. These restrictions include restricting the type of element that is pushed to the stack to just integers ranging from 0-9 and to limit the amount of elements in the stack to three. If the user runs into these restrictions, an error message will be displayed describing the restriction. The Clear function will clear the entire stack. The Exit function will exit the application. During all of these functions, the stack and the operation will be displayed to the user.

Obviously, the system must be 100% reliable at all times. A stack has strict limitations and must be followed or the main function of the application will be compromised. The ‘StackApp’ must perform at a high level by having a quick response time. Along with having high performance, the system must be readily available at all times.

This target system was created/influenced by using functional requirements, nonfunctional requirements, and architecturally influential factors. These are all described in separate sections below.

### Functional Requirements

Functional requirements are a description of a behavior that a system will exhibit under specific conditions (Wiegers & Beatty, 2013). The functional requirements for the ‘StackApp’ are as follows:

1. Add numbers
   1. Add numbers in an array to align values
   2. Integer must be between integer set 0-9
   3. Push integer-value
2. Store stack sequence
   1. Maximum stack size is 3
3. Remove numbers
   1. Pop number while applying LIFO principle
4. Display results
   1. Display stack contents with LIFO principle
   2. Give error message if input does not reach min/max numbers, ask for correction
5. Quit
   1. Quit application

### Nonfunctional Requirements

Nonfunctional requirements are a description of a property or characteristic that must show restrictions when the system is operated (Wiegers & Beatty, 2013). These non-functional requirements are listed below in rank of importance.

1. Reliability
   1. Application has to work 100% of the time.
2. Performance
   1. Response time: Output latency should be no greater than 1 second.
3. Availability
   1. Application must be available 24/7.
4. Scalability
   1. If the customer wants to adjust the stack size it should be possible.
5. Manageability
   1. Efficient deployment, updating, and testing.

### Architecturally Influential Factors

Architecturally influential factors are requirements that impact the basic structure of the system. The AIF’s for the ‘StackApp’ system are:

1. Android system
   1. System should be a mobile Android application
2. Modifiability
   1. System should be easy to modify.
3. Usability
   1. User interface should be simple and visually appealing.
4. Network dependability
   1. Application is fully functional offline

# User Stories

The team wrote the first draft of the user stories during the planning phase. The team met with the customer, reviewed, and discussed the requested features and expected functionalities. Both the team and the customer came prepared to the meeting by learning how to write user stories prior to the planning session. Since the requirements were not too complex, the team was able to write all of the user stories during the planning phase. Stories were written with acceptance criteria, so that the developers had a definition of what was required in order to determine how to measure the completion of the story. The customer then ranked user stories in order of priority. The development team estimated Story Points based on complexity and effort.

All user stories fell under one epic: **As a customer, I want an Android (LIFO) stack application, so that I can have it easily accessible on my Android device.**

Prior to the start of the second iteration, the team conducted a Product Backlog review with the customer. Stories were reviewed and prioritized once more. At this point, the customer added two more stories to the Product Backlog if time and resources allowed for the development of additional stories.

During the next Product Backlog review, prior to the last iteration, the customer once again added a story to the Product Backlog. This additional functionality was a result of the customer having seen a demo at the end of the second iteration.

All User Stories, as of Iteration III:

|  |  |  |  |
| --- | --- | --- | --- |
| Who | What | Why | Acceptance Criteria |
| As a customer | I want to be prompted to add integers to a list | so that I can create a stack | Input must be an integer between 0 and 9. |
| As a customer | I want a limit on the stack size | so that the stack remains small | The limit for the stack should be 3. An error message should display when the limit is reached. |
| As a customer | I want to have the stack displayed after each operation | so that I can see what is in the stack | Operation and integer should be displayed. |
| As a customer | I want a popup | so that I am notified if the stack is empty | An error message should display when the stack is empty. |
| As a customer | I want to be able to remove integers from the list | so that I can make room in the stack for new integers | The last integer entered into the stack must be removed. |
| As a customer | I want to be able to cancel | so that I can exit the stack | Exit the application |
| as a customer | I want a reset button | so that I can clear the stack | Clear the stack |
| As a customer | I want to be able to increase the stack size | so that I can add more numbers | Input integer to determine size |
| As a customer | I want to have my product logo on the app | so that I can brand the app | display in the navigation bar |
| As a customer | I want an about section | so that the user will have information about the app | click on the product logo to see the about information |
| As a customer | I want a number only keyboard | so that it will be easier to enter integers | display number only keyboard |

Table User Stories

# Products Backlog and Estimates

The Product Backlog was initially created during the planning phase. Upon each Sprint Review, the Product Backlog was updated, and priorities were reviewed for accuracy. Prior to the start of each iteration, during the planning meeting, stories were taken from the Product Backlog and placed into the Sprint Backlog for development. Story points are an estimate of the stories complexity and were established by the development team. Stories with a higher complexity that had a high risk of an architectural spike were given higher story points. Story points are only comparable between stories within the Caliware development team and are based on our experience and skill level as a team. Estimate hours are development hours and were established during the iteration planning sessions.

## Sprint I

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | Category | Reference | Story Points | Priority | Estimate hours | Status |
| push | User Story | User Story 1.1 | 1 | 1 | 6 | In Progress - Sprint 1 |
| limit stack size | User Story | User Story 1.2 | 1 | 3 | 3 | In Progress - Sprint 1 |
| display stack | User Story | User Story 1.3 | 3 | 2 | 3 | In Progress - Sprint 1 |
| popup when stack is empty | User Story | User Story 1.4 | 1 | 6 | 2 | Product Backlog |
| pop | User Story | User Story 1.5 | 3 | 4 | 2 | Product Backlog |
| exit | User Story | User Story 1.6 | 3 | 5 | 1 | Product Backlog |
| clear | User Story | User Story 1.7 | 1 | 7 | 3 | Product Backlog |

Table Sprint I Product Backlog

## Sprint II

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | Category | Reference | Story Points | Priority | Estimate hours | Status |
| push | User Story | User Story 1.1 | 1 | 1 | 6 | Completed in Sprint 1 |
| limit stack size | User Story | User Story 1.2 | 1 | 3 | 3 | Completed in Sprint 1 |
| display stack | User Story | User Story 1.3 | 3 | 2 | 3 | Completed in Sprint 1 |
| popup when stack is empty | User Story | User Story 1.4 | 1 | 6 | 2 | In Progress - Sprint 2 |
| pop | User Story | User Story 1.5 | 3 | 4 | 2 | In Progress - Sprint 2 |
| exit | User Story | User Story 1.6 | 3 | 5 | 1 | In Progress - Sprint 2 |
| clear | User Story | User Story 1.7 | 1 | 7 | 3 | In Progress - Sprint 2 |
| increase the stack size | User Story | User Story 1.8 | 3 | 8 | 2 | Product Backlog |
| product logo | User Story | User Story 1.9 | 1 | 9 | .5 | Product Backlog |
| about section from logo | User Story | User Story 1.9.1 | 3 | 10 | 2 | Product Backlog |

Table Sprint II Product Backlog

## Sprint III

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | Category | Reference | Story Points | Priority | Estimate hours | Status |
| push | User Story | User Story 1.1 | 1 | 1 | 6 | Completed in Sprint 1 |
| limit stack size | User Story | User Story 1.2 | 1 | 3 | 3 | Completed in Sprint 1 |
| display stack | User Story | User Story 1.3 | 3 | 2 | 3 | Completed in Sprint 1 |
| popup when stack is empty | User Story | User Story 1.4 | 1 | 6 | 2 | Completed in Sprint 2 |
| pop | User Story | User Story 1.5 | 3 | 4 | 2 | Completed in Sprint 2 |
| exit | User Story | User Story 1.6 | 3 | 5 | 1 | Completed in Sprint 2 |
| clear | User Story | User Story 1.7 | 1 | 7 | 3 | Completed in Sprint 3 |
| increase the stack size | User Story | User Story 1.8 | 3 | 9 | 2 | In Progress - Sprint 3 |
| product logo | User Story | User Story 1.9 | 1 | 10 | .5 | In Progress - Sprint 3 |
| about section from logo | User Story | User Story 1.9.1 | 3 | 11 | 2 | Product Backlog |
| number only keyboard | User Story | User Story 1.11 | 1 | 12 | 2 | Product Backlog |
| fix display bug | Bug Fix | BUG 1 | 1 | 8 | 1.5 | In Progress - Sprint 3 |

Table Sprint III Product Backlog

# Architectural Spike

During the planning phase of the project, the team discussed potential architectural spikes. We identified two potential architectural spikes, which were prior knowledge for the majority of the team in regards to Android development and unit testing for Android applications. We chose to work towards addressing these risks early on within our first iteration.

The first risk for an architectural spike was overcome by studying, practicing tutorials and learning about Android development, prior to the first iteration, as well as by leveraging XP’s pair programming technique. Caliware modified the pair programming technique to meet our team’s needs. XP pair programming suggests having a senior developer work side by side with a junior developer, for our team we had a developer with the most knowledge of Android development work through the first task while the development team participated. This allowed the development team to expand on the foundational knowledge by discussing coding style, best practices and standards during this session.

The second risk for an architectural spike was addressed towards the end of our first iteration. The team researched and learned about unit testing. Since we had one team member with prior knowledge of unit testing for Android applications, this team member worked through the first unit-testing task and then shared the solution and discussed with the team. Once again, best practices and standards were discussed.

Upon completion of the first iteration, we were confident as a team that the risks were properly addressed and where able to make more accurate time estimates going forward.

# Release Planning

## Product Backlog

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | Category | Reference | Story Points | Priority | Estimate Hours |
| push | User Story | User story 1.1 | 1 | 1 | 6 |
| limit stack size | User Story | User Story 1.2 | 1 | 3 | 3 |
| display stack | User Story | User Story 1.3 | 3 | 2 | 3 |
| popup when stack is empty | User Story | User Story 1.4 | 1 | 6 | 2 |
| pop | User Story | User Story 1.5 | 3 | 4 | 2 |
| exit | User Story | User Story 1.6 | 3 | 5 | 1 |
| clear | User Story | User Story 1.7 | 1 | 7 | 3 |
| increase the stack size | User Story | User Story 1.8 | 3 | 9 | 2 |
| product logo | User Story | User Story 1.9 | 1 | 10 | .5 |
| about section from logo | User Story | User Story 1.9.1 | 3 | 11 | 2 |
| number only keyboard | User Story | User Story 1.11 | 1 | 12 | 2 |
| fix display bug | Bug Fix | BUG 1 | 1 | 8 | 1.5 |
| Totals |  |  |  |  | **28** |

Table Product Backlog

## Release Structure

Releases will be attempted after every sprint. A more detailed structure of releases for the ‘StackApp’ is as follows:

### First Release (Internal)

The first release will be internal. The first release will allow the user to push integers into the stack. It will also validate the input to make sure the input is following the rules.

