Homework 1

Insertion Sort Application

Alexander Liao
aliao@csu.fullerton.edu

Danny Zhang
danny.zhang@csu.fullerton.edu

Eric Lyv
lyveric@csu.fullerton.edu

John Crisanto
jcrisanto1987@csu.fullerton.edu

John Mai
jmai1011@csu.fullerton.edu

Kevin Tran
kevintt@csu.fullerton.edu

Team OC (Group 3)

9/28/2016

CPSC 544 – Advanced *Software* Process
Dr. Chang-Hyun Jo
Department of Computer Science
California State University, Fullerton

Revision History

Date	Performed By	Actions
8/27/16	Entire Team	Team charter, title page
8/27/16	Entire Team	Documentation/report template, process definition
8/28/16	Entire Team	Scrum, Extreme Programming, and preliminary project timeline
8/29/16	Entire Team	Work-products, roles, practices
8/30/16	Entire Team	Vision and goals of the project, process definition, work-products, roles, practices
8/31/16	Entire Team	Functional requirements, non-functional requirements, influential factors
9/2/16	Entire Team	User stories, product backlog, estimates
9/3/16	Entire Team	Architectural spikes, release plan, technology preparation
9/6/16	Alex Liao, Danny Zhang	Iteration I sprint plan, sprint backlog, design
9/7/16	John Mai, Kevin Tran	Iteration I coding
9/8/16	Eric Lyv, John Crisanto	Iteration I testing, small releases, sprint review
9/12/16	Eric Lyv, John Mai	Iteration II sprint plan, sprint backlog, design
9/13/16	Danny Zhang, John Crisanto	Iteration II coding
9/14/16	Alex Liao, Kevin Tran	Iteration II testing, small releases, sprint review
9/19/16	John Crisanto, Kevin Tran	Iteration III sprint plan, sprint backlog, design
9/20/16	Alex Liao, Eric Lyv	Iteration III coding
9/21/16	Danny Zhang, John Mai	Iteration III testing, small releases, sprint review
9/23/16	John Crisanto, John Mai	Release documentation, training
9/23/16	Alex Liao, Danny Zhang, Eric Lyv, Kevin Tran	Operating, feedback
9/25/16	Entire Team	Lessons learned
9/25/16	Entire Team	Team evaluation

Table of Contents

Table of Contents

1.0 Process Definition

Software engineering is similar to other engineering activities, as it requires careful and detailed planning and management. In order to perform better planning, established software development processes are followed. In this project, we will focus on the implementation of two Agile processes that work well in tandem: Scrum and Extreme Programming (XP).

First, our Agile process allows us to implement an iterative development approach to building software products. This means that our process will be divided into multiple timeboxed (fixed time) iterations, with each iteration being treated as its own separate project with the goal of having a stable, integrated, and tested system at the end of each iteration. Our Agile process will be risk-driven, meaning that the more crucial and challenging elements will be assigned in the earlier iterations. Simultaneously, emphasis will also be placed on involving the customers throughout the development of the product. This approach leads to higher customer satisfaction and higher probability of developing a software product that coincides with the customers' expectations and requirements [3]. This process will also be much more flexible and adaptive compared to traditional approaches, where product development relies heavily on one release. A single development release would most likely result in limited customer involvement and satisfaction (e.g. Waterfall approach). In the following sections, the implementation of the Scrum and XP Agile process will be described in detail.

1.1 Scrum

Scrum is an Agile management process that focuses on building software that meets business needs. There are four phases in the Scrum lifecycle: *planning*, *staging*, *development*, and *release*. Our process will follow this structure of the lifecycle. Figure 1 provides an overview of the different phases that are involved.

Figure 1: The four phases in the Scrum lifecycle

As mentioned, our Agile process involves building a product in iterations. In the Scrum process, these iterations are referred to as “sprints”, which typically last 30 calendar days. As the Scrum team progresses through the sprint, typical artifacts are used to help keep track of progress. These include the product backlog, sprint backlog, burndown charts, and potentially shippable products. Ideally, a Scrum team will complete all items on their sprint backlog for every sprint; however, this may not always be the case. For good practice, no additional work may be added to a sprint once it is established. Additionally, 15-20 minute daily stand-up meetings will be held to keep the team in sync. In the Scrum process, the goal of the team is to deliver a coded, tested, and usable piece of software at the end of each sprint. Since client-driven adaptive planning is used during this process, a sprint review meeting will also be held around this time. Common attendees to this meeting include the Scrum team, the Scrum Master, product owner, customers, and management representatives. Scrum team accomplishments for this sprint will also be presented at this meeting.

A Scrum-based process is flexible and adaptive to changes and encourages the avoidance of a prescriptive process. Since our process is predominantly Scrum-based, we will mainly be utilizing these three key roles: *Scrum Master*, *Product Owners*, and *Scrum Team*.

- **Scrum Master**: the Scrum Master is the coach of the team. He/she helps remind the team of the product vision. When the team faces any external factors that may interfere with development, the Scrum Master is there to resolve those issues. It is the Scrum Master’s responsibility to acquire any resources that the team needs in order to complete their tasks.

- **Product Owner**: the Product owner represents the customers where he/she will be in charge of prioritizing the features that need to be developed during that iteration. He/she must collaborate with the team to ensure that the product will be successful. Once an iteration is established, the Product Owner must be available for questions and feedback throughout the iteration.

- **Scrum Team**: the Scrum Team is self-organized with a maximum of seven members. The members collaborate and meet daily in order to discuss any issues or blockers that they encountered while developing the product.

1.2 Extreme Programming (XP)

Extreme Programming (XP) is an Agile engineering process that focuses on customer satisfaction and empowers the developers to respond to constant changes in the software requirements. It promotes a collaborative environment by involving managers, developers, and customers throughout the software development lifecycle [1]. Our process follows XP practices such as test-driven development, automated testing, pair programming, simple design, refactoring, and so forth [2]. Though our process is a blend of Scrum and XP practices, we will enforce the following simple rules that XP uses for the following categories: *planning*, *managing*, *designing*, *coding*, and *testing*.

- **Planning**: in this phase, customers write user stories for the software product that they want to be built. The purpose of the user stories is to provide time estimates for the release planning meetings, where the team discusses the project timeline and how many iterations it will take to complete all user stories.

- **Managing**: it is important for the team to have open communication throughout the development process. As mentioned, there will be daily stand-up meetings that will help the team identify any blockers during the development phase. Project velocity charts will be used by management in order to measure the amount of work completed and provide a better time estimate regarding delivery of the product.

- **Designing**: simplicity is the model of XP; the design of the code needs to be testable, understandable, browsable, and explainable (TUBE).

- **Coding**: it is important to maintain a collaborative coding environment. Customers should always be available in order to provide feedback or answer any questions that the developers may have. It is important to provide quality code, therefore pair programming and other practices will be utilized during the process (see Section 2.3).

- **Testing**: testing is a crucial aspect of XP. There are many types of software testing; however, XP focuses on *Unit Testing* and *User Acceptance* testing.
    - Unit testing involves testing every unit of the system. Unit tests must be written before development and should be done using unit test frameworks. All code must pass unit tests before it can be released.
    - Acceptance testing is the final test, which tests against the user stories to ensure that the system being built meets the requirements.

Although our process will be based on the Scrum project lifecycle, the team will follow these aforementioned rules when building the software product. For reference, Figure 2 displays an overview of the five different phases in the XP project lifecycle [1].

Figure 2: Extreme Programming (XP) project lifecycle

1.3 Project Timeline Overview

There will be 3 iterations for this project. The duration of each iteration in Scrum is usually 30 calendar days. Unfortunately, due to time constraints, we will have 4-day iterations. One day of our iteration will be equivalent to 1 week. As mentioned, we will have daily stand-up meetings that will not exceed 20 minutes. We will have a retrospective and review meeting once a week that will last 1-2 hours. Our goals for each iteration will be as follows:

- **Iteration I**: create the main functions of the Insertion Sort App
    - The initial release of the application with basic user requirements
- **Iteration II**: improve use functionality of the application
    - The second release of the application with enhanced usability and performance
- **Iteration III**: enhance features, satisfy the business value, and finalize the application
    - Include any special features that distinguish our application from our competitors

2.0 Definition of Work-Products, Roles, and Practices

2.1 Work-Products

Although we are adhering to both Scrum and XP practices, we will be producing the typical work-products that are native to a traditional Scrum process. For our purposes, there will be four work-products: *product backlog*, *sprint backlog*, *burndown chart*, *potentially shippable product increment*.

Product Backlog

The product backlog will be composed of a list of features, usually in the form of user stories created by the customers, which are needed to complete the product. Each entry in the product backlog must hold customer value. The product backlog is a living document that goes through change throughout the entire project and will be used by the product owners to communicate with the stakeholders/customers. The product owner will be responsible for prioritizing the tasks within the product backlog.

All of the entries within the product backlog will be assigned an estimate using the concept of "story points". The team will take a reference user story and assign it an arbitrary number. The other user stories will then get story points based on a comparison with the reference user story. This estimation will then be used to assign a priority criteria value of low, medium, or high. The product owner must consider the following factors when prioritizing user stories: *risk*, *value*, *knowledge*, and *cost* [9]. Figure 3 displays an example of a product backlog.

ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

Figure 3: Example of a Scrum product backlog

Sprint Backlog

The sprint backlog is a list of tasks derived from the user stories that will be taken from the product backlog. It will be used by the Scrum team to keep track of the status of the project. The tasks will be created at the beginning of each sprint and will be posted on a wall so that team members can easily access it throughout the sprint. In order to keep track of the tasks for each sprint, each task must have a status and time estimation attached that will be updated frequently. Figure 4 is an example of a sprint backlog [13].

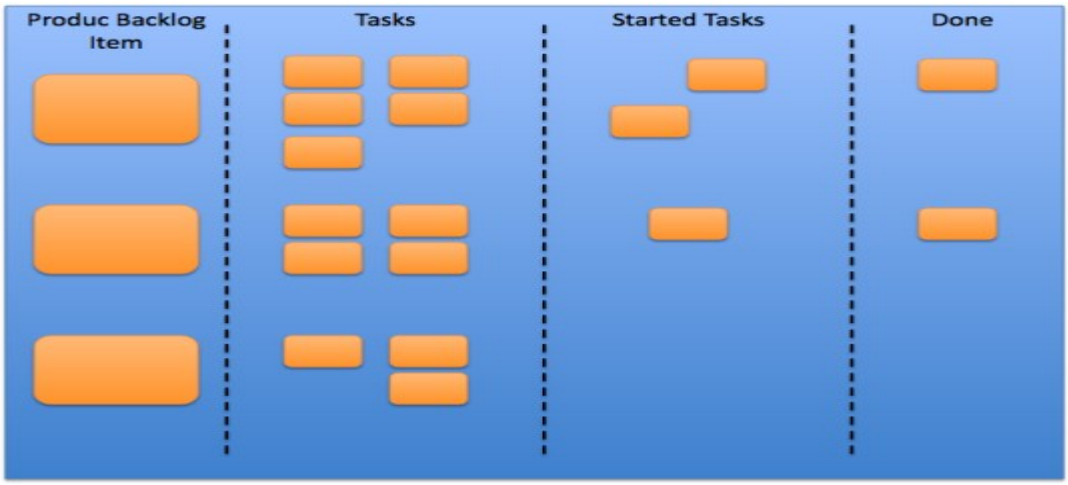


Figure 4: Example of a sprint backlog

Burndown Chart

The burndown chart is a visual measurement tool that is used to track the completed work per day against the project rate of completion for each sprint. The rate of progress is referred to as the "velocity", which represents the amount of story points completed per sprint. Only stories that are fully completed at the end of the sprint are counted. The burndown chart aids in estimating the time needed to complete the entries within the product backlog. Like the other work products, the burndown chart will be updated frequently so that the team has a better gauge as to which stories have not been addressed. Figure 5 is an example of a burndown chart [11].

Potentially Shippable Product Increment

The potentially shippable product increment refers to the software developed by the team by the end of each sprint. As mentioned, the goal is to have a stable, integrated, and tested system at the end of each sprint, which will be demonstrated during the sprint review meeting. These smaller product releases make the process more flexible and adaptive to customer demands. Each release will be documented as its own separate project.

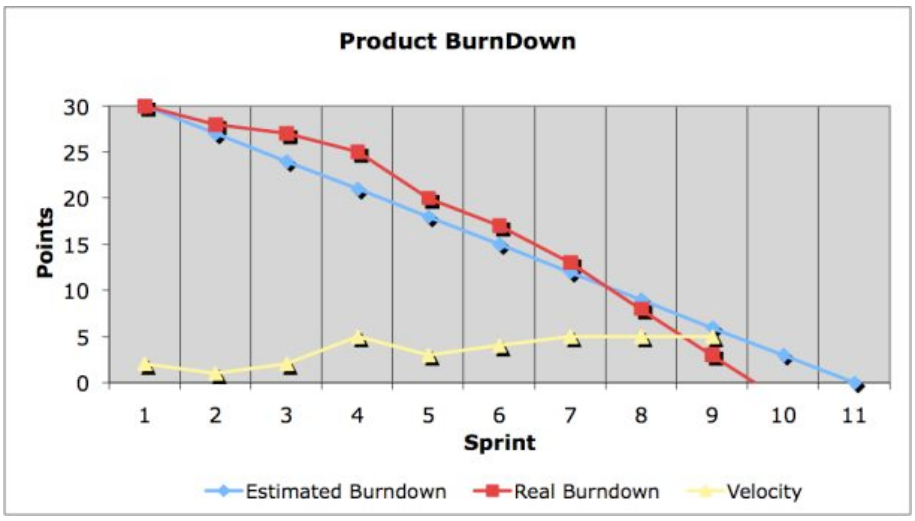


Figure 5 Example of a burndown chart

2.2 Roles

The Scrum process has two distinct categories of individuals, referred to as the chickens and the pigs. The chickens are individuals who have a vested interest or may benefit from the product that is being developed. These individuals are not necessarily held accountable for the product and are not as focused on the product and its customer-oriented goals. However, they may be consulted on the project and are usually informed of its progress. Typically, the chickens are those who have committed their only resources. During Scrum meetings, the chickens are only allowed to observe and may not interrupt the team's process. Examples of chickens include, but are not limited to, stakeholders, consultants, and managers.