* User Stories: 1.1, 1.2, 1.3
* Total Estimation: 14 hours

### Second Release (Internal)

The second release will be internal. This release should resemble the final product in almost every way. It will alert the user during certain events. It will allow the user to clear or pop the stack. Lastly it will allow the user to exit the application. This release could be used for a final release.

* User Stories: 1.4, 1.5, 1.6, 1.7
* Total Estimation :  19

### Third/Final Release

The third release will be the final release. It will provide all the functionality that the user requested and extras, including the ability to increase the stack size and an ‘About’ section so users can learn more about the application.

* User Stories: 1.8, 1.9, 1.9.1,1.11, Bug 1
* Total Estimation:  6

# Technology Preparation

Every new project brings along new challenges. One of these challenges is technology preparation. In order to have a successful project, the organization must be technologically prepared before development. In the beginning planning stages of the project, the team gathers all the necessary tasks, requirements, and restrictions and begins to devise a plan to ‘tackle’ development.

To develop the ‘StackApp’, according to the user requirements, the development team had to learn how to use the Android Platform. The development team referred to the Android 6.0 Software Development Kit (SDK). While some developers already had experience with developing for Android, other developers had to complete training materials and watch tutorials, some of which are listed below.

* <http://developer.android.com/index.html>
* <http://developer.android.com/sdk/index.html>
* <http://developer.android.com/tools/workflow/index.html>
* <http://developer.android.com/training/basics/firstapp/index.html>
* <http://developer.android.com/tools/samples/index.html>
* <http://www.lynda.com/Android-tutorials/Developing-Android-Apps-Essential-Training-Revision-Q2-2015/369905-2.html>

Android development requires basic knowledge in Java and XML. For developers that did not have previous experience with these languages, the used resources such as Lynda.com to gain the essential knowledge needed to begin developing for Android. Some of the Lynda.com courses that were used were:

* Java Essential Training
  + <http://www.lynda.com/Java-tutorials/Java-Essential-Training/377484-2.html>
* XML Essential Training
  + <http://www.lynda.com/XML-tutorials/XML-Essential-Training/145930-2.html>

In the beginning of the project, the team decided to use the Eclipse Integrated Development Environment (IDE), Eclipse Mars to develop the ‘StackApp’. The reasoning for this decision was that some developers already had experience using Eclipse for developing Android applications, its ease of use, and because it supports Android.  Android has a custom plugin for Eclipse called Android Development Tools (ADT). Via the official Android Developer website, ADT…

*“provides a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to let you quickly set up new Android projects, build an app UI, debug your app, and export signed (or unsigned) app packages (APKs) for distribution.”*[[1]](#footnote-1)

Eclipse and ADT provided all the necessary tools to complete this project including the most recent Android Software Development Kit (SDK). The ADT plugin required a newer version of Eclipse, the Eclipse JDT Plugin (included with Eclipse Mars), and at least Java Development Kit (JDK) 6.

To learn about Eclipse, the team used various online resources including:

* http://wiki.eclipse.org/Eclipse\_Articles,\_Tutorials,\_Demos,\_Books,\_and\_More

However, mid-project, it was deemed necessary to use the latest version of Android Studio (Android Studio 1.3.1) to develop the Android application. This IDE was selected because it is the official IDE for Android development and it offered all the necessary elements that were needed to complete this project including the Android SDK. Official support of the Eclipse plug is going to stop by year’s end, which is why the team decided to switch to Android Studio. The import process from Eclipse to Android Studio was problem free.

There are massive amounts of online resources that helped the development team learn how to use Android Studio such as:

* <http://developer.android.com/tools/studio/index.html>
* <http://developer.android.com/tools/studio/studio-config.html>
* <http://developer.android.com/sdk/installing/studio-tips.html>

For revision control and source code management, the team decided to use GitHub. Many of the team members did not have experience using GitHub so the resources provided by GitHub were useful.

Along with all the online resources that the team used to learn about all the necessary topics, the team also used on the developers that already had previous knowledge in the topics to prepare for the project. This provided some helpful tips and helped with nuances with Android Development that some of the online resources did not provide.

# Development

## Iteration I

Iteration I began upon completion of the planning and staging phase. The first iteration was time-boxed iteration that lasted six days. Overall, the first iteration was a learning experience for our team, in that we did not have much prior knowledge of the Scrum and XP processes. After having learned about both agile methodologies and defining our process as a team, this was our first round of actually working through the exercises, producing the work products and artifacts. Most of the team members were also new Android development, which added another layer of learning. Furthermore, getting to know our team members strengths, weakness, and areas of expertise helped us grow stronger as a team as we were able to share our knowledge to improve our overall skills.

### User Stories and Sprint Backlog

Story selection for the first iteration was client-driven and based upon client prioritization of the stories. However, since the potential risks identified by the team were based upon a learning curve for Android development, we were also able to include risk-driven selection into the first iteration.

#### Sprint Backlog:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Who | What | Why | Acceptance Criteria | Story Points | Rank | Status |
| As a customer | I want to be prompted to add integers to a list | so that I can create a stack | Input must be an integer between 0 and 9. | 1 | 1 | Completed in Sprint 1 |
| As a customer | I want a limit on the stack size | so that the stack remains small | The limit for the stack should be 3. An error message should display when the limit is reached. | 1 | 3 | Completed in Sprint 1 |
| As a customer | I want to have the stack displayed after each operation | so that I can see what is in the stack | Operation and integer should be displayed. | 3 | 2 | Completed in Sprint 1 |

Table Iteration I Sprint Backlog

### Iteration Planning

Our iteration planning session began with agreeing upon the duration of the sprint. We determined a start and end date as well as our available development hours. We reviewed and determined a time slot for our daily Scrum meeting in order to ensure that we selected a time that would fit well with all of our schedules.

Once the sprint parameters were established, we reviewed the Product Backlog and selected the stories for the Sprint Backlog. As a team, we read and discussed user stories and their acceptance criteria. Since the Product Backlog was small, we were easily able to review all stories in the Product Backlog. This gave us an overall understanding of the entire product requests, and we were able to quickly identify any potential architectural spikes.

We determined that during the first sprint, as a team, we had nine development hours available. We took the story with the highest priority and began writing tasks. Once the tasks were completed for the first story, we estimated development and complexity for each task. These estimates were a consensus between all developers on the team. If we came across an estimate where we did not all have the same estimate, then we discussed the reasoning behind each of our estimates and then voted again until our estimates were in alignment. This method of estimating helped us learn from our team members prior experiences. Once the story with the highest priority was complete with tasks and estimates, we moved on to the next story and repeated the process until we reached our capacity of nine development hours.

Based on the stories that were selected, we determined that the sprint would end with an internal release.

### Managing

Managing the first sprint began with a daily Scrum meeting. Since our team members are located in different geographical areas with various time zones, we met via Google Hangouts. Daily Scrum meetings were kept under 15 minutes, in which we reviewed the tasks that we worked on prior the Scrum meeting and identified any issues. We then discussed what tasks would be worked on next. At the end of each Scrum meeting, we updated the Burndown Chart and were able to use the Burndown Chart as an indicator that our sprint was on target.

A spreadsheet was used to track information for Sprint I including the Sprint Backlog, sprint tasks with status and hours, and the Burndown Chart. The spreadsheet was stored as a shared document and was accessible to all team members for updating.

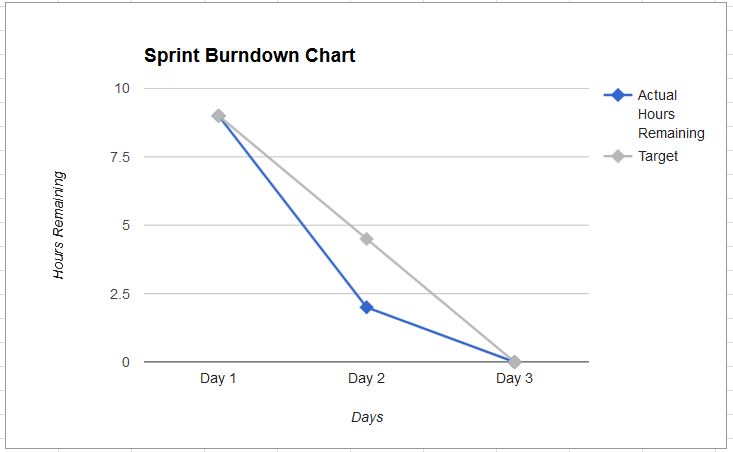


Figure Iteration I Burndown Chart

### Designing

Design decisions were crucial in the first iteration, as they would provide a foundation for future iterations. UI design was based on the customer's expectations. Code design was based on the best practices knowledge from our team member who had prior Android development experience. Although the application was small, we took architectural design for code and testing seriously as it was a learning experience for the rest of the development team.

### Coding

Caliware implemented XP’s pair programming for our development tasks. While it was only necessary to have one person coding and one other person provide feedback, Caliware as a group, decided to design our pair programming sessions with all team members present. The reason for this is that we had two team members, Anthony and Timothy, with a strong background in Android development. By having the entire team present, we were all able to rapidly learn the foundational skills needed for Android development as well as best practices and team coding standards. Pair programming sessions were held via google hangouts.