In contrast, the pigs are individuals who are committed to the project and are held responsible for building the product. These are the individuals who perform the work to bring the project to its completion. Examples of pigs include, but are not limited to, Scrum Masters, product managers, product owners, and developers.

For the purposes of our project, the key roles that will be involved in building our product are the Scrum Master, product owner, and Scrum team. Figure 6 shows an overview of these roles in a typical Scrum process.

Customer/Product Owner

The role of the customer is very important in an Agile process, especially since the planning for each iteration is client-driven and is based on customer feedback. For our process, the customer will also be the product owner. Since we are implementing a blend of Scrum and XP, the product owner will be responsible for writing user stories and acceptance tests. The product owner will also be responsible for choosing which stories are going to be part of the next sprint, and will create and prioritize the product backlog. As one of the main players in the process, the product owner should be available to answer any questions the development team may have and should set realistic and feasible goals for the team. At

the end of each sprint, the product owner will participate in the demo of the product and will provide feedback to the team regarding the product.

Management

For our process, the management role will be filled by one individual: the Scrum Master. The Scrum Master is an individual that gains fulfillment from facilitating others' success as their own. The Scrum Master should have strong communication and leadership skills since one of their main responsibilities involves resolving conflicts that may arise within the team.

As mentioned, the Scrum Master is the coach of the team. It is their responsibility to lead the daily Scrum meetings to keep the team in sync and tackle any obstacles that may prevent the team from achieving its goals for every sprint. The Scrum Master should be a strong facilitator and be available for support to both the product owner and the team. It is the Scrum Master's duty to help remind the team of the product vision, but it is very important to note that the Scrum Master does not have authority within the team. This means that they do not control other team members, assign tasks, or push for change.

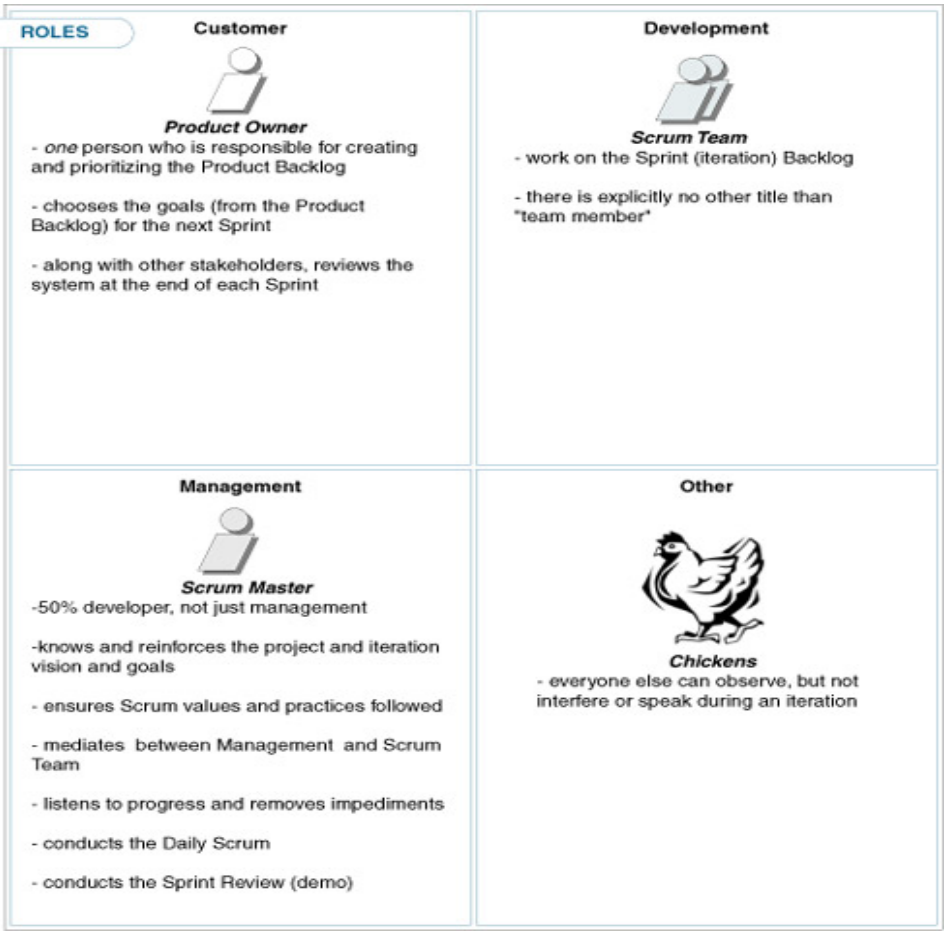


Figure 6: Roles in a Scrum process

Development

For our process, the development role is filled by the Scrum team. The Scrum team is a competent and cross-functional team that consists of team members that complement each other's' skill sets. In a Scrum team, there are no titles and all members of the team are working towards achieving the same sprint goals. The Scrum team has complete control of the amount of work that it takes on for each sprint; however, it is important to note that the product owner makes sure that the team takes on the work based on their priorities. Communication between team members is crucial, especially when deadlines are being enforced. The Scrum team must be cooperative with both the product owner and the Scrum Master because in the end, the team is in charge of building a product that adheres to the product owner's vision. The team must be respectful of the product owner's feedback and must perform the work that conforms to the product owner's needs.

2.3 Practices

For the purposes of this project, we will be following the Scrum lifecycle. However, we will incorporate core practices from both Scrum and XP processes throughout this project lifecycle.

First, we will address the pre-game planning and staging phase. We will act as the stakeholders and write our vision and goals for the product. Next, as part of the Scrum process and on behalf of the stakeholders, we will also take the role of the product owners by specifying and writing the software requirements (both functional and nonfunctional) for the product. These requirements will then be expressed in the form of user stories, which are short, but concise descriptions of functionality from a user standpoint. This is a common XP practice. At this point in the process, the product owners will categorize the user stories (e.g., main functions, feature enhancement, improve user functionality) and prioritize them based on their business value and risk. The product owners will also provide estimates for each story, which will be edited later once the Scrum team commits to the story. The prioritized user stories and their estimates will be recorded in the product backlog.

Before the beginning of each sprint, two meetings will be held. The product owners will meet with the stakeholders to refine and re-prioritize the current product backlog and will choose a set of goals for the following sprint. During the second meeting, the self-organized and self-directed Scrum team (usually seven members) will meet with the product owner and analyze the items in the product backlog. This is when the team will generate a sprint backlog of tasks, which will be prioritized and given estimates in the form of person-hours of effort. The purpose of this is to help the team in estimating the number of sprints it may take to complete all of the tasks generated. The goal of the team is to have a working software product that is stable, integrated, and tested by the end of each sprint. The product's features will be demonstrated to the customer for feedback during the sprint review meeting.

Each sprint is typically 30 calendar days long, and the team meets for about 15-20 minutes on each working day. In these meetings, team members can ask special questions and talk about issues or problems that may impede the team from reaching its goals. Once a sprint has been established, no additional work can be added to it. Additional work will be added to the product backlog, which will then have to be re-prioritized by the product owner.

For the development stage of the Scrum lifecycle, our goal is to have frequent releases of a stable, integrated, and tested product. We will implement mostly the XP process during this phase. As far as the design principles go, this project will utilize the XP simple design practice, where simplicity is the model and the design of the code needs to be testable, understandable, browsable, and explainable. We will also be applying the testing practice native to the XP process, where unit tests are written in advance and all code must pass all unit tests prior to being released.

As part of the coding process, it is essential to implement coding standards. This helps keep the project consistent and structured, thus improving readability of the code, as well as providing ease for refactoring. We will also promote team code ownership where everyone can contribute to all segments of the project without having definitive roles. This encourages team members to work as a whole, which is an important aspect of the XP process. When it comes to writing the production code, we will follow the common XP practice of pair programming where two programmers will generally work together; one will write the production code and the other will review the code.

Integration of the code to the team's repository is a very important concept in development. Here, we will employ the continuous integration practice where we will commit the code often to help detect compatibility problems during its early stages. All of these are good practices to follow because it increases efficiency and reduces the probability of errors in the code, while also promoting a collaborative approach to developing the product. Finally, after development, preparations must be made for the release phase. Prior to the product release at the end of each sprint, the team has to make sure that releases are kept at a sustainable pace and all important principles of the process have been followed. Again, each release must be carefully tested and the product must abide by the core principles mentioned previously. Table 1 contains the core practices of XP, and for this project, a mixed approach of these listed practices combined with Scrum were followed.

1. Planning game	7. Pair programming
2. Small, frequent releases	8. Team code ownership
3. System metaphors	9. Continuous integration
4. Simple design	10. Sustainable pace
5. Testing	11. Whole team together
6. Frequent factoring	12. Coding standards

Table 1: XP Core Practices

3.0 Pre-Game: Planning and Staging

In the Scrum lifecycle, the first stage is the pre-game stage. There are two major phases in the pre-game stage: planning and staging. In planning, the stakeholders who are involved in the project will create a vision and set goals for the product to be produced. During this stage, they will also take into consideration the risks and the time needed for this project. Funding and resources will also be evaluated in this stage. In staging, our goal is to setup the project to fulfill the vision that was made during planning. This phase requires us to list and prioritize product backlogs in order to start the first iteration.

3.1 Vision

Application Description

Team OC is designing an application for our customers to demonstrate the insertion sort algorithm on a mobile platform. Our application will allow users to enter integers that follow certain constraints, and then performs an insertion sort on those integers. Single digit integers (0 through 9) will be accepted in the application. The program requires a minimum of two integers to be entered as inputs to be sorted. The maximum number of integers it will accept is eight. For each iteration, the program will check one integer in the array, progressively moving through the array. As it discovers a misplaced lesser integer, it moves it out of the array and inserts it into the proper position within the sorted list. When the program has iterated through all the integers, the array is sorted.

Goals of the Project

For this application, we will focus on three goals: building an application that meets the customer's requirements, properly and accurately demonstrating the insertion sort algorithm, and delivering a stable, bug-free program to the customer by the scheduled date.

We are working with a local school district that wants to strength its computer science curriculum. Part of that curriculum includes demonstrating different sorting algorithms, the insertion sort algorithm being one of them. By creating an application that accurately demonstrates the insertion sort algorithm, we will allow the school district to introduce this application as an effective educational tool to be used in the classroom setting.

In order for the school district to use this in the classroom, our application must conform to their requirements. We will be working closely with our customer to finalize requirements and make sure each iteration introduces functionality that is relevant to our customer. We will test the application as tasks are being completed to ensure that what we deliver is stable and bug-free. The project will be delivered to our customer on time.

Goals of the Organization

As a new and small startup group, our goals are:

- To create a high quality application that meets the business requirements of our customers
- To have our customers use our application in a live environment
- To familiarize ourselves with Scrum and XP software development methodologies.

We believe our organizational goals to be realistic and attainable.

To meet the first goal, we must fully understand our customer's business requirements and grasp the project goals. Once we have a strong understanding of the customer's needs and wants, we will refer back to them throughout the software development lifecycle. Each requirement or change must align with the business requirements and contribute to solving the problem, so it is important to keep them in mind as we progress through development. This will result in a final product that is relevant to the business requirements and meets the needs of our customer, regardless of what changes, expected or unexpected, we may face along the way.

Our second goal aligns closely with the first goal. From an organizational perspective, we want to deliver a product that our customer can proudly and confidently use in a live environment. We must ensure that our functional and nonfunctional requirements provide value to the customer and that it addresses their business needs. One of the business requirements is to use this in an academic setting, so we must check that the application performs the insertion sort correctly and effectively. An effort should also be taken to see that the application is stable, bug-free, and user friendly.

Our third organizational goal is slightly more difficult to achieve, but is realistic and attainable nonetheless. We will be adhering to Scrum and XP methodologies when creating and developing our product. As we follow the Scrum and XP processes, we will be solidifying what we have learned with hands-on experience. We expect to face challenges and problems including performing tasks in an inefficient manner. These challenges are recognized and accepted, as long as we do not allow them to slow us down significantly and we learn from them. When we face difficulties, we will work together as a team to overcome them in order to stick to our schedule. After we have solved a problem, we will meet and discuss what the issue was, how we can prevent it from reoccurring, and what we learned. We want to follow the Scrum and XP methodologies closely in order to prevent problems. However, we will need to anticipate problems for realism. By learning from our mistakes, we will have a better grasp of the established methodologies.

System Features

We believe these features to be the most essential functional requirements:

- Application should display a full and complete UI when opened
- Users should be able to enter any valid character in the input field
- When the user has entered a valid input and presses the sort button, the application will run an insertion sort
- Users will be informed when an invalid input is entered
- Users can exit the application

System Context Diagram

The diagram shows how external factors and the environment interact with the system.