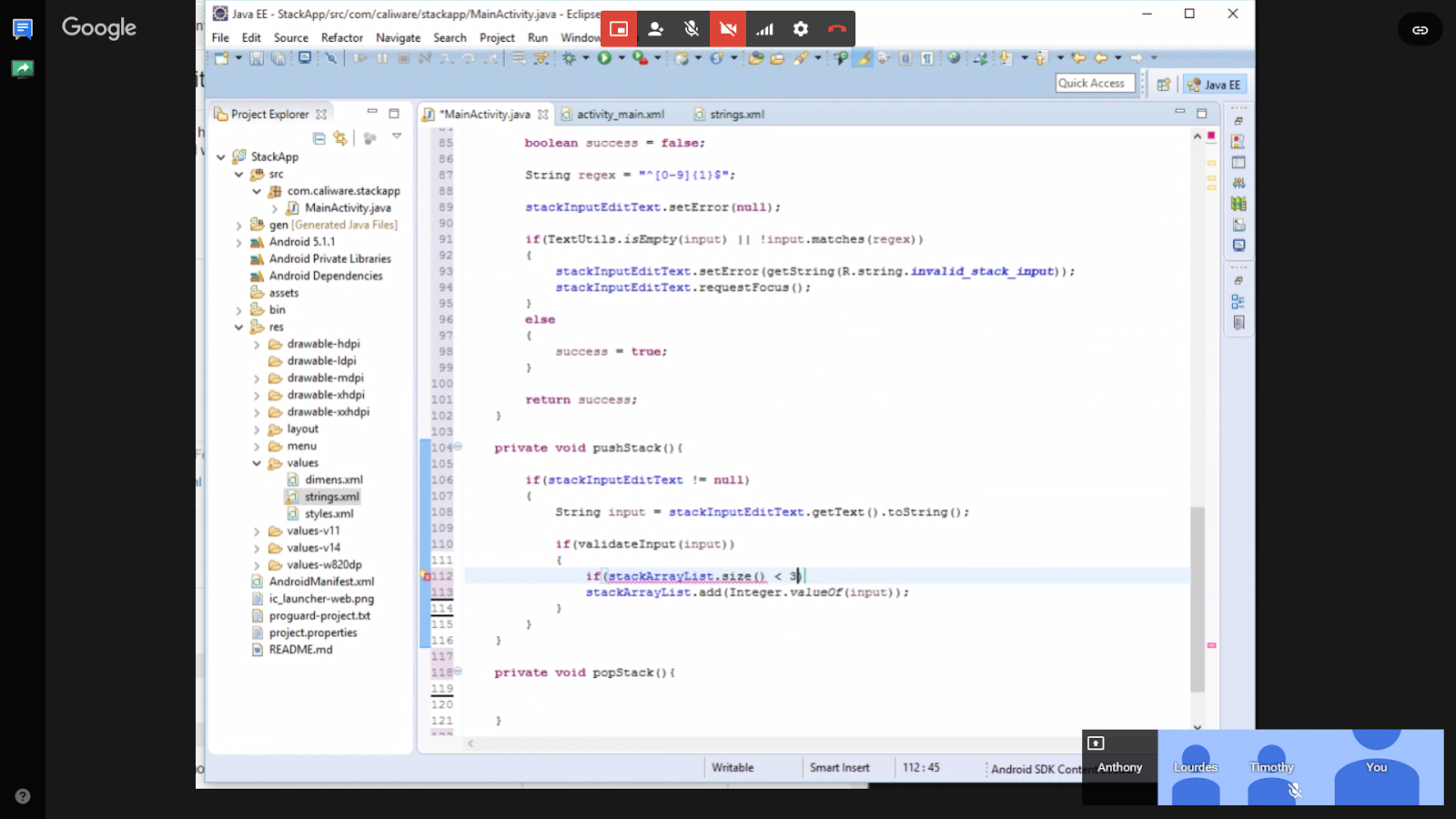


Figure Google Hangout Pair Programming

### Testing

Testing during the first iteration was given high importance, in that we had to determine a strategy and standards for our testing. We had one team member who had previous experience with testing Android application using the JUnit framework and we were able to leverage his expertise for our project. The unit tests were used to test individual method to ensure that they worked properly. Additionally, we tested for scenarios in which we expected the method to fail. By testing both pass and fail scenarios we were confident that the methods would pass the acceptance criteria. We did not test for every possible scenario, but aimed to capture the most probable scenarios.

At the end of the iteration, we also performed testing by running the application on one of our phones and running through various scenarios.

### Small Releases

Caliware had the first internal release of the ‘StackApp’ Android application. During this release, we demoed the current status of the application internally. The application is functional with most of the features that were requested during the planning stage present. These features include:

* Creating a stack
* Pushing an accepted value into the stack
* Disallowing certain characters into the stack
* Displaying the stack
* Disallowing certain stack ranges

Below you can see a few screenshots of the first release:

|  |  |  |
| --- | --- | --- |
| main_screen.png  Figure Main Screen | pushed_stack.png  Figure Pushed Value to Stack | full_stack.png  Figure Max Stack |

The internal feedback from the first release was positive. The release was bug free and was very responsive. All parties were pleased with the release.

### Sprint Review

At the end of our first sprint, we conducted a sprint review with the customer. During the Sprint Review, the customer was informed of the successful completion of all Sprint Backlog user stories in the first sprint. Caliware was on target and looking forward to begin the next sprint.

### Sprint Retrospective

Upon the completion of all Sprint I work products, and ceremonies, the team came together for a Sprint Retrospective in which we evaluated the iteration. We reflected upon iteration and asked ourselves, what went well, what went wrong and what can we do better.

* What went well?
  + We completed the tasks in our sprint on time. We did not encounter any roadblocks and were able to complete the majority of the tasks well within the beginning of the sprint.
* What went wrong?
  + No issues were reported in regards to development. As far as the iteration planning we identified that our iteration was too long.
* What can we do better?
  + Shorten the duration of the next iteration.

### Backlog Review

In preparation for the next sprint, we met with the customer to re-evaluate the Product Backlog. This gave the customer the opportunity to adjust priorities. Given our velocity and capacity, we identified that we had room to add additional stories to the Product Backlog for possibly being included into our last iteration. This allowed the customer to add additional functionality to the original requested features. Some negotiation occurred so that the new feature requests did not deviate too far from of the original project scope.

## Iteration II

In second iteration, we took what we learned from the first iteration and built upon it. Our process was clearer and easy to follow as this was the second time going through the process for our team and we had a point of reference. Overall, we only came across one impediment that was identified on the last day of the iteration and was resolved within the next working day. We found that not only our velocity had increased, but also we were able to quickly resolve errors that we came across during our pair programming sessions.

### User Stories and Sprint Backlog

Story selection for the second sprint was customer-driven. With an improvement in the team's ability to estimate tasks, we were able to select more stories for the iteration in comparison to the previous sprint.

#### Sprint Backlog:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Who | What | Why | Acceptance Criteria | Story Points | Rank | Status |
| As a customer | I want a popup | so that I am notified if the stack is empty | An error message should display when the stack is empty. | 1 | 3 | Completed in Sprint 2 |
| As a customer | I want to be able to remove integers from the list | so that I can make room in the stack for new integers | The last integer entered into the stack must be removed. | 3 | 1 | Completed in Sprint 2 |
| As a customer | I want to be able to cancel | so that I can exit the stack | Exit the application | 3 | 2 | Completed in Sprint 2 |
| as a customer | I want a reset button | so that I can clear the stack | Clear the stack | 1 | 4 | Completed in Sprint 2 |

Table Iteration II Sprint Backlog

### Iteration Planning

The iteration planning session for the second sprint began once again with an agreement upon the parameters of the iteration.  As a team, we decided to make and adjustment to the length of the iteration and shortened the iteration by 2 days. This decision was made based upon recommendations from the sprint retrospective from the prior iteration.

Once the length of the iteration was determined we were then able to determine our capacity in terms of available development hours. As a team we had a total of 8 development hours available.

We then proceeded to review the stories in order of customer priority. We created the tasks and placed our estimates until we reached without exceeding the available hours. We were able to include 4 user stories with a total estimate of 7.5 hours into our second iteration. We discussed any potential architectural spikes and found that this iteration was low risk.

### Managing

We continued to use the same tools and process for managing Sprint II that we used in Sprint I, which were the daily Scrum, a burndown chart and a shared spreadsheet for tracking task hours and status. Our daily Scrum meetings consisted of reviewing what tasks were completed the previous day, what tasks we were going to work on next and what barriers if any. For the most part, we were on target during the iteration but as we approached the final day of the sprint, we ran into an issue with the tasks for unit testing. Since the tasks were not completed until the day after the iteration ended, the Burndown chart reflects a slower velocity on the third day and we are able to see that we did not burn down to zero hours.

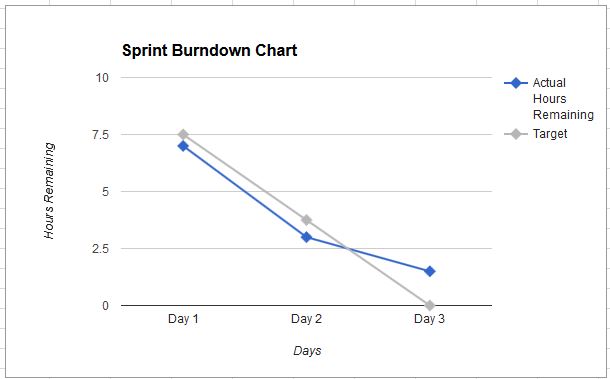


Figure Iteration II Burndown Chart

### Designing

The team continued to build upon design decisions that were made during the first iteration in regards to code and architectural design. UI design was once again based on user stories, acceptance criteria, and user requirements.

### Coding

During the second iteration, we continued to practice pair programming since we found that it worked so well for us during the first iteration. This time we were able to switch roles and have the experienced developer watch as a team member with less Android application development experience wrote the code. The experienced developer ensured that best practices and team standards were used. We did find that we came across more errors during the pair programming development session but the team as well as the senior developer was there to help work through the errors.

### Testing

For this iteration, since no one on the team had any experience in testing android user interfaces, our testing process was halted in an attempt to actively learn how to test android GUIs. This was a great learning opportunity for the team to get a full grasp of how the android user interface works and how to test it to its fullest.

### Small Releases

At the end of the second iteration, all functionality in the original scope of the project was completed and version 1 was ready for external release to the customer. Features that were completed during the second iteration include:

* A popup error message when the stack is empty
* Popping the most recently added value from the stack -  LIFO
* The ability to exit the application via the home menu
* The ability to clear the stack

### Sprint Review

At the end of the iteration, we met with the customer to review the features that were completed during the iteration. The customer was able to see a fully functional product will all features in the original scope of the product completed. During the demo section of the Sprint Review, the team identified a functionality that was not working as per the acceptance criteria from a story that was completed in the first iteration. The bug was noted, to be discussed in more detail during the backlog review.

### Sprint Retrospective

Upon completion of the second iteration, we met as a team to assess the sprint and look for areas of improvement.

* What went well?
  + We were able to accomplish more stories in a shorter amount of time in comparison to the first iteration.
* What went wrong?
  + In the middle of the second iteration we had to migrate our code from Eclipse to Android Studio. We did not notice any issues, but this put extra emphasis on our testing.
* What can we do better?
  + As a team, improving our knowledge on UI testing can help increase our velocity.

### Backlog Review

In preparation for the third iteration, we met with the customer to review the remaining user stories in the Product Backlog for prioritization. Upon having seen the application, the customer added more stories to the Product Backlog. In addition to the user stories, the customer prioritized fixing the bug that was found during the sprint review. At this point, the bug was looked at in more detail. The steps to recreate the bug were identified and documented. We found that upon pushing or popping a number to or from the stack, the stack was displaying but not the action.