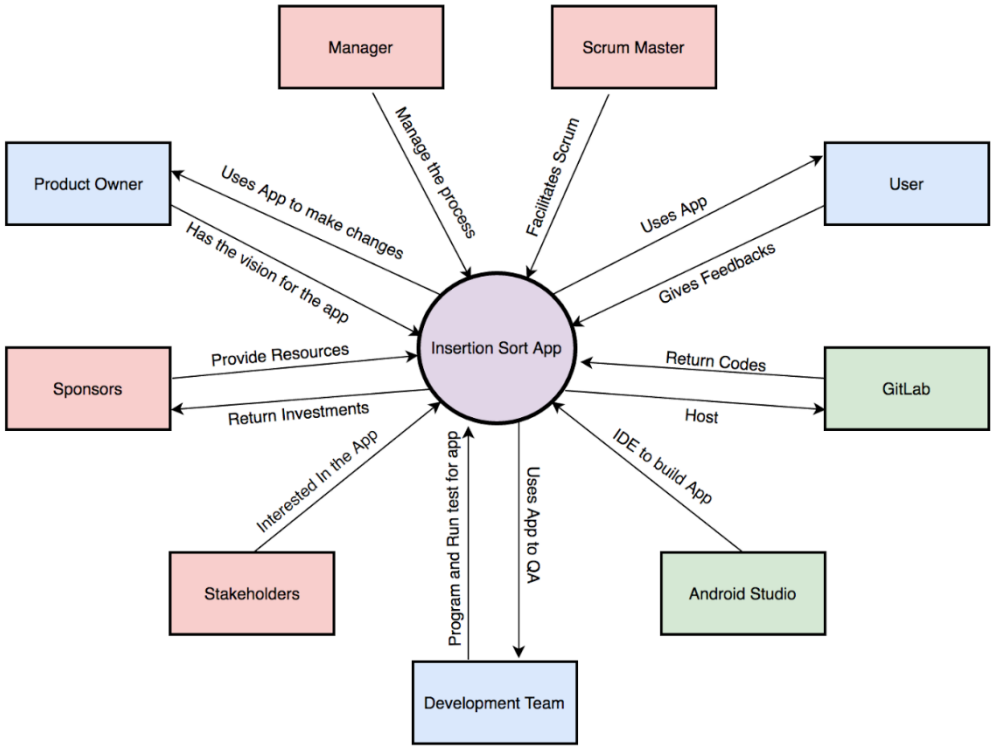


Figure 7: system context diagram

The red-labeled boxes represent people who are a part of the project but are not directly involved. The blue-labeled boxes indicate the people who are directly involved with the process of developing the application. The development team is involved with programming this app. The user and product owner are there to provide feedback and changes to the application. The green-labeled boxes represent the environments where the code base is hosted and the tools used to develop the application.

Stakeholders

We have identified several key stakeholders to work with in regards to our application. Our stakeholders consist of those who are internal to our organization, as well as those who are external.

Internal stakeholders:

- Project manager
- Scrum team

External stakeholders:

- Computer Science teachers who will be using this application

- Head of Computer Science Education in the school district
- Representative from the team that sets education curriculum in the school district
- Representative from the school district's IT department

We have identified several key external stakeholders due to the fact that our customer plans on implementing the application within a school district's computer science curriculum. As a result, teachers using the application are clear stakeholders. They will be able to provide insight into what we should consider to make the application ideal for a classroom setting. The Head of Computer Science Education and the representative from the team that sets the educational curriculum are stakeholders as well because they will be the ones who need to implement this application into the actual curriculum itself. The representative from the IT department will want to ensure that the application conforms to district IT policies.

3.2 Requirements

Functional Requirements

Functional requirements specify how the application will perform under a specific set of conditions. They tend to give a rundown of what developers must implement in order to help users accomplish their tasks. These implementations can be calculations, data manipulations, or any specific functionality that clearly defines what the system is supposed to do. Furthermore, by having the application enable users to accomplish their tasks, functional requirements fulfill both the user requirements and the business requirements of the company itself.

- FR-1: when the application is opened, it shall display the full user interface
- FR-2: users shall be able to enter any valid character when they select the input field
- FR-3: when the user presses the sort button, the application shall check the validity of the input
- FR-4: when the user inputs a valid input and presses the sort button, the application shall perform an insertion sort on the integers entered
- FR-5: the application shall display each iteration of the sort being performed when the user enters a valid input and presses the sort button
- FR-6: users shall be able to exit the application when they press the exit button.

We believe that FR-1, FR-2, and FR-4 should be the highest priority during the first sprint. These three requirements are the basic functions of the application. Without these functional requirements, the application would not be fully functional. Therefore, in our first sprint, we will focus on these requirements first.

Nonfunctional Requirements

Nonfunctional requirements specify the property or characteristic the application must display or some constraints that the system may experience. While functional requirements define what the system is supposed to do, nonfunctional requirements describe how the system is supposed to be. These requirements are better known as "quality attributes" of the application in question. Quality attributes

such as performance, safety, portability, availability, and other characteristics which are deemed important to its users or developers are usually established in this section of the requirements.

- NFR-1: the application shall be available at all times
- NFR-2: this application shall run on Android operating system
- NFR-3: if the application detects fewer than two or greater than eight integers, the application shall present the user with an error message
- NFR-4: if the application detects non-integers, integers less than zero, or integers greater than nine, the application shall present the user with an error message
- NFR-5: for each iteration of the sort, the integer being affected shall be bolded to indicate to the user what action the sort algorithm is performing
- NFR-6: the application shall display instructions to the user telling them how to use the application
- NFR-7: the user interface shall use an appropriate font family and size for clear readability
- NFR-8: if the user does not know what to do, the application shall have a help button
- NFR-9: when the user exits the application, the input will not be saved in the application
- NFR-10: the application shall display credits to the developers of the application

NFR-2, NFR-3, and NFR-4 should have the highest priorities on the sprint backlog because it affects the quality of the application. NFR-2 is important because the application needs to be able to run within an Android environment. NFR-3 and NFR-4 are constraints that the application must abide by for the application to function properly.

Architecturally Influential Factors (AIFs)

Architecturally influential factors are factors that affect the architectural decision of the application. These architectural decisions can be determined by: the customers who this product is being built for; the business requirements that the product needs to meet; the environment this product interacts with; and many other factors that can affect the finished product.

- AIF-1: the customer/user is an important AIF. The software will be developed based on what the user wants. If there are any changes made to the requirements per the user's or customer's request, the system architecture of the software will have to change to accommodate the request.
- AIF-2: cost is another important AIF. The product we are building requires a lot of resources. Without proper funding, we will not be able to supply these resources to develop a quality product. Low funding will hinder the production of the software, and will place a constraint on its design and affect its overall quality.
- AIF-3: time is also an important AIF. Without proper time management, the product will not be completed in a timely manner. Poor time management leads to developers not having enough time to implement features or functions for the design and software that would make the product better. Therefore, we believe that time is an important AIF because it will affect the system's architecture.

Requirement Management

It is important to maintain and manage the requirements. There must be requirements management so that user requests do not steer us away from the vision and goals of the project. Our company has established a requirement change process that will effectively manage any proposed changes. Like any other requirements change process, we have come up with an agreed upon requirements baseline. Any changes beyond the baseline are to go through the requirements change process. If a user wants to make a change in the requirements, the requested change will go through our change control board (CCB). The CCB consists of a small group of project stakeholders who will evaluate the request made. They will take into consideration how much time and how many resources will be needed to make such a change. They will also identify if the request is within the scope of the current project. If it is not in the scope of the project, the request may be moved to a future iteration or into a different project.

However, if the request does fall within the scope of the project in question, we can incorporate it into the list of requirements and prioritize its importance relative to the other requirements. All changes will be recorded and retained for future reference in case we need to revert back to an earlier version of the requirements. All requirement change statuses will be monitored for issues and completion within the scheduled time frames.

3.3 User Stories for Planning

User stories are features that the user requests. These stories determine how much time should be estimated for the team to plan their release. The user stories should be accomplished within one day to three weeks. User stories should never be too complicated or detailed because the XP process requires the customer representatives to be available should there be a need to verify any of the stories. The product owner has the responsibility of prioritizing the user stories that the team should work on for each iteration. Stories are created before the beginning of an iteration and not in the middle of them because Scrum does not allow any work to be added in the middle of an iteration. Scrum and XP allow for flexibility and adaption in case there is a need to change any of the user stories.

Our team sat down together and came up with the user stories by acting out the roles of the users. We came up with 12 user stories and wrote them onto index cards that were then placed on our storyboard. We prioritized the user stories based on their business value and risk. We generated a sprint backlog of tasks, which were derived from the user stories. We decided to place the user stories that had the highest business value and risk into the first iteration. Afterwards, we prioritized those tasks and estimated the time needed to complete them in the form of person-hours of effort. Our development team looked over the tasks of the prioritized user stories and worked on a plan to complete them within the first iteration. Once the first iteration was complete, we reviewed the feedback that we received and checked for additional changes to be made to the user stories. After the review, we moved on with deciding which tasks would be completed in the next iteration. The process was repeated for the remaining user stories, and we felt it was reasonable to split them into two parts: Iteration II and Iteration III.

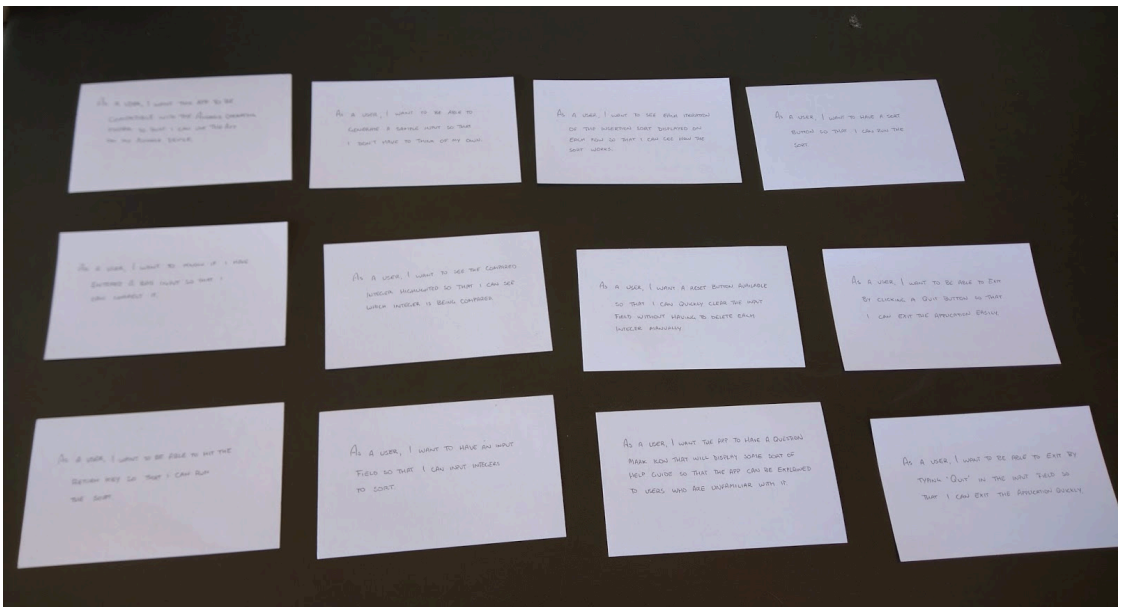


Figure 8: user stories

Our user stories are formatted in Table 2 below:

User Story Code	User Story
US-1	As a user, I want this app to be compatible with the Android operating system so that I can use the app on my Android device.
US-2	As a user, I want to have an input field so that I can input integers to sort.
US-3	As a user, I want to have a sort button so that I can run the sort.
US-4	As a user, I want to be able to hit the Return key so that I can run the sort.
US-5	As a user, I want to know if I have entered a bad input so that I can correct it.
US-6	As a user, I want to see each iteration of the iteration sort displayed on each row so that I can see how the sort works.
US-7	As a user, I want to see the compared integer being highlighted so that I can see which integer is being compared.
US-8	As a user, I want the app to have a question mark icon that will display some sort of help guide so that the app can be explained to users who are unfamiliar with it.
US-9	As a user, I want to have a reset button available so that I can quickly clear the input field without having to delete each integer manually.
US-10	As a user, I want to be able to generate a sample input so that I do not have to think of my own.

US-11	As a user, I want to be able to exit by typing 'quit' in the input field so that I can exit the application easily.
US-12	As a user, I want to be able to exit by clicking a quit button so that I can exit the application easily.

Table 2: user stories

3.4 Product Backlog and Estimates

As part of the development process, the product backlog is created during a meeting with the Scrum team and all of our stakeholders. The product backlog helps our team keep track and trace items in the backlog back to the business requirements. The backlog is transparent and highly visible to everyone who is involved and is always kept up to date. Prioritization is a constant ongoing activity. We select which tasks are setup for each upcoming iteration and which items are removed from the backlog. The product owner is ultimately responsible for the product backlog and the prioritization of each task. Everyone on the team is responsible for keeping the backlog up to date.

ID	User Story	Task	Priority	Estimated Hours
US-1	As a user, I want this app to be compatible with the Android operating system so that I can use the app on my Android device.	Program and design app under Android environment	High	2
US-2	As a user, I want to have an input field so that I can input integers to sort.	Create input field	High	1
US-3	As a user, I want to have a sort button so that I can run the sort.	Implement insertion sort algorithm Set sort button to implement sort function	High	3
US-4	As a user, I want to be able to hit the Return key so that I can run the sort.	Add return key handler into input	Low	1
US-5	As a user, I want to know if I have entered a bad input so that I can correct it.	Implement alert box for invalid inputs	High	2
US-6	As a user, I want to see each iteration of the iteration sort displayed on each row so that I can see how the sort works.	List phases of Insertion Sort	High	1

US-7	As a user, I want to see the compared integer being highlighted so that I can see which integer is being compared.	Display bolded comparison value in output	High	1
US-8	As a user, I want the app to have a question mark icon that will display some sort of help guide so that the app can be explained to users who are unfamiliar with it.	Create a 'help' button Set 'help' button to display product information	Medium	2
US-9	As a user, I want to have a reset button available so that I can quickly clear the input field without having to delete each integer manually.	Create 'reset' button Set 'reset' button to clear input fields	Low	1
US-10	As a user, I want to be able to generate a sample input so that I do not have to think of my own.	Create 'random' button Create random function	Medium	3
US-11	As a user, I want to be able to exit by typing 'quit' in the input field so that I can exit the application easily.	Implement secondary method of quitting app	Low	1
US-12	As a user, I want to be able to exit by clicking a quit button so that I can exit the application easily.	Create 'exit' button	Low	1

Product Backlog (Overview)

Ticket #	Type	Task	Priority	Estimated Time (hours)
1	Main function	Build Android skeleton that supports Nougat using SDK 24	Critical	3
2	Main function	Create Input Field	Critical	1
3	Main function	Create 'SORT' Button	Critical	1
4	Main function	Implement Insertion Sort Algorithm	Critical	2
5	Improve user functionality	List phases of Insertion Sort	Critical	2
6	Improve user functionality	Implement alert box for invalid inputs	Major	1

7	Improve user functionality	Display bolded comparison value in the output	Major	1
8	Enhanced features	Implement 'RESET' button	Major	1
9	Enhanced features	Create 'EXIT' button	Major	1
10	Enhanced features	Add return key handler to input field	Major	1
11	Enhanced features	Add 'RANDOM' button	Minor	2
12	Enhanced features	Implement 'HELP' button	Minor	1
13	Enhanced features	Implement secondary method of quitting the app	Minor	1

Table 3: product backlog overview

With adequate product owner involvement, our developers can build more closely aligned to deliver what the stakeholder wants and what the customer needs. Table 4 shows an example of our sprint backlog. After analyzing our user stories, functional, and nonfunctional requirements, these first four tasks allows us to create an up and running application. They are planned in our first iteration and they are our highest priority.