## Iteration III

Prior to beginning the third iteration, the ‘StackApp’ application met the requirements that were discussed during the initial planning phase. The team had additional time and resources, which allowed for development of additional features based on user stories that were identified throughout the backlog reviews. During the third iteration, our knowledge of the process continued to increase as we became more comfortable and familiar with the various work products, artifacts, roles, and activities. We continued to increase our technical knowledge as well.

### User Stories and Sprint Backlog

User stories for the third iteration included two user stories and one bug fix.

#### Sprint Backlog:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Who | What | Why | Acceptance Criteria | Story Points | Rank | Status |
| As a customer | I want to be able to increase the stack size | so that I can add more numbers | Input integer to determine size | 3 | 2 | Completed in Sprint 3 |
| As a customer | I want to have my product logo on the app | so that I can brand the app | display in the navigation bar | 1 | 3 | Completed in Sprint 3 |
| (BUG FIX)  As a customer | I want the display message fixed | so that I can see the action as well as the number | Application should display action as well as the number that was either pushed or popped | 1 | 1 | Completed in Sprint 3 |

Table Iteration III Sprint Backlog

### Iteration Planning

The iteration planning session for the third iteration began with determining the available development hours. We chose to keep the length of the iteration consistent with the previous iteration since it worked well for our team and schedules. We selected stories based on customer priorities, wrote the tasks, and gave tasks estimates until we reached our available development hours.

### Managing

Tools and process for managing the sprint remained consistent with the previous two iterations. We once again used daily Scrum meetings, a task board, a burndown chart and Google Hangouts for pair programming.

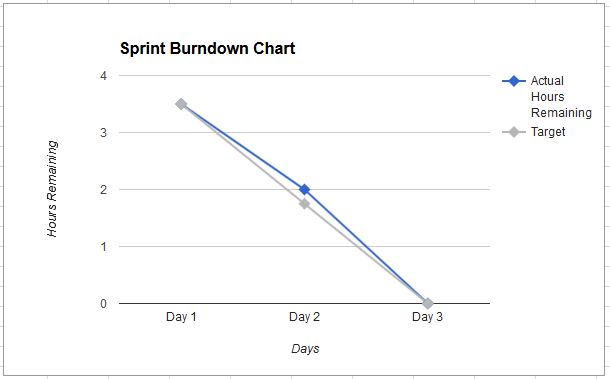


Figure Iteration III Burndown Chart

### Designing

UI design continued as per described in the user story and acceptance criteria. During this iteration, we came across a story that was not clear on the design so we reached out to the customer for clarification. The customer was readily available and easily accessible. We needed clarification on where to add the functionality for increasing the stack size. The customer chose to have this on the setting menu as a drop down item.

### Coding

We continued to use pair programming for programming. Caliware’s best practices and coding standards continue to be improved upon as we continue to code.

### Testing

Testing continued as per Caliware’s standards. No major issues were encountered during testing for the third iteration.

### Small Releases

Features for the third iteration release included:

* Allow the user to increase the stack size
* Add the product logo to the application
* Resolve bug so that the action is displayed upon popping or pushing an integer into the stack

### Sprint Review

The Sprint Review session with the customer included a review of what was completed during the third iteration as well as a demo of the product. This was our most exciting review as we were presenting a completed product. The customer was pleased with the overall product and functionality.

### Sprint Retrospective

* What went well?
  + Overall we were happy with the quality of the product that we produced.
* What went wrong?
  + User story and acceptance criteria for one of the stories needed more clarification.
* What can we do better?
  + Improve process for acceptance criteria.

### Backlog Review

During the allocated time for the backlog review with the customer, the remaining items were reviewed. At this point in time, there were no more resource hours available for the project. The customer decided not to request additional hours as they were pleased with the product as is and did not have additional funds for additional resources. The customer was transitioned over to Caliware’s support department. A survey in regards to their experience with Caliware’s development team was sent to the customer so that we can identify areas for improvement for future engagements.

## Acceptance Testing

At the end of each sprint we had the customer participate in acceptance testing. The customer reviewed the stories and acceptance criteria that were completed in the sprint and ensured that all acceptance criteria was met by UI testing and performed the requested functionality. The customer not only tested the new features, but also ensured that any existing functionality was still working as it was before.

### Sprint 1 Acceptance Testing

* Test adding a value to the stack
* Test to ensure that validation did not allow to add non numeric values to the stack
* Test to ensure validation only allowed integers [0 to 9]
* Ensure that the stack was displayed

### Sprint 2 Acceptance Testing

* Test adding a value to the stack
* Test to ensure that validation did not allow to add non numeric values to the stack
* Test to ensure validation only allowed integers [0 to 9]
* Ensure that the stack was displayed
* Ensure that the user receives a popup error message when the stack is empty
* Ensure that upon pushing the pop button the last value added to the stack is removed
* Test exit from the home menu
* Test that the stack is cleared

### Sprint 3 Acceptance Testing

* Test adding a value to the stack
* Test to ensure that validation did not allow to add non numeric values to the stack
* Test to ensure validation only allowed integers [0 to 9]
* Ensure that the stack was displayed
* Ensure that the user receives a popup error message when the stack is empty
* Ensure that upon pushing the pop button the last value added to the stack is removed
* Test exit from the home menu
* Test that the stack is cleared
* Test increasing the stack size
* Test to ensure that the stack size cannot be set to a negative number
* Ensure that the product logo is visible on the app
* Ensure that the action and number are displayed upon pushing or popping a number

# Release

The third release of the ‘StackApp’ will be the final release. This version of the application will be distributed to the customer. To accompany the final release, training documentation, operating documentation, etc. will be provided.

## Release Procedure

After the development team approves the final release of the application, the quality assurance department, and ultimately the customer/product owner the process of building the final version of the application can begin. Ultimately, this means that the team needs to build the APK (Android Application Package) of the ‘StackApp’ (Building and Running from Android Studio, n.d.). This file will be distributed to the customer/product owner.

### Downloading and Installing

Caliware will provide the download link for ‘StackApp’. Once the APK file has been downloaded to an Android phone, an installation pop up should requesting permission to install. After this permission has been accepted, ‘StackApp’ will be installed and be ready for use.

## Training/Operating

One of the team’s main objectives was to make the ‘StackApp’ easy to use and easy to learn. You will see that this is evident when operating the application. Below you will see companion documentation for each function that will be encountered while using ‘StackApp’.

### Opening ‘StackApp’:

1. Go the Application Drawer
2. Click on the ‘StackApp’ icon

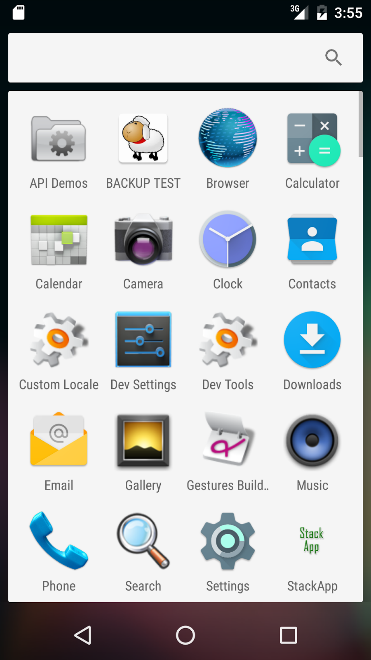


Figure Training - Open StackApp Icon

### Main Screen:

|  |  |
| --- | --- |
| Main Screen.png  Figure Main Screen | main screen 2.png  Figure Main Screen with Numerical Pad |

### How to Push:

1. Enter a valid single digit number into the input box
2. Press the ‘Push’ button
   1. If an invalid number was entered, an error will pop up
   2. If the stack is full, an error will pop up

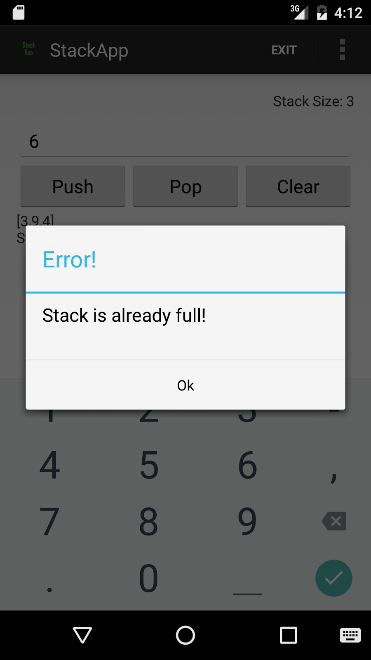
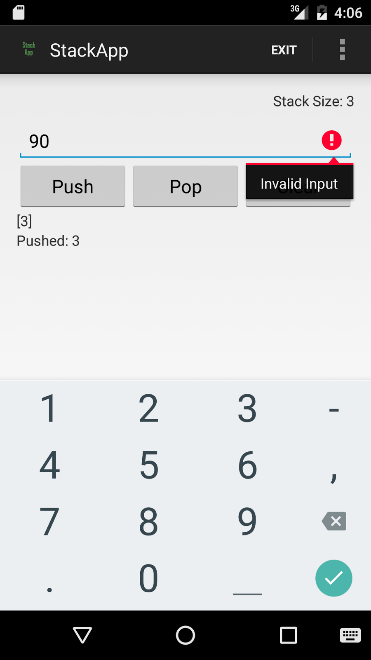
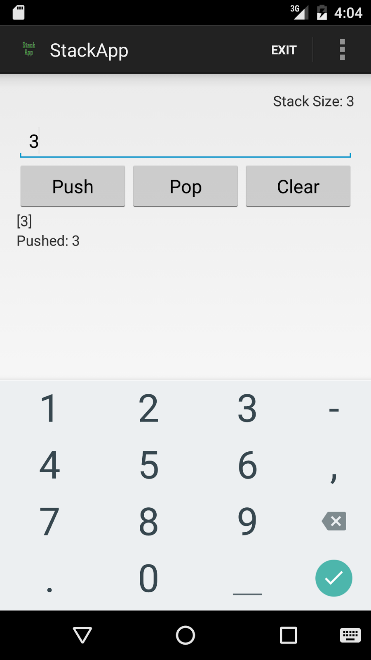
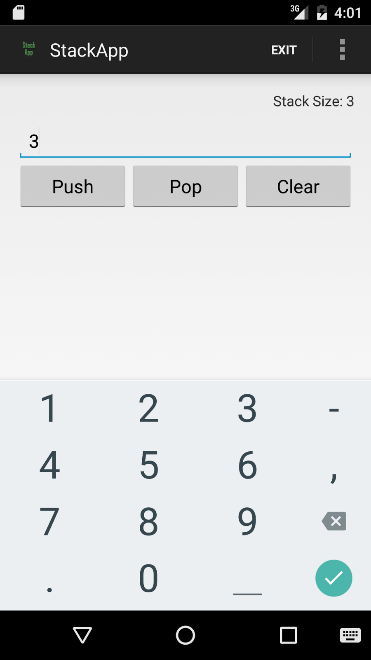