ID	Task	Priority	Estimated Time (hours)
US-1	Program and design app under Android environment	High	1
US-2	Create INPUT Field	High	1
US-3	Create SORT Button	High	1
US-6	Implement insertion algorithm AND list phases of insertion sort	High	4

Table 4: sprint backlog

3.5 Architectural Spike

Architectural Spikes

In Extreme Programming, a spike solution is to create a task that is directed at laying a question to rest, or to pull together information. These tasks are generally not aimed at producing anything towards the product. The goal of architectural spikes is to lessen the risk of encountering a distressing technical problem. Architectural spikes are often very minimal programs used for the sole purpose to investigate possible solutions. When working on architectural spikes, it is important to only take up the problem under consideration, and to ignore all other matters.

System Metaphor

To assist with communication between ourselves and our customers when discussing application design and requirements, we have defined a system metaphor. The metaphor that we have used to represent our system is a discount campaign run by a specific brand through an online retail store. To take advantage of the discount, users would place qualifying items into their shopping carts. When they have completed the checkout process, the online store would prepare the products for shipment, and the package carrier would deliver the products to the customer.

The campaign places a constraint on the quantity of eligible items for the discount, which matches the limitation on the number of integers allowed within the input field. If users input too few or too many items, they would not qualify for the discount or the ability to sort their input. To qualify for the discount, each item must also be from a particular brand. This falls in line with the system's requirement that integers must be between 0 and 9 since those numbers can be classified as a group or brand. The input field is synonymous with a user's shopping cart, and the clicking of the "Sort" button is the same as the user going through the checkout process. The sorting of the user's input values is similar to the discount verification, payment processing, and preparing for shipment phases of the online order process. After this processing is complete, the user has an expectation of receiving the discounted goods that they have purchased, which is the same as the user expecting to see a sorted array of integers. The sorting snapshots in the result set are also similar to the order receipt that the user receives detailing their purchases, as well as the associated costs and discounts applied. Using this metaphor has made it easier for us to discuss design decisions and requirements without having to get into the nitty-gritty technical details, which may confuse our intended audience.

High Risks

We believe one architectural spike will be parsing user input to check its validity. We believe that this will be a spike because we have not done input checking on Android before, and are unsure if there are classes that support it. We plan on digging into the Android API for string parsing functions and exploring different data structures to store the user input to discover a solution. We will also look into regular expressions to see whether that will lead to the desired results.

Another architectural spike that we will have is satisfying the need to highlight the integer that is being compared. We believe this will be a spike because we are unsure as to whether to whether or not Android programming allows developers to highlight or bold individual characters within a string object. In an effort to find a solution to this problem, we plan on reading the String documentation in the Android API. We see whether there are ways to manipulate the style of specific characters in a string. If it is not possible, we will look to see if Android supports HTML.

3.6 Release Planning

After determining which requirements for our Insertion Sort App are most important for achieving business success, and prioritizing each task in the product backlog, we created a release plan. From the very beginning, our goal is to follow Agile methods. We are planning to release deliverables in multiple sprints throughout the course of the project. Table 5 illustrates our release plan. It helps correlate some measure of time to complete business requirements and tasks which are then updated in the product

backlog. Release planning also provides a way to gain customer feedback. After each sprint, the Scrum development team delivers a functional product to the user which encourages the user to provide product feedback. Adequate customer involvement and valuable feedback helps the team adapt and adjust to any product changes requested by the user. While planning for any iterations and product releases, it is important to consider some contingency buffers. This helps accommodate any new or "last-minute" changes in the requirements that the product owner might ask for. Without some type of buffer in the timeline, any product rework could cause our schedule to slip and fall behind, which ultimately adds additional time and cost. We account for these buffers when planning our estimated times.

Name	Type	Estimated Start	Estimated End	User Stories	Requirements
Iteration I	Main Functions	Sept. 6	Sept. 9	US-1 US-2 US-3 US-6	The initial release of the app with basic user requirements
Iteration II	Improving User Functions	Sept. 12	Sept. 15	US-5 US-7 US-9 US-12	Enhanced usability and performance
Iteration III	Enhancing Features	Sept. 19	Sept. 22	US-4 US-8 US-10 US-11	Satisfy the business value, and finalize the Insertion Sort App and include any special features that distinguish our app from our competitors.

Table 5: release planning table

Checkpoints helps us monitor our project and provide a basis for evaluation and assessment; we want to make sure the project is going as planned. We have our project set into different phases. Every project phase is passed through a predetermined checkpoint to ensure project requirements, tasks and deliverables are being met. At each checkpoint, we stop, analyze, and make formal assessments in order to identify any potential issues in the current phase before moving forward. Table 6 illustrates our project monitoring table. The table includes the status, date, and week each checkpoint was reached. The table also displays the amount of project resources that we utilized in order to reach the checkpoint.

Week	Date	Checkpoint	Resources %	Status
1	9/6/2016	Planning	45%	Completed
2	9/9/2016	First Prototype	100%	Completed

3	9/22/2016	Debugging/Code	100%	Completed
4	9/24/2016	QA test	70%	Completed

Table 6: project monitoring table

3.7 Technology Preparation: Android Development Environment

Regardless of how detailed the planning and staging is, Android application development requires a team with a technical background. There are many resources available on the internet that can be used to get the team started on making the Insertion Sort application.

The first way we plan on preparing the team for Android development is through the use of tutorials. The members of the team who are unfamiliar with Android programming will start off by downloading Android Studio and following the Creating an Android Project tutorial to get an idea of how projects are created and how to locate files in the file structure [4]. The tutorial also exposes the user to the Android Studio interface. After completing the Creating an Android Project tutorial, individuals will follow the Running Your App tutorial which presents a step-by-step guide on how to run the application on a device or emulator [5].

When the team has an adequate grasp of Android programming, the individuals will follow different guides on Android Developer based on what their needs are [6]. They will also be asked to explore the Android Nougat API to utilize the built-in classes and methods [7]. Some team members who have had previous exposure to Android programming may use deprecated methods as long as they perform a version check prior to each usage. Android Studio indicates when a deprecated method is used, and we recommend the developer to switch to some new API or to find some other way of doing it.

The team will use Slack to stay in contact with one another throughout the development process. All individuals are held responsible for finishing their assigned tasks, so it is their duty to reach out to others if assistance is needed. For tasks that are dependent on other parts to be completed, Slack messages will be exchanged to get statuses. The majority of roadblocks should be handled through Slack messages, if possible. In events where Slack is not an appropriate medium, email or phone communication will be utilized.

The team will use email, Slack, and phone calls to communicate. Since the team cannot meet daily for Scrum Meetings, we will set aside a time every two or three days for a conference call. Team members are expected to make time in their schedule to take part in the conference calls. Every individual should check their email and Slack messages daily, and respond to one another within twenty-four hours of receiving the inquiry. Email will be used to communicate important information such as group meeting locations, conference call times, and major deadlines. There will a handful of in-person meetings that will take place throughout. Each team member should make every effort to attend those meetings.

The team will be using GitLab as version control for the project [8]. Each member will have read and write access to the code base. Members should create their own branches locally as working copies, and push to the development branch as tasks are completed. By doing this, the development branch is always working and the team can pull from the development branch to start working from the latest

version. This will ultimately help reduce the number of code conflicts the team will face and allow individuals to have an idea of what the rest of the team is working on. As code is pushed to GitLab, individuals should review their peers' work and provide pertinent comments or feedback.

For files that are not related to Android development, the team will use a shared Google Drive folder to store and share documents. We anticipate keeping meeting notes, any written documents, and useful media available on the internet.

4.0 Development

4.1 Iteration I

User Stories

User Story Code	User Story
US-1	As a user, I want this app to be compatible with the Android operating system so that I can use the app on my Android device.
US-2	As a user, I want to have an input field so that I can input integers to sort.
US-3	As a user, I want to have a sort button so that I can run the sort.
US-6	As a user, I want to see each iteration of the iteration sort displayed on each row so that I can see how the sort works.

Iteration Planning

On September 6, 2016, we held our sprint planning meeting for the first iteration. The meeting consisted of the Scrum Master, the product owner, and the development team. Prior to the start of the meeting, the product owner had already ensured that he was content with the current product backlog priorities. The product owner, alongside with the development team, then moved the top priority items from the product backlog onto the sprint backlog. Our priority items were in the form of user stories and the top priority user stories for Iteration I were US-1, US-2, US-3, and US-6. Prioritization of these user stories were driven by the highest business value and risk. We then came up with a sprint goal for the first iteration and our sprint goal was to implement the basic insertion sort functionality including the input text field, the sort button, and displaying the intermediate phases of the sort function to the user.

Based on our defined sprint goal, we then created sprint tasks out of the sprint backlog priority items (user stories), and we then estimated the time it will take to complete these tasks. Our team agreed that the work assigned can be completed within the sprint timebox of 4 days. So, after the sprint was established, allocation of additional work to the sprint was no longer allowed.

For each task that was created, we assigned story points that indicated the estimated level of difficulty. We used button creation as our baseline difficulty and rated that as a 1. From there, we estimated each remaining task by assigning a Fibonacci sequence number to each one relative to the established baseline. We assigned the largest values to insertion sort implementation and the random sequence generator. The next largest values were assigned to the Android project skeleton, the exception handler, and the help display. Tasks that received the lowest estimate included the input field, the return key handler, the list of insertion sort steps, the bolded comparison values in each sorting step, the reset button, the ability to exit by typing “quit”, and the exit button. We then recorded these tasks onto Post-

It notes and placed them all into a requirements tracking chart. To keep track of each task's status, we created four distinct status tracks: Pending, In Progress, QA, and Done. For the first iteration, we placed the tasks that were created for US-1, US-2, US-3, and US-6 into the In Progress track. At each Scrum meeting, we would then update the corresponding chart by moving the Post-Its into their respective tracks based on their current status.

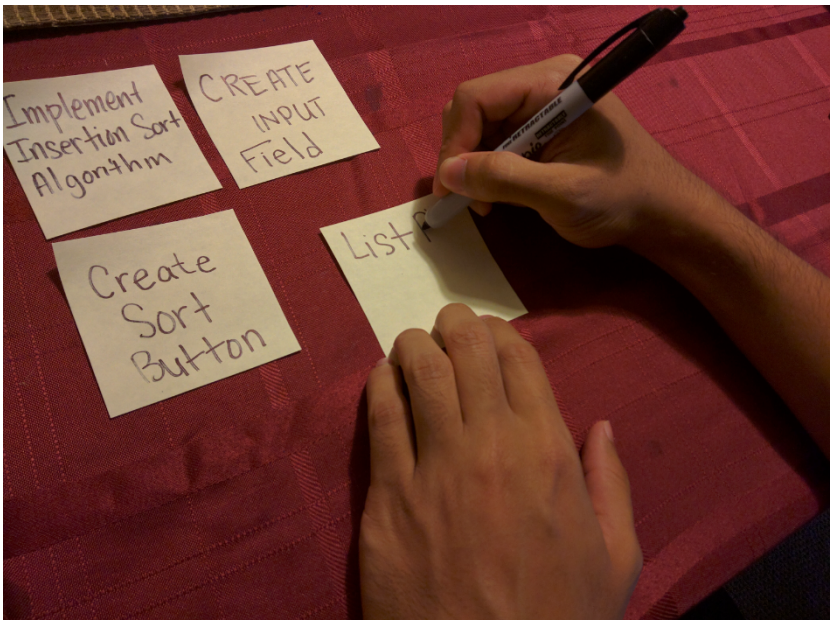


Figure 9: writing out tasks

Figure 10: sprint backlog for Iteration I

Managing

On the first day of our sprint, we made sure the meeting is attended by all team members. As we gathered up all of our project resources, our development team worked closely in a single room environment. We had a total of 6 members working side-by-side while keep each other up to date with what tasks each member was working on. We set a very simple agenda listing all the things we need to do for the day. During development, we would have open air discussions and we would talk about ideas, improvements and goals that can be made to the project. For example, during our open air discussions, we came up with an idea and we noticed that Danny Zhang was really proficient at programing on Android Studio so we decided to practice pair programming with another team member, Kevin Tran, our lead developer. This is a form of cross training that helps one member of the team learn from another member in the team that is more knowledgeable in a certain area. As we all have different skill set, we all can learn from one another. It also allows the project to move at our projected pace since everyone is working together and no one was overloaded with work that they could not finish. Furthermore, a collaboration using pair programing promotes a sense of collective ownership among our team members so that everyone is motivated to work harder to create new ideas with instant feedback from other members of the group.

Designing

For this first iteration, we decided to create an app that would display an input field and a sort button. When the sort button is clicked, the application would perform the sort and display the result underneath the sort button. The input array would also be included in the output.

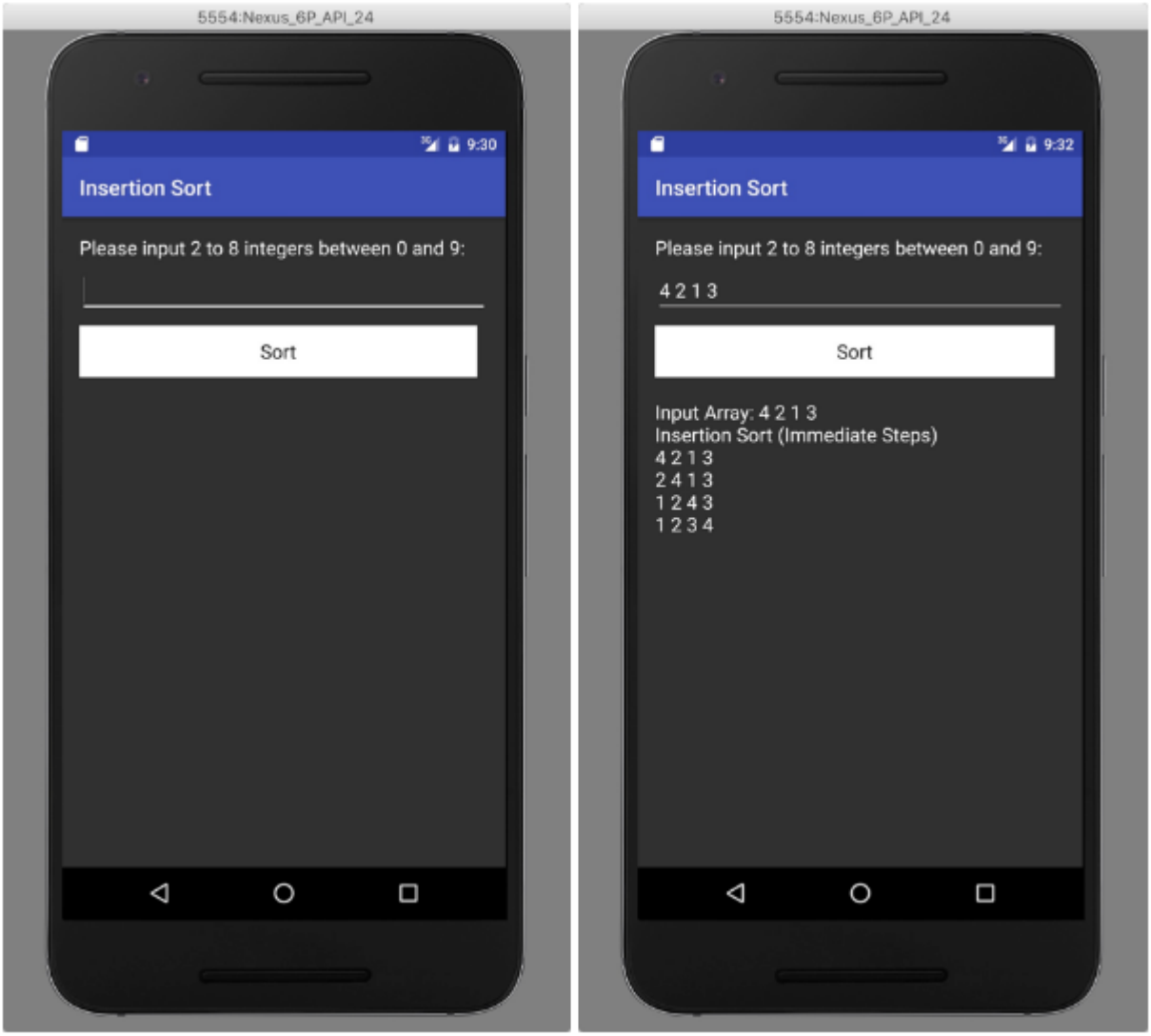


Figure 11: Iteration I design

Coding

When we kicked off the coding efforts for the first iteration, it did not take long before we noticed the how writing proper user stories would be to the team's benefit. After we set up the coding environment on Android Studio, we were able to split the user stories up, and begin coding. In the first iteration, we were able to finish all four of the user stories. Throughout the entire process, any questions related to the user stories were taken to Alex, the product owner. Alex would clear up any misunderstandings and the team would move forward with their assigned task.

US-1

We used Android Studio to create a new Android project. This basic project ensures that the code and application will be compatible with the Android operating system. We used this as our project baseline.

US-2

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:background="@color/colorBackgroundMain"
  android:gravity="end"
  android:layout_height="match_parent"
  android:layout_width="fill_parent"
  android:orientation="vertical"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="teamoc.com.insertionsort.MainActivity" >

  <!-- Input Label -->
  <TextView
      android:labelFor="@+id/input"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:text="@string/sort_instructions"
      android:textColor="@color/colorTextView"
      android:textSize="@dimen/text_size_main" />

  <!-- Input Field -->
  <EditText
      android:id="@+id/input"
      android:backgroundTint="@color/colorTextEditBackground"
      android:imeOptions="actionGo"
      android:inputType="text"
      android:layout_height="wrap_content"
      android:layout_marginBottom="@dimen/input_margin_bottom"
      android:layout_marginTop="@dimen/input_margin_top"
      android:layout_width="fill_parent"
      android:textColor="@color/colorTextView"
      android:textCursorDrawable="@null"
      android:textSize="@dimen/text_size_main" />
  ...
</LinearLayout>
```

strings.xml

```
<resources>
  <string name="input_array">Input Array:\u0020</string>
  ...
```

```
</resources>
```

US-3

MainActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
  ...
  Button sortBtn = (Button) findViewById(R.id.sort);
  sortBtn.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View view) {
         sort();
     }
  });
  ...
}

private void sort() {
  // Hides keyboard when Sort button is pushed
  View view = this.getCurrentFocus();
  if (view != null) {
     InputMethodManager imm = (InputMethodManager)getSystemService(
        Context.INPUT_METHOD_SERVICE);
     imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
  }

  TextView inputView = (TextView) findViewById(R.id.input);
  String inputStr = inputView.getText().toString().toLowerCase().trim();
  TextView output = (TextView) findViewById(R.id.output);
  String outputStr;

  String[] inputArray = inputStr.split(" ");
  outputStr = getResources().getString(R.string.input_array) + inputStr +
     "<br>" + getResources().getString(R.string.insertion_sort_imm_steps) +
     "<br>" + InsertionSort.sortWithOutput(inputArray);

  Spanned spannedResult;
  if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N){
     spannedResult = Html.fromHtml(outputStr, Html.FROM_HTML_MODE_LEGACY);
  } else {
     spannedResult = Html.fromHtml(outputStr);
  }

  output.setTextColor(Color.WHITE);
  output.setText(spannedResult);
}
```

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <!-- Button Bar -->
    <LinearLayout
        android:gravity="end"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal" >

        <!-- Sort Button -->
        <Button
            android:id="@+id/sort"
            android:background="@color/colorBtnBackground"
            android:layout_height="wrap_content"
            android:layout_marginEnd="@dimen/button_spacer"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:text="@string/sort_btn_text"
            android:textSize="@dimen/text_size_main"
            style="@style/Widget.AppCompat.ButtonBar" />

    </LinearLayout>
    ...
</LinearLayout>
```

strings.xml

```
<resources>
   ...
   <string name="sort_btn_text">Sort</string>
   ...
</resources>
```

US-6

InsertionSort.java

```
package teamoc.com.insertionsort;

public class InsertionSort {

   /**
```

```java
    * Sorts an array of integers (as strings) and takes a snapshot of each
    * phase during the sorting process.
    *
    * @param input The list of integers to be sorted
    * @return An HTML-formatted output of the sorting process
    */
   public static String sortWithOutput(String[] input) {
       if (input.length == 0) return "";
       if (input.length == 1) return input[0];

       StringBuilder sb = new StringBuilder();

       // Sort the array
       for (int i = 1; i < input.length; i++) {
           // Print the array's current state
           sb = takeSnapshot(input, sb);

           int comparisonValue = Integer.valueOf(input[i]);
           int j = i - 1;
           while (j >= 0 && Integer.valueOf(input[j]) > comparisonValue) {
               input[j + 1] = input[j];
               j--;
           }
           input[j + 1] = comparisonValue + "";
       }

       // Print the result
       sb = takeSnapshot(input, sb);

       return sb.toString();
   }

   /**
    * Takes an HTML-formatted snapshot of the input array in its current
    * state.
    *
    * @param input The list of integers to be sorted
    * @param sb The HTML-formatted output
    * @return An HTML-formatted snapshot of the input array
    */
   private static StringBuilder takeSnapshot(String[] input,
                                             StringBuilder sb) {
       for (int m = 0; m < input.length; m++) {
           sb.append(input[m]);
           sb.append(" ");
       }
       sb.append("<br>");

       return sb;
   }
}
```

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <!-- Results -->
    <TextView
        android:id="@+id/output"
        android:gravity="start"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:maxLines="15"
        android:paddingTop="@dimen/output_padding_top"
        android:singleLine="false"
        android:textColor="@color/colorTextView"
        android:textSize="@dimen/text_size_main" />
</LinearLayout>
```

Testing

US-1

After creating the base project, we set up a Nexus 6P emulator running Android Nougat x64 to test a basic Hello World app. We also installed the base application on a Nexus 6 running Android Nougat to confirm that the project runs on a real Android device.

US-2

After adding the input field to activity_main.xml, we recompiled the project and ran it on the Nexus 6P emulator. We checked to make sure that the input label displayed the text properly and that the input field was present. We input characters into the box and confirmed that the input matched the displayed text in the input field. Both the soft keyboard and computer keyboard were used to test input registration.

US-3

We added the code to MainActivity.java and activity_main.xml. We compiled the project and ran it on the Nexus 6P emulator. We checked that the Sort button was created and that its location was under the input field. The text for the Sort button was also examined for spelling. We also tapped it on the emulator to see the button getting pressed, and released. We created a listener and had some temporary test code to make sure that pressing the button can lead to an event.

US-6

We created a new class for the insertion sort method, InsertionSort.java. We also added a new TextView in activity_main.xml for the text to display at. We compiled the project and ran it on the Nexus 6P emulator. We checked to make sure that the sorting was done properly. To do so, we input the following strings into the input field: “1 0”, “2 6 4 3”, and “9 8 7 6 5 4 3 2 1 0”. We also adjusted the style

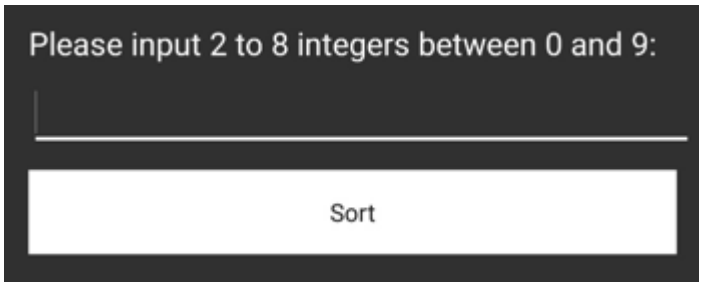
of the TextView based on what looked good on the Nexus 6P. We visually checked to make sure that the displayed numbers in the text output matched the intended text, and that the number of displayed lines matched the numbers of steps in the insertion sort. Lastly, we checked to make sure that the final line contained the integers in a sorted order, with spaces in between each integer.

Small Releases

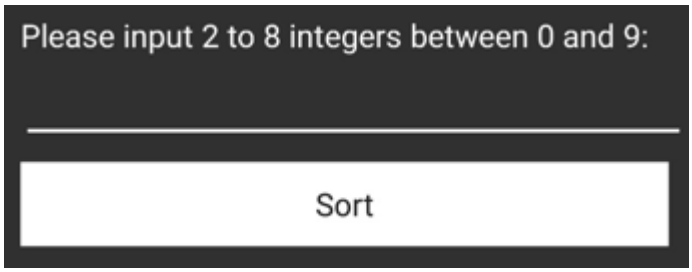
After the testing phase, we distributed the application to the key stakeholders for user feedback. While the application passed our testing phases, we needed to make sure that our application was working properly for our customers. The feedback would also allow us to make sure that the application aligned with their vision.

The users confirmed that each user story was successfully implemented in the release; however, there was a minor design issue that they asked us to correct. The stakeholders felt the font on the sort button may be too small for some users. Luckily this issue was minor and easily fixed. We addressed the issue by making the font size larger and released the updated version to the stakeholders for their feedback. They approved the new font size and the application for this iteration.

Before



After



Daily Scrum Meetings and Sprint Review

We held our daily Scrum meeting at 9:00pm every single day. Since some of us have work in the daytime the best time to hold our Scrum meeting was at 9:00pm. Being an online class, we were only able to meet up in person once per iteration. The rest of our daily Scrum meetings were held on Google Hangouts.