Figure How to Push

### How to Pop:

1. Press the ‘Pop’ button
2. If the stack is empty, an error will pop up

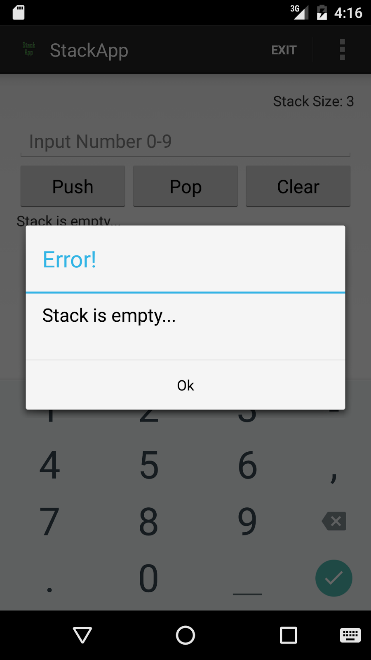
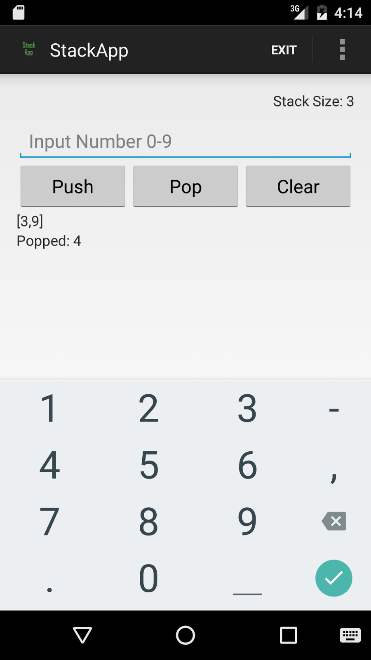


Figure How to Pop

### How to Clear:

1. Press the ‘Clear’ button

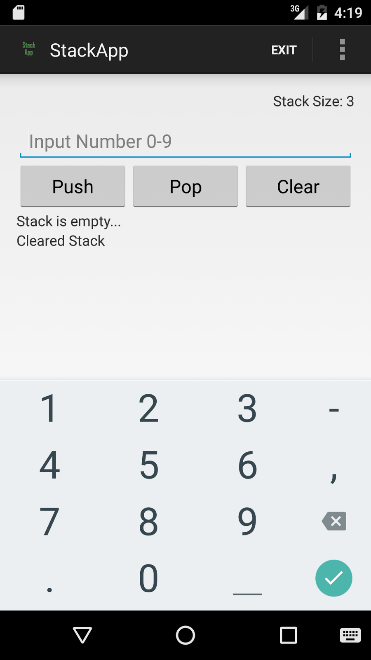


Figure How to Clear

### How to Exit:

1. Press the ‘Exit’ button in the action bar
   1. This will clear the stack and revert the stack size to three

### How to Change Stack Size:

1. Press the Menu/Action Overflow Button
2. Select the ‘Change Stack Size’ option
3. Enter in desired stack size limit
4. Press ‘Enter’
   1. If the new stack size limit is lower than the previous stack size limit, the stack will clear
   2. The new stack size will be displayed in the main screen at the top right

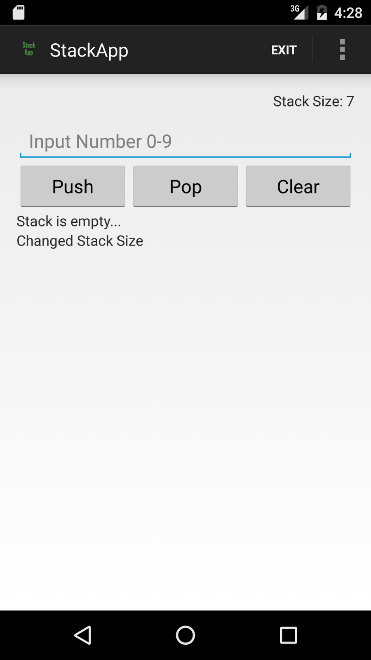
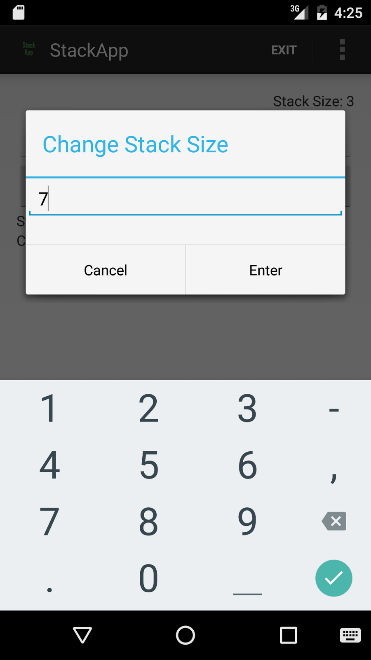
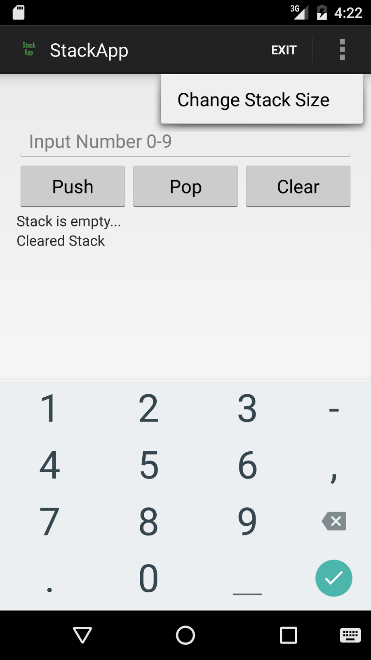


Figure How to Change Stack Size

# Estimation

The work of estimation is coordinated by the Scrum master but involves the coordination of all the members of the development team. Before an iteration begins, the Scrum Team meets to work with the stakeholders to discuss the user stories in the backlog. The team must answer the question, “What can be done in the current iteration” because is it assumed that the Development Team knows the most about what it takes to create the functionality.

The Development Team performs the actual work of estimation. The method, which our development team used, is to build user stories and assign the weight using a Fibonacci number sequence as tool to conceptualize the amount of effort required for each user story. The Fibonacci numbers used are 1, 2, 3, and 5 which makes it easier to think of the estimation in an abstract way rather than thinking in terms of hours.

The user stories are written in the customer’s own words using clear and simple language in the following format:

As a <who>, I want <what (feature)>, so that <why (value)>

For example:

As a **customer**, I want **to be prompted to add integers to a list**, so that I can **create a stack**.

In addition, we state the acceptance criteria on which to base our requirements

Input must be an integer between 0 and 9.

Next, the Development Team assigns story points using a Fibonacci number to estimate the effort of each user story. The estimation is done as a team because one developer might think of some aspect that is more complicated than the other team members did not consider. The user stories are ranked in the order of priority.

From the user stories, the Development Team decides how much of the high priority tasks it can complete during the next iteration. Following the selection of user stories, the selected stories are broken down into individual tasks so that the developers can focus on how to accomplish the task and to make it easier to estimate the hours they think is required. The tasks are added to the **Not Started** column in the Task Board.  We used a spreadsheet since our team is working remotely. The team divides the work, and each developer will move it to the **In Process** column when starting a task.

# Project Planning

According to Humphrey, project plan defines the work and how it will be done (1989). For our project plan, we incorporated Scrum practices to define goals and objectives, establish a Work Breakdown Structure (WBS), create project measurements, review productivity measures, create estimations on project completion, and track our overall progress. The project plan allows us to review if tasks were completed in a timely manner, providing the team with a framework for managing the project.

## Goals and Objectives

The goals and objectives of the product were established during the pre-game phase. The goal was to build an Android Stack application and to use agile practices incorporating both Scrum and XP practices. For our WBS, we began by defining our overall process. We then defined our process into further details by defining our work products, practices, and roles. Afterwards, we gathered our requirements by creating user stories and then built a conceptual design as shown in Appendix A.  From there, we then planned our iterations by creating a task board with estimated hours to complete each task. We had three iterations that were one week long each. By estimating the number of hours needed to complete each task, we were able to develop a schedule for our project. We created a calendar to show which dates were specified for iteration planning, iterations, demos, and sprint reviews. Initial requirements were vague in the beginning as were our estimations. Our project plan, requirements, size estimates, and schedules were refined continued as the project moved along at the end of each phase review, or after each iteration in our process. To track our progress, we used a burndown chart. Our phase review was conducted in a sprint review. This refinement results in an overall clear, detailed design and implementation strategy (Jo, 2015).

## Estimations

By estimating the hours to complete each task, we had a way to our time that was needed to complete the product. No Line-of-Code (LOC) measurement was defined early in the project. However, since we had versioning control, we are able to analyze LOC after the project and see our progression. We measured LOC by looking at the MainActivity. Our product had 138 amount of lines in version 1, 194 amount of lines in version 1.2, and 232 amount of lines in the final release. We also did not use function points for initial estimations. Only after completing the project, our function points had five inputs, one output, one calculations, and zero databases. If we were to develop another product similar in size, we can use these measurements to estimate time, cost, resources, and schedule.

## Productivity Levels

Since our software maturity level is low, we do not have significant data to analyze productivity factors. Our productivity can only be measured by the number of hours estimated versus number of actual hours spent on coding tasks. This can be viewed by analyzing our task board and burndown chart. The task board listed each task per user story with estimated hours needed to complete the task. These estimates were made by the programmer’s experience. We can also analyze the amount of hours spent completing these tasks in comparison to LOC. This can be converted into number of programmer months and time required. Again, if we were to have a project in similar size, we can use these measurements to estimate the costs, resources, and time required.

# Project Monitoring and Control

## Project Monitoring

Project monitoring and control occurred at two levels of the project. At a high level, the project was monitored for the overall product. At a more detailed level, the project was monitored at each sprint.

At a high level, primarily the Scrum Master and Product Owner monitored the product. During the initial planning phase, the Scrum Master met with Product Owner to establish a vision and scope. The Product Backlog was defined and estimations were given to each story. A timeline was created which outlined time-boxed iterations. The Product Backlog was prioritized, reviewed, and updated prior to the start of the next iteration. Tentative release dates were derived based on priorities and timeline.