During our daily stand-up, we would start off by answering the following three questions:

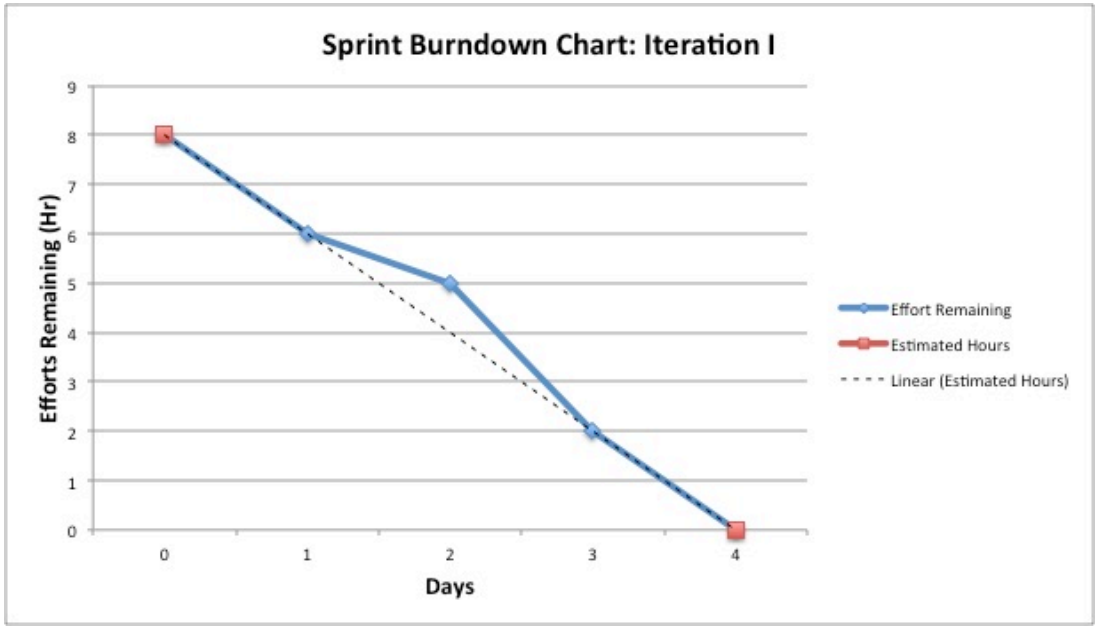
1. What have I completed since the last Scrum meeting?
2. What will I work on between now and next Scrum meeting?
3. What are the obstacles that are preventing me from meeting the iteration goals?

Figure 12: picture of a stand-up meeting that we were able to do in person

We met up for our sprint review in person after each iteration. During our sprint review we compared our sprint backlog to the tasks on the product backlog. We made sure that each tasks were completed in a timely manner and that it satisfies the requirements in the product backlog.

We started our sprint on September 6, 2016 and ended on September 9, 2016. In this iteration we tried to implement the main functions of the application. The first day of the sprint, we set up the Android code skeleton on Android Studio and started coding the input field and the sort button. On the second day, we created the insertion sort class. Implementing the insertion sort algorithm and finding a way to display the phases of the sorting app was a little complex so we decided to do pair programming to figure it out. The pair programming helped us identify the issues and we were able to implement the sorting algorithm by the end of third day. The last day of the sprint, we were able to finish up our tasks and then we did an acceptance test using a Nexus 6P emulator. The application ran smoothly and we were able to finish our task for the first iteration.

Figure 13: Sprint Review – Going over the iteration & comparing it with the Sprint Backlog



Graph 1: Sprint Burndown Chart for Iteration I

The following graph is a burndown chart for Iteration I. This graph allows us to track the amount of working hours that we have left during each iteration. The x-axis shows the days in the sprint and the y-axis shows the hours that we have left to finish the tasks. We allowed ourselves 8 hours to start in this iteration. First day we were able to spend a few hours to set up the environment and implement some of the tasks. On days 2 and 3 we spent most of those hours implementing the sort algorithm. We were a little off schedule on day 2 since we had to figure out how to properly implement the sorting algorithm. By day 4 we were able to finish the application and bring it to acceptance testing.

4.2 Iteration II

User Stories

User Story Code	User Story
US-5	As a user, I want to know if I have entered a bad input so that I can correct it.
US-7	As a user, I want to see the compared integer being highlighted so that I can see which integer is being compared.
US-9	As a user, I want to have a reset button available so that I can quickly clear the input field without having to delete each integer manually.
US-12	As a user, I want to be able to exit by clicking a quit button so that I can exit the application easily.

Iteration Planning

Our process for the second iteration was very similar to the first. On September 12, 2016, we held our sprint planning meeting for the second iteration. The meeting consisted of the same individuals which were the Scrum Master, the product owner, and the development team. Upon successfully completing all the tasks in the sprint backlog from the first iteration, the product owner, alongside the development team, then moved the next top priority items from the product backlog onto the sprint backlog. Our priority items were in the form of user stories and the top priority user stories for Iteration II were US-5, US-7, US-9, and US-12. Our sprint goal for the 2nd iteration was to improve user functionality including implementing the alert box for invalid inputs, adding the reset button and exit button, and adding the visual aid of highlighting the comparison value in the output.

Based on our defined sprint goal, alike Iteration I, we created sprint tasks out of the sprint backlog priority items and we then estimated the time it will take to complete these tasks. As usual, our team ensured to the best of our knowledge that the work assigned can be completed within the sprint timebox of 4 days, so, after the sprint was established, allocation of additional work to the sprint was no longer allowed.

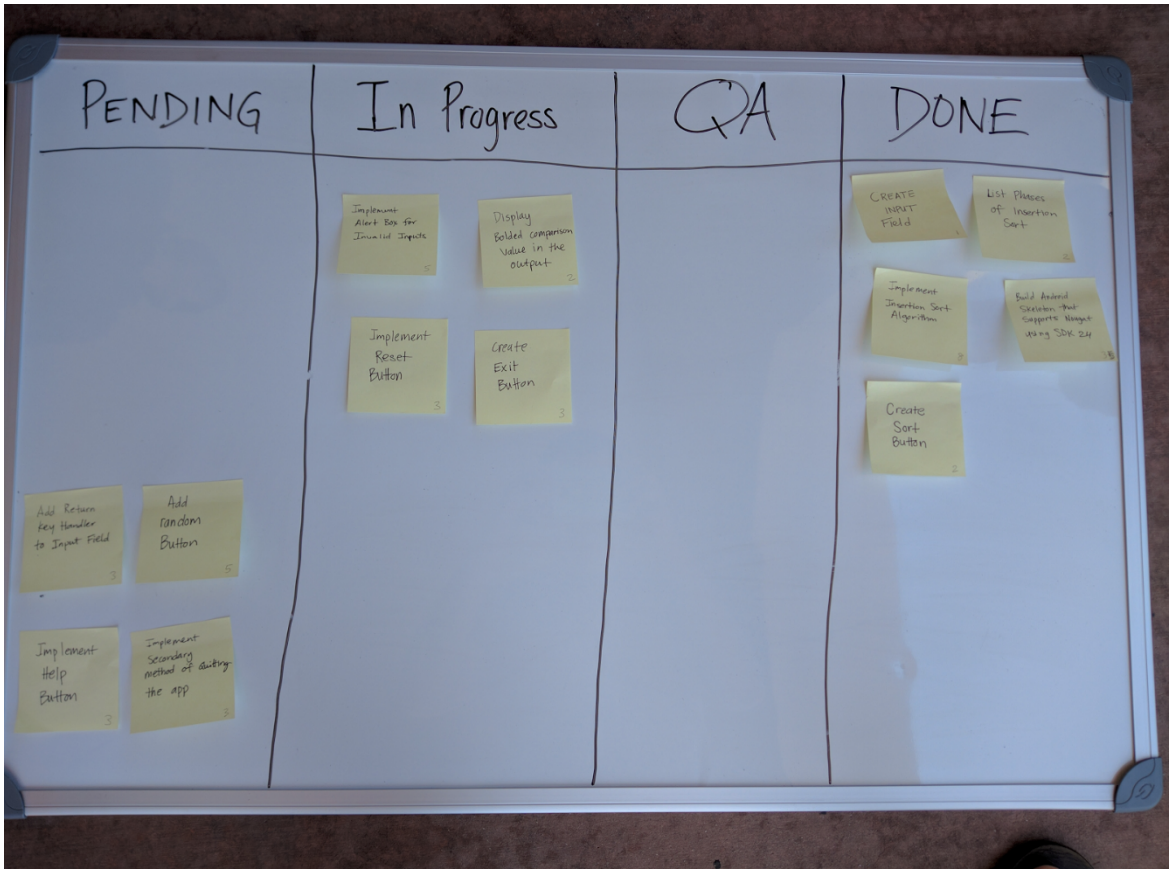


Figure 14: Sprint Backlog for Iteration II

Managing

Ever since practicing paired programing during our first iteration, our productivity increased dramatically. We are using pair programing again to complete our task for the second iteration. Before any work begins, we set an agenda for the day. As we gather up all of our project resources, all six team members worked together in the development room with two people programming together. One team member programs while the other reviews the developing source code. The reviewer must be attentive to catch any code that may accidentally change a function. To keep track of progress, a member of the team will log in our completed task and update any changes.

Designing

For the second iteration, we chose to create a reset button adjacent to the sort button to unify the two controls. By keeping them in line with each other, users would not be confused about which buttons would affect the data in the input field. We also decided to display the invalid text as an alert message when the user pressed the sort button if an invalid input was detected. The exit button was placed in the upper-right corner instead of the current button row since it was not specifically linked to the input field.

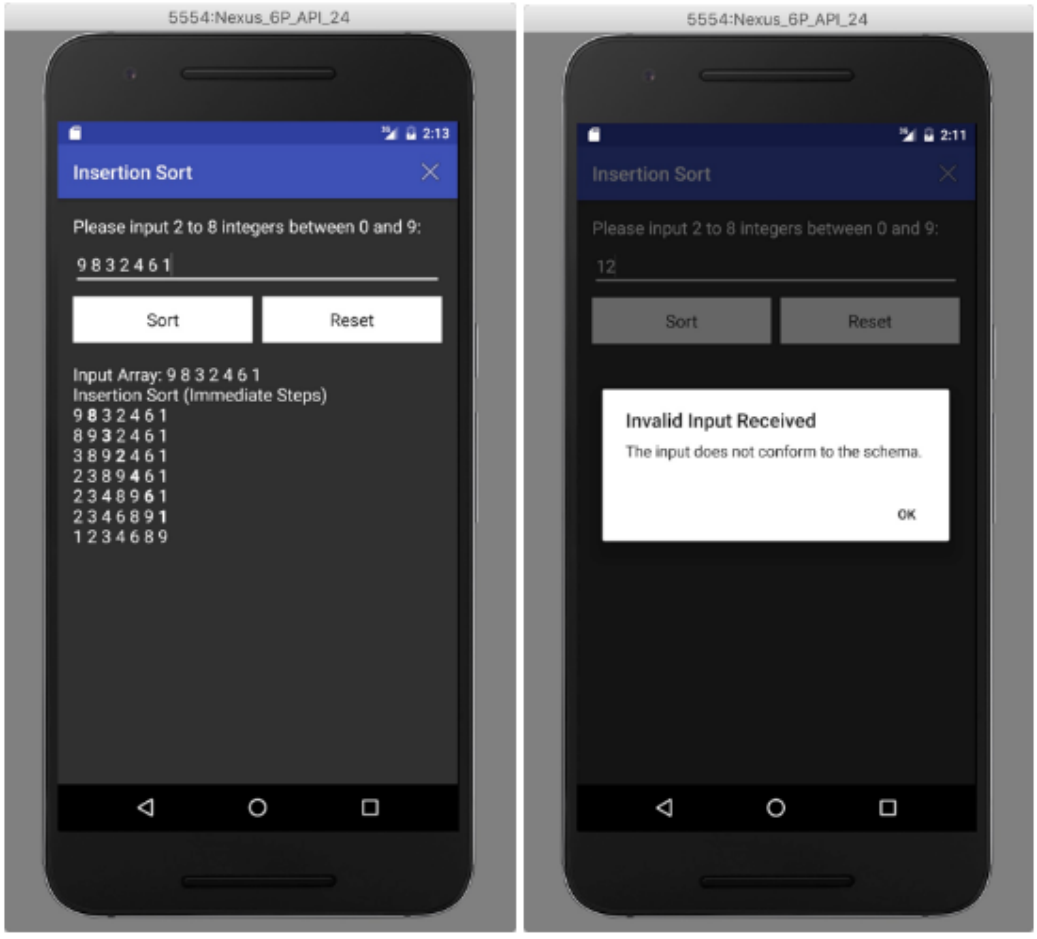


Figure 15: Iteration II design

Coding

Coding for the second iteration was very similar to the first iteration. Two of our user stories in this iteration were improvements of what was accomplished in the first iteration. The reset and exit functions were relatively straightforward. There were some questions on the implementation of the user stories. For one, we needed to have more details on how the bad input message would be displayed. Our product owner, Alex, thought through the issue and decided it would be best to have the input error message display through a popup message. By following the Scrum methodologies, we were able to get the roadblock cleared up without falling behind on the schedule.