At the sprint level, the project was monitored in more detail and more frequently. All work to be completed for the duration of the iteration was established and estimated at the beginning of the iteration. No additional work was added in the middle of an iteration. A daily standup meeting was scheduled for the same time each day. During this meeting we reviewed what tasks we had worked on, what task we would work on next, and we identified any impediments. During the daily scrum meeting, tasks were updated with status and remaining development hours. Any impediments that were identified during the scrum meeting were addressed immediately after the meeting with the goal to remove the barrier within that day.

A Sprint Burndown Chart was used as a visual guide to see if the sprint was on target to meet the sprint goals. The burndown chart depicts the estimated development hours remaining over time. As the tasks were updated, the sprint burndown chart reflected the updated values. The burndown chart had two lines. The first line was the target velocity. This was calculated by dividing the number of estimated hours by the number of days. This gave us the number of target hours her day in order to develop at a steady sustainable pace. If the actual burndown was above the target then we knew that there was a higher probability of not meeting the deadline.

## Project Control

Control at the project level was monitored during the sprint reviews. As the product owner added new stories to the Product Backlog, there was consideration on the overall scope and vision for the product. Items that were added that fell beyond the vision and scope were given a lower priority and were subject to not being completed if the project exhausted its allocated time and resources.

At the sprint level control was monitored by watching for an increase in scope based on ambiguous acceptance criteria. If a story had an original estimate of 5 hours and the development team came across a requirement or acceptance criteria that was not clear they look for clarification and were cautious that the new criteria did not go too far beyond the original scope of the story. Had the scope increased then a decision would have had to be made as to continue with the story with increased scope. If kept, then another story would have been removed.

# Configuration Management

Configuration management is a critical part to the development of ‘StackApp’ as it allowed our team to effectively minimize time-consuming development issues. This allowed our team to make changes to the software quickly and without issue, as we were able to formulate a baseline for each iteration, revise and update the application, and quickly test the created portion of the main system with ease.

## Change Management

In order to manage changes to the software, our team used two different methods, pair programming and repositories (GitHub). With pair programming, our team was able to have two or more individuals working on a certain part of the code at the same time. This allowed for the use of multiple individuals with multiple levels of experience to use their variety in expertise in the development process. With this, there was a smaller margin for error in development because the greater quantity of individual inputs removed the small mistakes that would later be caught in quality assurance, causing a delay in release.

The second change management tool our team used was pushing developed code to an online repository. For this project we used GitHub, a widely used software repository website. A repository shared with the entire team allowed for everyone to have access to the source code and thus everyone had the ability to either commit their own changes to the software or allowed us to divide multiple, customer demanded updates between different members of the group easily. Of course there are complications to this system. For instance, if someone was to not update their version of the code to the most recent uploaded version, they could be developing source code that is no longer correct, bugged, or missing crucial elements. This issue however, is resolved with effective management involvement and active daily Scrum.

## Versioning

Our team had two versions of ‘StackApp’, the initial release with the basic functions demanded by the customer. This included a way to exit the program, a text-box for user inputs, pop, push and clear buttons, and an area for outputted text from the program. After this however, the customer desired a larger volume of features in the application. This came in the form of an about/help area to provide the user with information about the app and a user-guide which will resolve any use questions a user may have. The customer demanded a keyboard within the application which would allow them to input numbers without the need for the built-in Android keyboard.

The method of versioning we used to solve this issue was with an evolutionary release approach. Instead of taking an entire part of the code and changing it or re-working it completely, we took what we already had, for which the customer already liked, and molded it into a greater product for the 1.2 release. The software thus, evolved into a new, improved version of itself which gave the user more features and improving their overall experience with the application.

## Requirement Changes

Another form of configuration management our team used was our upkeep with requirements and ensuring that our requirements for the application evolved alongside the application itself. As discussed earlier in this section, the customer required more of the application later on after the initial release then what was originally discussed. This made our team divide the application into versions in order to update the product effectively.

## Testing

The last of the configuration management our team used would be with the establishment of a testing practice. The next section will delve deeper into this segment, but testing was used as a tool for configuration management. We established unit tests as well as functional testing within the application, using the JUnit testing framework. For our unit testing, we developed methods that replicated the application and programmatically simulated user inputs and stack creation as well as stack destruction. The unit test then would make assertions based on the expected output of the program and if the output followed the assertion, the test would pass. As an example, if the second element in a stack was expected to be the integer “2” then the unit test would assert that assumption. If an integer “3” was inserted into that place in the stack however, the test would fail the assumption.

Along with unit testing we also had some functional testing. Our team developed tests, which would replicate GUI selections rather than programmatically making selections. This allowed us to completely replicate user actions and then create tests based on all possible user actions for the application. Not only now were we able to ensure basic stack functioning was working in the application, giving the user the proper output, but also verify the GUI’s functions provide the user with the desired response.

This phase allowed our team the chance to inquire about possible issues with the system for the customer but also displayed to us the errors we overlooked in development. Along with the solidifying of the software, the unit tests and functional tests are set up in a way that would allow for easy updating of our testing protocol. This means that as we have future versions of the software, which will inevitably mean changes with existing methods, changing tests to fit the new versions of the application is a swift task, thus ensuring quick releases.

# Quality Assurance

## The Role of SQA

Quality Assurance is an important part of the development process as it provides the ability for the development team to truly see the mastering of their craft while also ensuring a stable, long-lasting product. SQA has two main roles, Software Quality and User Satisfaction, which solidifies the assurance of a product that meets the desired standards of the consumer (Jo, n.d.-d). Software Quality is important for our product because our customer had specific requirements for which the application had to follow. These strict design requirements require an even stricter testing procedure to ensure a quality product. To ensure this software quality, we turned to unit testing which would allow us to set up all possible desired outcomes and fully test the limits of our application. With unit testing, we can create test cases, about which will specifically target the parts of the application for which the customer is the most concerned. Once this is achieved, the requirements set by the customer have been fully tested, and it is certain that the customer will have no failures with the application, next to be looked at is User Satisfaction.

The user satisfaction as defined consists of: a compliant product, good quality, and delivery within budget and schedule (Jo, n.d.-d). With good management and the effectiveness of Scrum or XP (depending on the team) development, an on time delivery with solid budgeting should not be an issue. So for a satisfied consumer testing becomes that much more important as the QA team must ensure the product complies with the customer and is of good quality. For our program, to do this we tested every desired functionality thoroughly with unit tests but instead of stopping at desired functionality, we shifted focus to the undesired functionality, by the means of running unit tests that intentionally fail the software and ensure that the program will capture these failures and handle them the correct way.  For example, the customer required a base stack of three and the program should run an error if the user attempts to push a fourth input to the stack. Therefore, in our unit tests, we tested the ability to push three elements to the stack, but then set up an assertion that would push a fourth element but expect a failed output. If the test passes, the program handled the inability of a fourth element effectively, though if the test had failed then the element was allowed to pushed onto the stack, thus going outside of the user’s desires. Without this type of testing, we cannot assure the application is of good quality or satisfies the customer’s demands.

## The SQA Plan

Our team formulated a small SQAP, which laid out the groundwork for our testing portion of development to begin writing test cases for the application. Essentially, the team would create a method in the code, as demanded by the user, and we would discuss all of the functioning of this method. Once all of the functionality is discusses, our team would then delve into all of the possible ways for which this method could fail. Then the tester would create the proper method to test all of the elements discussed beforehand.

## Independent Verification and Validation

Part of having a successful QA team is to have a team that is independent from the developers. To ensure that there is no missing details, the QA team must be able to have the authority to push broken code back to development. This means that for a functional development environment, testers should not be intertwined with the developers aside from sending broken code and suggestions for fixing issues with the product.

For our team however, this was not possible as we utilized pair programming and everyone had some input into the development process. Therefore, to have our own type of proper QA outside of this scope, we decided since we developed together with pair programming, we would QA together with pair programming. This decision was made off the basis that since the entirety of our project was developed together, our testing should be as an entire group. This, as with pair programming development, allowed the team to contribute a variety of different test cases to the testing process thus giving us a very full testing experience, which might not have happened if we decided to split the development of this specific application off into traditional development and QA separation.

# Risk Management

According to Khrisnan (2015), risk management or risk mitigation is the “practice of systematically deciding cost effective approaches for minimizing the outcome of threat and maximizes the opportunities in the future” (p.305). Simply, risk management identifies areas of risk or uncertainty in the software process and offers suggestions to mitigate these risks. Risk management is crucial as risks may endanger the completion of a project and may overall lead to waste of time, money, and resources. For risk management, we identified risks for the project into two groups: management and technical risks.

## Management Risks

Management or planning risks pertain to inadequacies in the non-technical areas of the software areas (Mohd-Rahim, Wang, Boussabaine, Abdul-Rahman, & Wood, 2014). For our process, we identified schedule conflicts, schedule estimation, experience with implementing a software process as management risk factors.

### Scheduling Conflicts

Scheduling conflicts risks are uncertainties in which team members cannot fully commit or participate. Scheduling was critical between our team members. Each member has a full-time job limiting our available times to meet. In addition, our team members are not geographically located near each other. Our team members are from different states, and the team needed to address the time-zone differences. These risk factors required that all team meetings had to be completed remotely. To combat these risks, the team agreed to meet at least once per week and schedule other meetings if necessary in advance. We also used Google Hangouts as a means to communicate with each other. This tool was effective as it was installed on all of our mobile devices, and team members could contact each other easily.

### Estimation Inaccuracies

Estimation inaccuracies are risks as it can result in poor software estimates, poor schedule estimates, and poor staff estimates (Humphrey, 1989). Estimations pertain to product size estimates and resource estimates. Because early estimations are often unacceptable, contingencies are required. The team should have contingencies in place in the event estimations are inaccurate. For example, the team should have a contingency in the event tasks cannot be completed on time or certain tasks have become aware that they are too costly. In iteration planning, we gave ourselves more than enough time to complete tasks.

### Inexperience with Implementing Software Process and Agile

Another area of risk was the inexperience with software process. Team members did not have significant experience practicing agile processes. All team members had to learn about the agile process, Scrum, and XP. The team members needed to learn how to implement those practices. For agile processes, high-risk areas include workflow, customer experience, and project size (Khrisnan, 2015). There was a substantial learning curve to ensure workflow was correctly completed. Our team needed to ensure the process reflected the core agile principles. The customer experience is a high-risk area because the emphasis of the agile process is customer satisfaction. Project size is a high-risk area in agile processes because of its inherent ideals of implementing an adaptive and evolutionary process and being flexible to change. A general risk from using an agile process is the lack of documentation, which may lead to implications of scale.

## Technical Risks

Technical risks involve any technology related uncertainties to the development of the product. For our process, technical risks involved the environment inexperience and poor requirements.

### Inexperience with Android Platform and GitHub

A technical risk was the inexperience with the Android Platform and using GitHub as a source control. Android requires knowledge of Java and XML. In addition, team members had no experience in using GitHub. This inexperience is a risk to the project as it may slow or interrupt the velocity of the project. To mitigate these risks, our team had to complete training tutorials.

### Poor requirements

Poor requirements is a technical risk as it directly affects developing the product itself. Some examples of poor requirements are ambiguous requirements and creeping user requirements. Ambiguous requirements lead to different expectations and wastes time as developers try to implement solutions for the wrong problem (Wiegers & Beatty, 2013). Creeping user requirements are requirements that evolve during development and are out of scope (Wiegers & Beatty, 2013). To mitigate these risks, our team must keep the project within scope limitations and have all stakeholders review the requirements.

# Lessons Learned

## Lessons Learned - Anthony Farina

In my short career as a software engineer, I have not encountered a formal software process. Before this homework, I have never heard of agile processes, let alone Scrum or XP. This homework has taught me what a process is and how to be a part of a process. I learned that a defined software development process is a very important part of developing a software product. It allows the team to work more efficiently by catching inconsistencies and keeping the product/development in scope, which can save a lot of time and resources. What I would like to learn more about is how a very small team can use a defined process without being slowed down. It seemed to me that a defined process could slow down development but I am guessing that this is a necessary drawback.

## Lessons Learned - Lourdes Lopez

I had some prior experience with agile process, but it was only partial knowledge of the process. I knew of bits and pieces of the process in which I participated in as a developer, without really having an overall understanding of the process, the various roles and activities that went beyond the developer involvement. What I really enjoyed about this project was being able to delve into the different agile processes such as Scrum and XP and exploring the differences and similarities. By customizing both Scrum and XP to fit our team, it was required to have an understanding of the benefits of each activity or work product that we selected. One thing that I would like to learn more about is what are the best practices for a larger project in which there are multiple teams working on the same project simultaneously. What work products and processes are shared vs ones that are conducted by each team independently? For example, do they share the same code base or work independently and then merge at the end of the iterations.

## Lessons Learned - Joanna Hang

I had no prior experience with a formal software process. From this homework, I learned the importance of a software process and types of processes used in software engineering, specifically the agile process. I learned how to define a process and how to implement a process. I learned the differences between Scrum and XP, and I learned how to combine both processes. In the future, I would like to learn how to implement a process with a product that is more difficult in scale. With a relative ease in the product application, I found it difficult to create our process within short time frame for a small application.

## Lessons Learned - David Sullivan

Since I was not familiar with the details of the agile process before this assignment, I learned a lot by working on this project.  Working on an actual development project using the Scrum and XP processes was a great learning experience that I do not think I would have an opportunity to learn in my organization. By taking different roles in the process, it helped me to see the development from different perspectives. It was useful to start with a process and build a plan to follow that helps you to clear up any misunderstandings in the requirements and work together with everyone when there are more clearly defined goals.  While this project was relatively simple, we could easily coordinate on the project even without all having worked together before. I could see how following the agile process could increase the product value and speed up development time. Now I am curious how the result would be if I tried to follow agile process on one of my projects for work.

## Lessons Learned - Timothy Cioffi

Since I was not familiar with the details of the agile process before this assignment, I learned a lot by working on this project.  Working on an actual development project using the Scrum and XP processes was a great learning experience because I have not had an opportunity to try using a software development process my organization. By taking different roles in the process, it helped me to see the development from different perspectives. It was useful to start with a process and build a plan to follow that helps you to clear up any misunderstandings in the requirements and work together with everyone when there are more clearly defined goals.  While this project was relatively simple, we could easily coordinate on the project even without all having worked together before. The pair programming was a good way to get up to speed on the Android platform. I could see how following the agile process could increase the product value and speed up development time. Now I am curious how the result would be if I tried to follow agile process on one of my projects for work.

## Group Experience

We learned about the agile process, and we learned how to use the Scrum and XP methodologies. Specifically, we used Scrum for team and project management, and we used XP for development. Together, we self-managed our team where individual team members volunteered for work. We learned how to handle changes in scope as we added more features in each iteration. We learned to do pair programming, which helped the non-Android developers learn from the experienced Android developers. This also reduced the number of bugs during programming. Overall, this project facilitated and improved communication between our team members.

# References

Android. (n.d.). *Android developers*. Retrieved from http://developer.android.com/index.html

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). *Manifesto for agile software development.* Retrieved from <http://www.agilemanifesto.org/>

Gassner, D. (2015, April 17). *Developing android apps essential training* [Lynda.com online course]. Retrieved from <http://www.lynda.com/Android-tutorials/Developing-Android-Apps-Essential-Training-Revision-Q2-2015/369905-2.html>

Gassner, D. (2015, July 24). *Java essential training* [Lynda.com online course]. Retrieved from <http://www.lynda.com/Java-tutorials/Java-Essential-Training/377484-2.html>

Humphrey, W. (1989). *Managing the software process*. Reading, Mass.: Addison-Wesley.

International Scrum Institute. (n.d.). *Release planning*. Retrieved from <http://www.scrum-institute.org/Release_Planning.php>

Jo, C. (n.d.-a). *Advanced software process part I: Software process maturity 3. software process assessment* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Jo, C. (n.d.-b). *Advanced software process part II: The repeatable process 6. the project plan* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Jo, C. (n.d.-c). *Advanced software process part II: The repeatable process 7. software configuration management (Part I)* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Jo, C. (n.d.-d). *Advanced software process part II: The repeatable process 8. software quality assurance* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Jo, C. (n.d.-e). *Agile process & xp.* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Jo, C. (n.d.-f). *Scrum.* [PDF document]. Retrieved from CPSC 544 Advanced Software Process Web site: <https://moodle-2015-2016.fullerton.edu/course/view.php?id=8752>

Krishnan, M. (2015). Software development risk aspects and success frequency on spiral and agile model. *International Journal of Innovative Research in Computer and Communication Engineering, 3*(1), 301-310.

Mar, K. (2015, April 18). *[Scrum 101] Scrum and xp (extreme programming)* [Video file]. Retrieved from <https://www.youtube.com/watch?v=HwOMtectTkQ>

Marini, J. (2014, February 27). *XML essential training* [Lynda.com online course]. Retrieved from <http://www.lynda.com/XML-tutorials/XML-Essential-Training/145930-2.html>

Mohd-Rahim, F., Wang, C., Boussabaine, H., Abdul-Rahman, H., & Wood, L. (2014). Factor reduction and clustering for operational risk in software development. *Journal of Operational Risk, 9*(3), 53-88.

Scrum alliance. (n.d.) *Learn about scrum.* Retrieved from <https://www.scrumalliance.org/why-scrum>

Wells, D. (1999). *The rules of extreme programming.* Retrieved from <http://www.extremeprogramming.org/rules.html>

Wiegers, K., & Beatty, J. (2013). *Software requirements.* Redmond, WA: Microsoft Press.

# Team Charter

|  |  |  |
| --- | --- | --- |
| **Course Title** | CPSC 544 (50) Advanced Software Process | *All team members participated in the creation of this charter and agree with its content.* ***Date*** *08/31/2015* |
| **Instructor** | Dr. Chang-Hyun Jo |
| **Course Dates** | 08/24/2015 – 12/11/2015 |

**Team Members** (Contact Information)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Address (city, state, country) | Phone | Cell | Email |
| Timothy Cioffi-Dinkel | San Diego, CA | 951-775-6150 | 951-775-6150 | timothycioffi@csu.fullerton.edu |
| Anthony Farina | Bear, DE | 302-299-5495 | 302-299-5495 | afarina@csu.fullerton.edu |
| Joanna Hang | Fullerton, CA | 714-686-7401 | 714-686-7401 | johang@fullerton.edu |
| Lourdes Lopez | Turlock, CA | 209-205-0581 | 209-205-0581 | lourdeslopez@csu.fullerton.edu |
| David Sullivan | Fullerton, CA | 714-686-9488 | 714-686-9488 | dsullivan@fullerton.edu |

**Team Member Skill Inventory** (Areas individual members can contribute)

|  |  |
| --- | --- |
| Timothy Cioffi-Dinkel | * MySQL * Java/C++ * XML, HTML/CSS, PHP * Photoshop * Game Development, Unity/C# |
| Anthony Farina | * MySQL - Database administration and design * Java * Android Development * Microsoft Office * Google Web Toolkit |
| Joanna Hang | * SQL Server - Database administration and design * SQL Reporting Services * HTML, CSS, JavaScript, jQuery, Bootstrap, and Less * Microsoft Office * Adobe Photoshop, InDesign, Digital Publishing Suite |
| Lourdes Lopez | * Data Analytics - Complex queries using T-SQL or PL-SQL * Business Intelligence tools - Business Objects Web Intelligence, Business Objects Universe Design, Microsoft SSRS and Crystal Reports * Web Application Development with C# ASP.NET and JavaScript * Experience with agile development using Scrum methodologies * Microsoft Visio for documenting business process |
| David Sullivan | * iOS Development * Android Development * Java * C#, ASP.NET, JavaScript, Bootstrap * Microsoft Web API |

**Team Goals** (Project goals, team process goals, quality goals, etc.)

|  |
| --- |
| * Combine the unique skill sets of each of our team members, so that we can produce a successful project as well as learn from each other’s areas of expertise. * Learn about the software process by reviewing course material as well as experiencing the process as we work towards our course assignments. * Improve upon our existing knowledge of software development processes so that we will develop software in a more organized manner. * Keep track of our weekly assignments and goals so that we meet our deadlines and are well prepared for upcoming assignments. * Establish and maintain strong communication between teammates in order to facilitate success. |

**Team Roles** (Define roles of each member to achieve goals)

|  |  |
| --- | --- |
| Lourdes Lopez  Team Lead | * Act as the team liaison which includes communicating with the professor and/or class regarding the team’s progress and/or asking questions * Turn in assignments, take a screenshot of submitted assignment, and e-mail team members * Collaborate with Facilitator on agendas * Manage project timeline |
| Joanna Hang  Recorder | * Take meeting notes and put them in Google Drive * Document ideas and key decisions from meetings * Ensure team members know the purpose and goals for assignments |
| Timothy Cioffi-Dinkel  Facilitator | * Inform team of any changes to meetings * Assist team lead with agenda before meetings * Coordinate video calls/start video calls * Begin/end conference calls with a check-in with each team member * Ensure all team members participate and ask everyone for input. * Cover topics on the agenda * Manage meeting times |
| David Sullivan  System Architect | * Ensure the project is within the scope of technical requirements * Evaluate functional and non-functional requirements * Review project through a technical standpoint |
| Anthony Farina  Quality Assurance | * Ensure process meets users’ quality expectations * Extract collaborative assignments from Google Drive and convert documents to appropriate document type (Word, Excel, PDF, etc.) * Perform final review by checking format and reviewing assignments * Upload final copy to Google Drive and inform Team Lead that assignment is ready to turn in |