US-5

MainActivity.java

```
public void sort() {
  ...
  TextView inputView = (TextView) findViewById(R.id.input);
  String inputStr = inputView.getText().toString().toLowerCase().trim();
  ...
  String regex = "[0-9](\\s[0-9]){1,7}";
  String outputStr;

  if(!inputStr.matches(regex)) {
      outputStr = getResources().getString(R.string.error_line_1);
      AlertDialog.Builder errorAlertBuilder = new AlertDialog.Builder(this);
      errorAlertBuilder.setMessage(outputStr);
      errorAlertBuilder.setTitle(getResources().getString(R.string.error_title));
      errorAlertBuilder.setPositiveButton(getResources()
          .getString(R.string.ok_text), null);
      errorAlertBuilder.setCancelable(true);
      AlertDialog errorAlert = errorAlertBuilder.create();
      errorAlert.show();
      Button okTextColor = errorAlert.getButton(DialogInterface.BUTTON_POSITIVE);
      okTextColor.setTextColor(Color.BLACK);
      okTextColor.setBackgroundColor(Color.WHITE);
  } else {
      ...
  }
}
```

US-7

InsertionSort.java

```
private static StringBuilder takeSnapshot(String[] input,
                                          StringBuilder sb,
                                          int highlightIndex) {


   for (int m = 0; m < input.length; m++) {
```

```
        sb.append(m == highlightIndex ? highlightValue(input[m]) : input[m]);
        sb.append(" ");
    }
    sb.append("<br>");

    return sb;
}

private static String highlightValue(String value) {
    return "<b>" + value + "</b>";
}
```

US-9

MainActivity.java

```
public void reset(View view) {
  EditText input = (EditText)findViewById(R.id.input);
  TextView output = (TextView)findViewById(R.id.output);
  input.setText(null);
  output.setText(null);
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:background="@color/colorBackgroundMain"
  android:gravity="end"
  android:layout_height="match_parent"
  android:layout_width="fill_parent"
  android:orientation="vertical"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="teamoc.com.insertionsort.MainActivity" >
  ...
  <!-- Button Bar -->
  <LinearLayout
      android:gravity="end"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:orientation="horizontal" >
      ...
      <!-- Reset Button -->
      <Button
```

```
            android:id="@+id/reset"
            android:background="@color/colorBtnBackground"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:onClick="reset"
            android:text="@string/reset_btn_text"
            android:textSize="@dimen/text_size_main"
            style="@style/Widget.AppCompat.ButtonBar" />
    </LinearLayout>
    ...
</LinearLayout>
```

US-12

app_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- Exit Button -->
    <item
        android:id="@+id/exit"
        android:icon="@android:drawable/ic_menu_close_clear_cancel"
        android:title="@string/exit_btn_text"
        app:showAsAction="always" />

</menu>
```

MainActivity.java

```
package teamoc.com.insertionsort;
...
public class MainActivity extends AppCompatActivity {
    ...
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.app_menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        if (item.getItemId() == R.id.exit) {
            exit();
        }
```

```
        return true;
    }
    ...
    public void exit() {
        android.os.Process.killProcess(android.os.Process.myPid());
        System.exit(1);
    }
    ...
}
```

Testing

US-5

To test this feature, we entered a series of valid and invalid inputs and checked whether the error message would display in each case. All cases were validated successfully.

Valid Inputs	Test Results	Invalid Inputs	Test Results
4 2	No error message	4	Error message
5 6 9 1 4 2 3 4 7	No error message	3 2 5 1 5 6 7 3 2 2	Error message
4 3 9 4 5	No error message	1 0 2 4 3	Error message
9 9 9 9 1 9 9	No error message	7 17 4 2 3	Error message
2 5 3 2 5 7	No error message	9 test 1	Error message
1 2 4 5 1 4 5 6	No error message	R 3 2 11	Error message

Table 7: US-5 results

US-7

To test this feature, we entered the following input from the Homework 1 outline provided in class and compared the result from the application to the expected result. The output of our test case matched the expected result.

Input:

9 8 3 2 4 6 1

Expected Result:

9 **8** 3 2 4 6 1
8 9 **3** 2 4 6 1
3 8 9 **2** 4 6 1
2 3 8 9 **4** 6 1
2 3 4 8 9 **6** 1
2 3 4 6 8 9 **1**
1 2 3 4 6 8 9

Actual Result:

9 **8** 3 2 4 6 1
8 9 **3** 2 4 6 1

3 8 9 **2** 4 6 1
2 3 8 9 **4** 6 1
2 3 4 8 9 **6** 1
2 3 4 6 8 9 **1**
1 2 3 4 6 8 9

US-9

We tested the Reset feature in three major steps. We started with filling in the input field with a string without pressing the Sort button. When we clicked the Reset button, we confirmed that the input field was cleared. In the second step, we filled in the input field with a valid string, and clicked Sort. This resulted in both the input and output fields being filled in. We clicked the Reset button and confirmed that both the input and the output fields were cleared. In the last step, we filled in the input with a valid string and selected Sort. This populated the output field. We manually erased the input field so that only the output field was left. We clicked the Reset button and confirmed that the output was cleared.

US-12

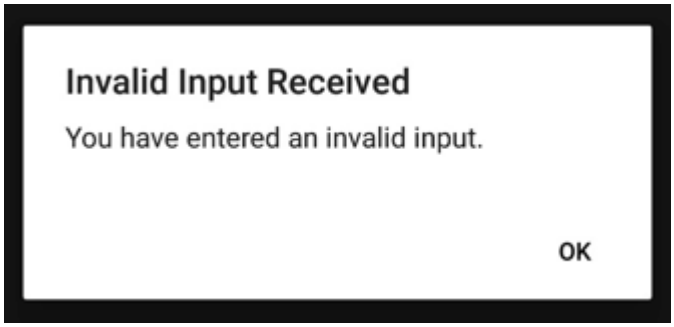
To test the Quit button feature, we used the button after performing different actions in the application. We used the Quit button immediately after opening the application to confirm that the InsertionSort application would quit. We also checked to make sure that the Quit functionality would work after filling in the input field. We also checked to make sure that the Quit button would close the application after selecting Sort. In our final test, we ran 21 sorts with different inputs, left the application running for 15 minutes, and then performed a Quit using the button. Our Quit button passed all of our tests.

Small Releases

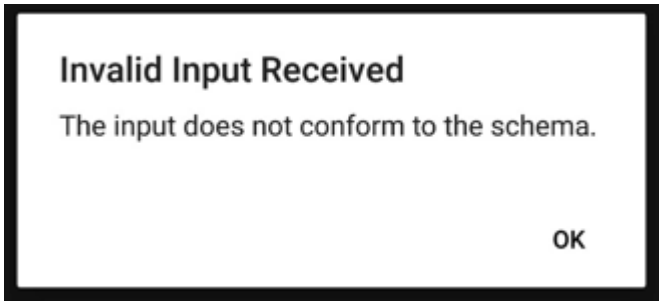
Similar to our process in the first iteration, we released the application to the customers for feedback upon completion of our testing. In this iteration, we implemented error messages to inform users of illegal inputs. While this was a functional requirement of our application, the stakeholders did not include any particular verbiage to use. The Scrum team ended up creating the verbiage to use for the error message.

After using this release, our customers reported to us that they preferred to use their own error message. They sent over a copy of the error message they preferred. We were able to easily replace our error text with the copy the customers sent over. We recompiled the application and sent an updated release to the customer. They approved of the release after updating the error message text.

Before



After

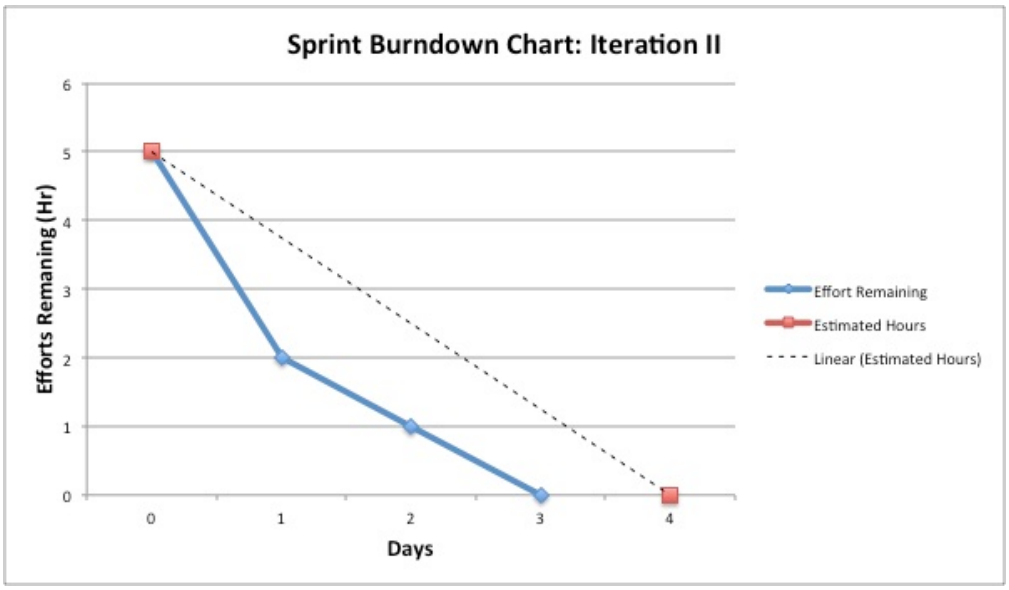


Daily Scrum Meetings and Sprint Review

For the second iteration we plan on improving the user functionality of the application. Our Scrum meeting consisted of dividing up the tasks from our user stories and estimating how much time we would need for each task.

Figure 16: Scrum meeting

This sprint ran from September 12, 2016 to September 15, 2016. Most of the tasks in this sprint were fairly straightforward and so our development team was able to finish all the tasks by day three. We also did pair programming to learn and QA the codes to ensure everything will be good. We did an acceptance test again on the last day and everything ran smoothly.



Graph 2: Sprint Burndown Chart for Iteration II

In this sprint Burndown graph for Iteration II, it shows that we started at 5 hours but were able to finish most of the task by day 3. We spent the a little time on day 4 to do acceptance test.

4.3 Iteration III

User Stories

User Story Code	User Story
US-4	As a user, I want to be able to hit the Return key so that I can run the sort.
US-8	As a user, I want the app to have a question mark icon that will display some sort of help guide so that the app can be explained to users who are unfamiliar with it.
US-10	As a user, I want to be able to generate a sample input so that I do not have to think of my own.
US-11	As a user, I want to be able to exit by typing 'quit' in the input field so that I can exit the application easily.

Iteration Planning

Our process for Iteration III was very similar to the first and second iteration. On September 19, 2016, we held our sprint planning meeting for the third iteration. The meeting consisted of the same individuals as the previous two sprint planning meetings. Upon successfully completing all the tasks in the sprint backlog from the second iteration, the product owner, alongside the development team, moved the remaining priority items from the product backlog onto the sprint backlog. The remaining priority user stories for Iteration III were US-4, US-8, US-10, and US-11. Our sprint goal for the third iteration was to enhance the insertion app features including adding a random button and a help button, adding the return key handler to the input field, and also implementing a secondary method for quitting the application.

Based on our defined sprint goal, alike Iterations I and II, we created sprint tasks out of the remaining sprint backlog priority items and we then estimated the time it will take to complete these tasks For good practice, our team ensured to the best we can that the remaining work assigned will be completed within the sprint timebox of 4 days to complete the product.

Figure 17: moving tasks on sprint backlog

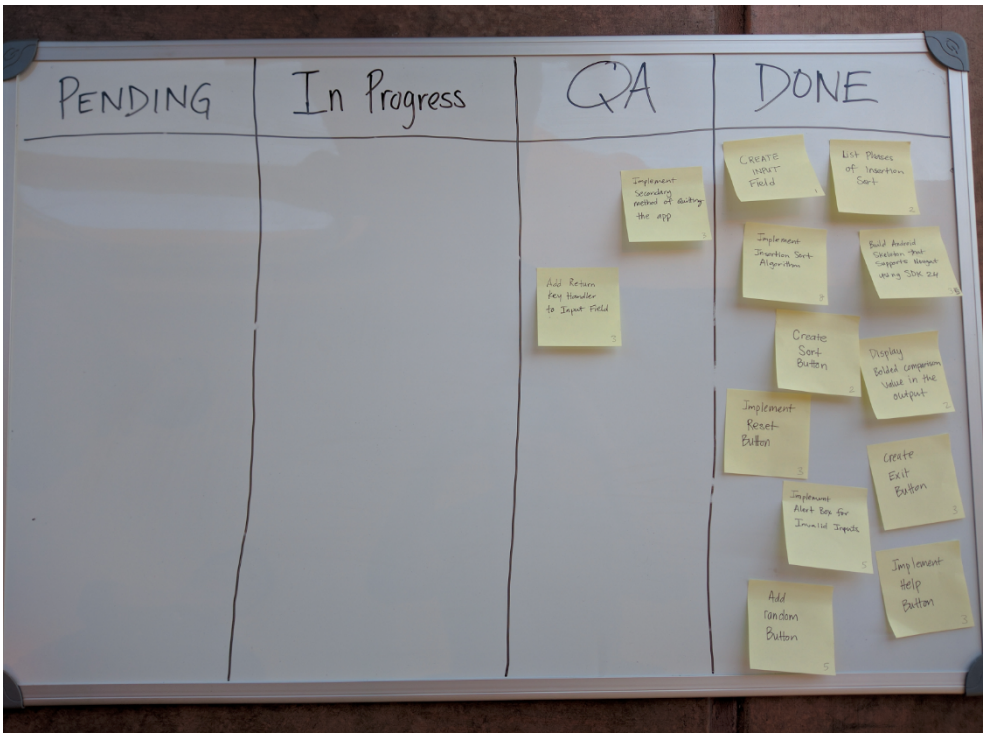


Figure 18: Sprint Backlog for Iteration III

Managing

In the final stretch, we set an agenda, plan for our third iteration and have an open discussion in our development room. Having all the necessary project resources is key for an efficient working environment. We have all 6 team members again working side by side as more testing is involved. We have our sprint backlog in the middle of the development room. As we're going over the backlog we are keeping track of the final tasks needed to be done

Designing

For this third iteration, we chose to include a reset button in the same row as the existing controls since it affected the input field. We also decided to add a help button to the same row as the exit button since it was not linked to the input field. The error and help messages would also need to be updated to support the new features.

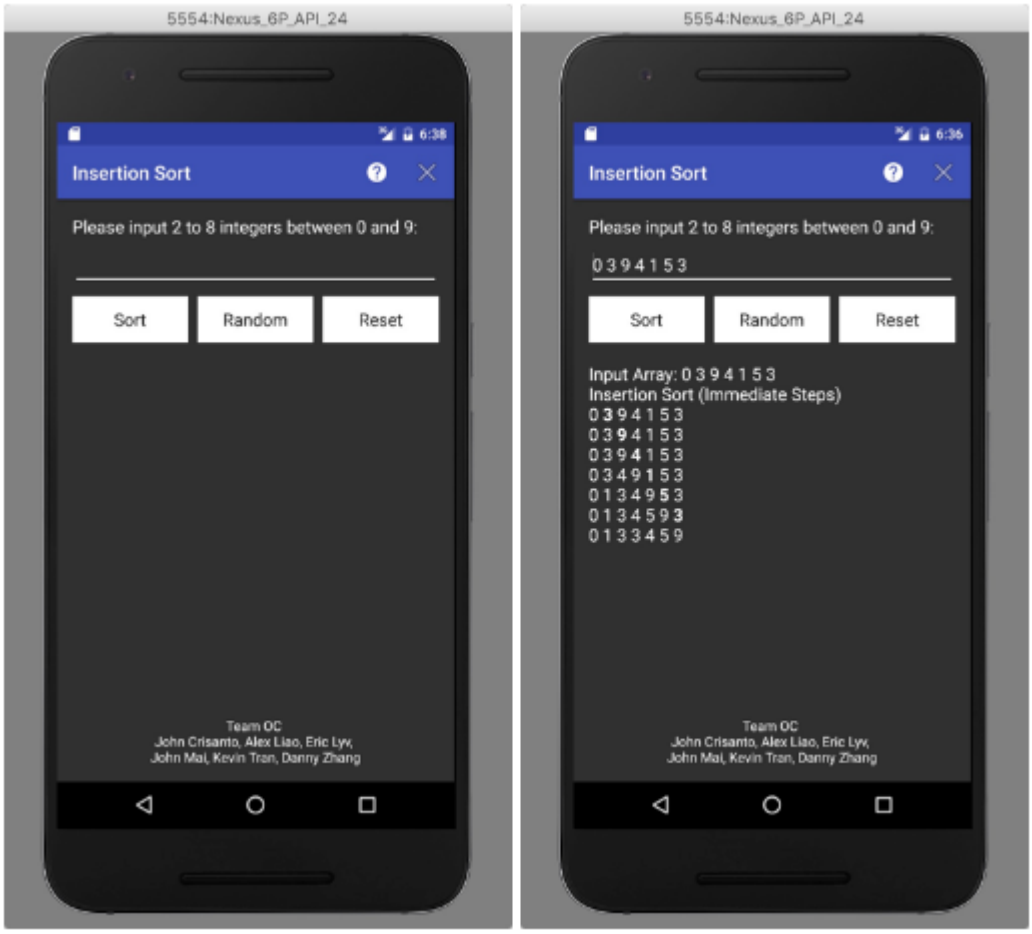


Figure 19: Iteration III design

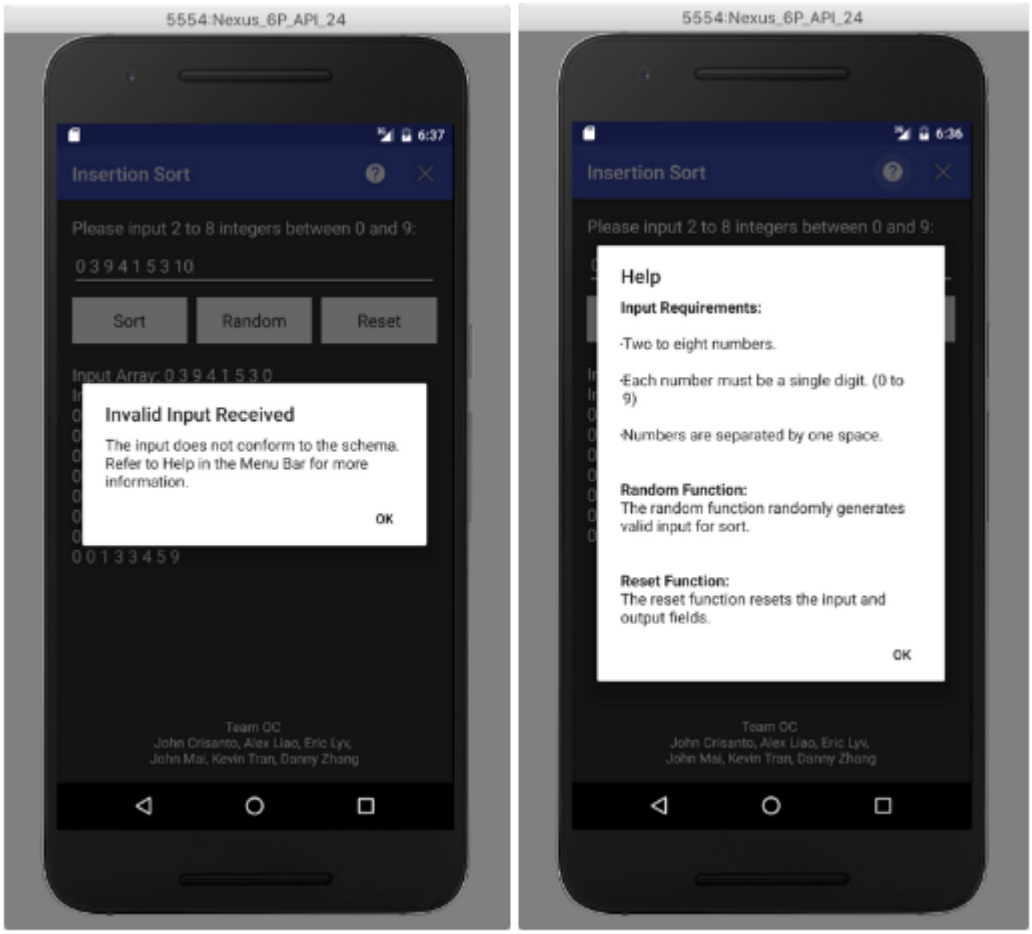


Figure 20: Iteration III design with error and help message

Coding

By the third iteration, we were accustomed to the process flow. We were able to address many questions in the early Scrum meetings. For example, before the coding efforts began, the developers asked the product owner, Alex, about what type of help message he wanted to display for User Story 8. Alex was also a lot more decisive in the way he wanted things to be handled. There was a noticeably shorter wait time before roadblocks were resolved. Furthermore, the developers were better about managing time in their schedules to develop the user stories. It was clear that the Scrum methodology was embraced in Iteration III.

US-4

MainActivity.java