**Ground Rules**

|  |
| --- |
| * All team members will check Titanium daily. * All team members will check emails at least once per day and reply within 24 hours. * All team members will meet on Tuesday’s at 5:15 PM PST using Google Hangouts. * Google Hangouts will be the main tool for communication between team members. * Google Hangouts will be installed on each member’s mobile phone. * All team members will check Google Hangouts at least twice per day. * All team member will complete collaborative assignments using Google Drive. * All team members must be respectful, honest, and clear to all other team members at all times. * No plagiarism or cheating. Each team member will proof read each other’s work to prevent this. * Group will rotate team leaders each semester. * All team members will communicate and ask questions. * All team will equally contribute, show effort, and will not leave the work to a few individuals. * All team members will be supportive, willing to help and assist other team members. * Each member must be open to constructive criticism from each other team member. * Each member is responsible for their tasks. Tasks must be completed by the deadlines set by the professor and or the group. * If emergency arises that may affect deadlines, meetings, etc., team member will at least inform the team leader or if possible the entire group via Google Hangout about the situation so the group can plan accordingly. |

**Time Commitments/Availability** (Pacific Time)

|  |  |
| --- | --- |
| Timothy Cioffi-Dinkel | * M-F 5pm-8pm * Sat-Sun all day |
| Anthony Farina | * M-F 5pm-8pm * Sat-Sun all day |
| Joanna Hang | * M-F 5pm-8pm * Sat-Sun all day |
| Lourdes Lopez | * M-F 5pm - 8pm * Sat-Sun all day |
| David Sullivan | * M-F 5pm - 8pm * Sat-Sun all day |

**Conflict Management**

|  |
| --- |
| * In order to avoid conflict among team members, roles and responsibilities should be equally designated. * If conflicts arise, concerns should be submitted to the agenda and will be addressed to the whole team to settle conflicts. |

**Risk Management**

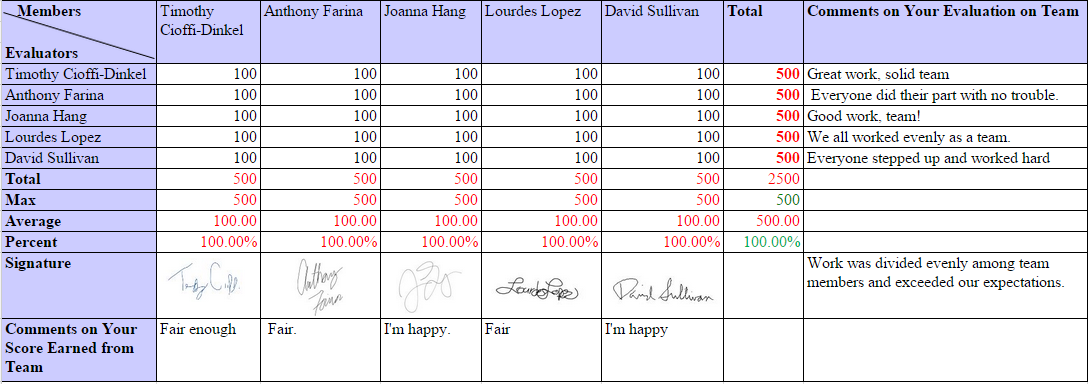
|  |
| --- |
| * Identify risk factors within team (meeting time conflicts , external commitments to work and family) and within the software process (training, organizational policies, configuration) * Prioritize tasks in order of importance and communicate to stakeholders * When problems arise, communicate with team and stakeholders * Actively monitor progress and reevaluate requirements |

**Team Evaluation Criteria**

|  |
| --- |
| * Evaluate each individual team member on their own work. * Evaluate each individual team member on their participation in group discussions. * Evaluate each individual team member on their ability to collaborate on the group project. * Evaluate each individual team member on their ability to produce material in a timely matter. * Evaluate each individual team member on their ability to fulfill role. |

# 

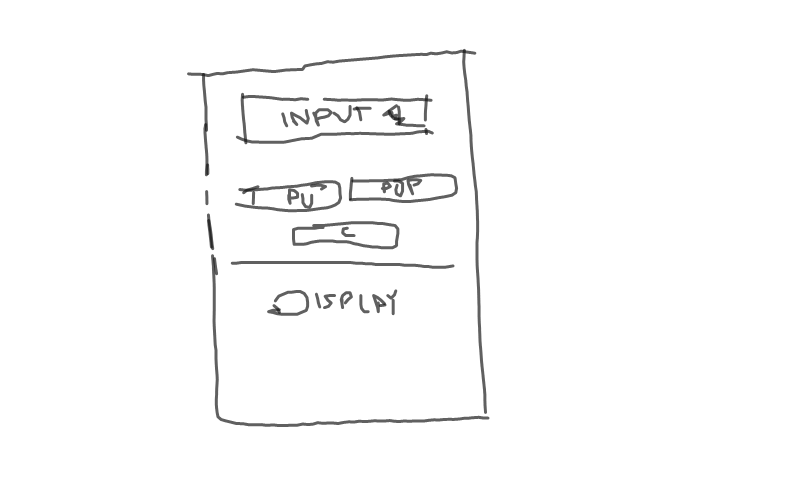
# Team Evaluation



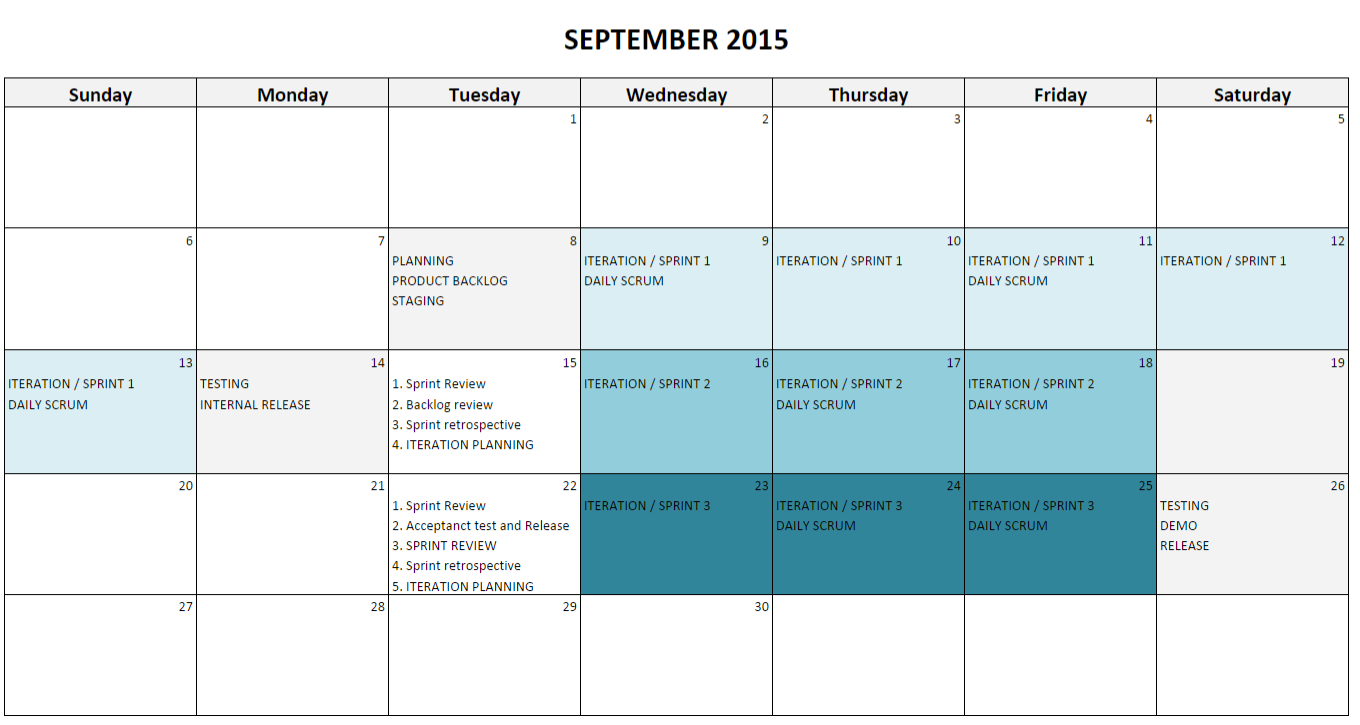
# 

# Appendix A - Sketches

A.1 - First Preliminary Sketch



# Appendix B - Timeline

****

# Appendix C - List of Tables and Figures

**Tables**

[Table 1 Caliware Software Process Model 5](#_Toc431231945)

[Table 2 Operation and Stack 16](#_Toc431231946)

[Table 3 User Stories 20](#_Toc431231947)

[Table 4 Sprint I Product Backlog 21](#_Toc431231948)

[Table 5 Sprint II Product Backlog 21](#_Toc431231949)

[Table 6 Sprint III Product Backlog 22](#_Toc431231950)

[Table 7 Product Backlog 24](#_Toc431231951)

[Table 8 Iteration I Sprint Backlog 28](#_Toc431231952)

[Table 9 Iteration II Sprint Backlog 34](#_Toc431231953)

[Table 10 Iteration III Sprint Backlog 37](#_Toc431231954)

**Figures**

[Figure 1 System Context Diagram 15](#_Toc431231930)

[Figure 2 Iteration I Burndown Chart 30](#_Toc431231931)

[Figure 3 Google Hangout Pair Programming 31](#_Toc431231932)

[Figure 4 Main Screen 32](#_Toc431231933)

[Figure 5 Pushed Value to Stack 32](#_Toc431231934)

[Figure 6 Max Stack 32](#_Toc431231935)

[Figure 7 Iteration II Burndown Chart 35](#_Toc431231936)

[Figure 8 Iteration III Burndown Chart 38](#_Toc431231937)

[Figure 9 Training - Open StackApp Icon 41](#_Toc431231938)

[Figure 10 Main Screen 42](#_Toc431231939)

[Figure 11 Main Screen with Numerical Pad 42](#_Toc431231940)

[Figure 12 How to Push 43](#_Toc431231941)

[Figure 13 How to Pop 44](#_Toc431231942)

[Figure 14 How to Clear 44](#_Toc431231943)

[Figure 15 How to Change Stack Size 45](#_Toc431231944)

1. http://developer.android.com/sdk/installing/installing-adt.html [↑](#footnote-ref-1)