```
package teamoc.com.insertionsort;
...
public class MainActivity extends AppCompatActivity {
    ...
    TextView.OnEditorActionListener inputListener =
                                            new TextView.OnEditorActionListener(){
```

```
        public boolean onEditorAction(TextView inputView,
                                      int actionId,
                                      KeyEvent event) {
            if (actionId == EditorInfo.IME_ACTION_GO ||
               (event != null && event.getAction() == KeyEvent.ACTION_DOWN))
            {
                String inputStr =inputView.getText().toString().toLowerCase().trim();
                sort();
            }
            return true;
        }
    };
    ...
}
```

US-8

MainActivity.java

```
package teamoc.com.insertionsort;
...
public class MainActivity extends AppCompatActivity {
    ...
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        if(item.getItemId() == R.id.help) {
            String outputStr = "<b>" +
                    getResources().getString(R.string.help_line_A1) + "</b>" +
                    "<ol><li>" + getResources().getString(R.string.help_line_A2) +
                    "</li>" + "<li>" +
                    getResources().getString(R.string.help_line_A3) + "</li>" +
                    "<li>" + getResources().getString(R.string.help_line_A4) +
                    "</li></ol>" + "<br>" + "<b>" +
                    getResources().getString(R.string.help_line_B1) +  "</b>" +
                    "<br>" + getResources().getString(R.string.help_line_B2) +
                    "<br>" + "<br>" + "<br>" + "<b>" +
                    getResources().getString(R.string.help_line_C1) + "</b>" +
                    "<br>" + getResources().getString(R.string.help_line_C2);
            AlertDialog.Builder helpAlertBuilder = new AlertDialog.Builder(this);
            if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N)
            {
                helpAlertBuilder.setMessage(Html.fromHtml(outputStr,
                                            Html.FROM_HTML_MODE_LEGACY));
            } else {
                helpAlertBuilder.setMessage(Html.fromHtml(outputStr));
            }
        helpAlertBuilder.setTitle(getResources().getString(R.string.help_btn_text));
        helpAlertBuilder.setPositiveButton(getResources().getString(R.string.ok_text),
                                           null);
            helpAlertBuilder.setCancelable(true);
            AlertDialog helpAlert = helpAlertBuilder.create();
```

```
            helpAlert.show();
            Button okTextColor = elpAlert.getButton(DialogInterface.BUTTON_POSITIVE);
            okTextColor.setTextColor(Color.BLACK);
            okTextColor.setBackgroundColor(Color.WHITE);
        }
        ...
    }
    ...
}
```

app_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- Help Button -->
    <item
        android:id="@+id/help"
        android:icon="@drawable/ic_help_white_24dp"
        android:title="@string/help_btn_text"
        app:showAsAction="always" />
    ...
</menu>
```

strings.xml

```
<resources>
    ...
    <string name="help_btn_text">Help</string>
    <string name="help_line_A1">Input Requirements:</string>
    <string name="help_line_A2">Two to eight numbers.</string>
    <string name="help_line_A3">Each number must be a single digit. (0 to 9)</string>
    <string name="help_line_A4">Numbers are separated by one space.</string>
    <string name="help_line_B1">Random Function:</string>
    <string name="help_line_B2">The random function randomly generates valid input
                                for sort.</string>
    <string name="help_line_C1">Reset Function:</string>
    <string name="help_line_C2">The reset function resets the input and output
                                fields.</string>
    ...
</resources>
```

US-10

MainActivity.java

```
package teamoc.com.insertionsort;
...
public class MainActivity extends AppCompatActivity {
   ...
   public void random(View view) {
       EditText input = (EditText)findViewById(R.id.input);

       int length = -1;
       while(length < 2 || length > 8) {
           length = (int) (Math.random() * 6 + 3);
       }
       String randomStr = new String("");
       for (int i = 0; i < length; i++) {
           int rand = -1;
           while(rand < 0 || rand > 9) {
               rand = (int) (Math.random() * 10);
           }
           randomStr = randomStr + Integer.toString(rand) + " ";
       }
       input.setText(randomStr);
   }
   ...
}
```

US-11

MainActivity.java

```
package teamoc.com.insertionsort;
...
public class MainActivity extends AppCompatActivity {
   TextView.OnEditorActionListener inputListener = new
                                                TextView.OnEditorActionListener() {
       public boolean onEditorAction(TextView inputView,
                                     int actionId,
                                     KeyEvent event) {
           if (actionId == EditorInfo.IME_ACTION_GO ||
               (event != null && event.getAction() == KeyEvent.ACTION_DOWN)) {
               ...
             if(inputStr.equals(getResources().getString(R.string.quit_input_text)))
             {
                 exit();
             }
             else {
                 sort();
             }
```

```
            ...
    };
    ...
}
```

strings.xml

```
<resources>
    ...
    <string name="quit_input_text">quit</string>
    ...
</resources>
```

Testing

US-4

To test this feature, we filled the input field with different strings and hit the Return key. We verified that the Sort method was called and that the output was displayed in the output field. In another test, we selected the input field and selected return without filling in an input. We verified that this case would lead to an error message being displayed in the output field.

US-8

To test the help guide feature, we selected the help icon and confirmed that the help guide pop-up appeared. We tested this feature by using the feature immediately after opening the application, after inserting a string into the input field, and after initiating the Sort. In all of these cases, the help pop-up appeared. We also confirmed that selecting Ok would close the help pop-up.

US-10

To test the random feature, we selected it and confirmed that random input would be filled into the input field. We selected the Random button followed by the Sort button to make sure that the input generated was valid. We also manually checked the input to make sure that the generated input was correct. We used the Random function 25 times to ensure that the integers and length of the strings in the generated input were varied. Each of the generated inputs were valid.

US-11

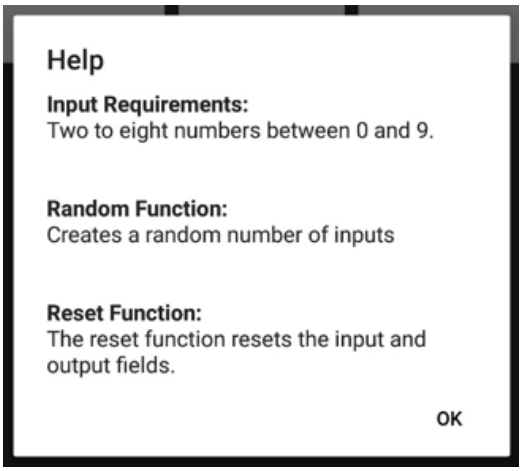
Testing this user story was very straightforward. We typed ‘quit’ into the input using the soft keyboard followed by return to exit the application. We confirmed that the application closed when this sequence was performed. We also ran the application on an Android emulator instance and typed ‘quit’ in the input field using a computer keyboard, followed by return. The application exited, confirming that the feature was implemented correctly.

Small Releases

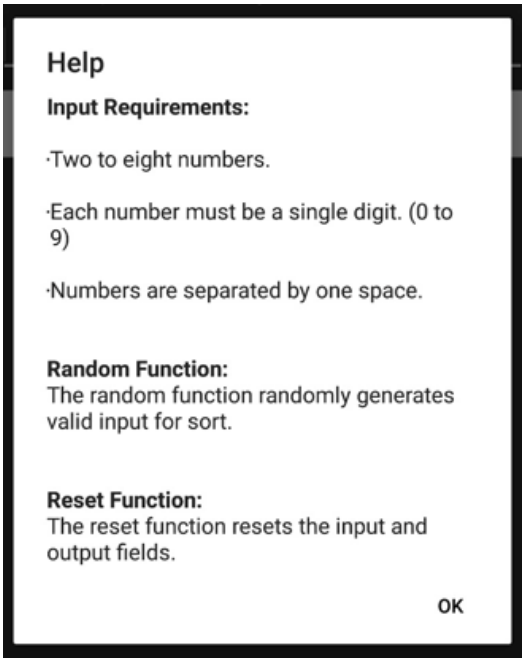
We released our application to customers for feedback once again, after completing our internal testing. Similar to Iteration II, we created the help guide using our own verbiage. We included what we felt was relevant, but we kept in mind that our users may want to use different verbiage.

After releasing the application to our customers, they provided us text that they wanted us to use for the help guide. Upon receiving this text, we updated the help guide per their specification and sent an updated release of Iteration III to the customers. They confirmed the help guide text was correct and accurate and approved the release.

Before



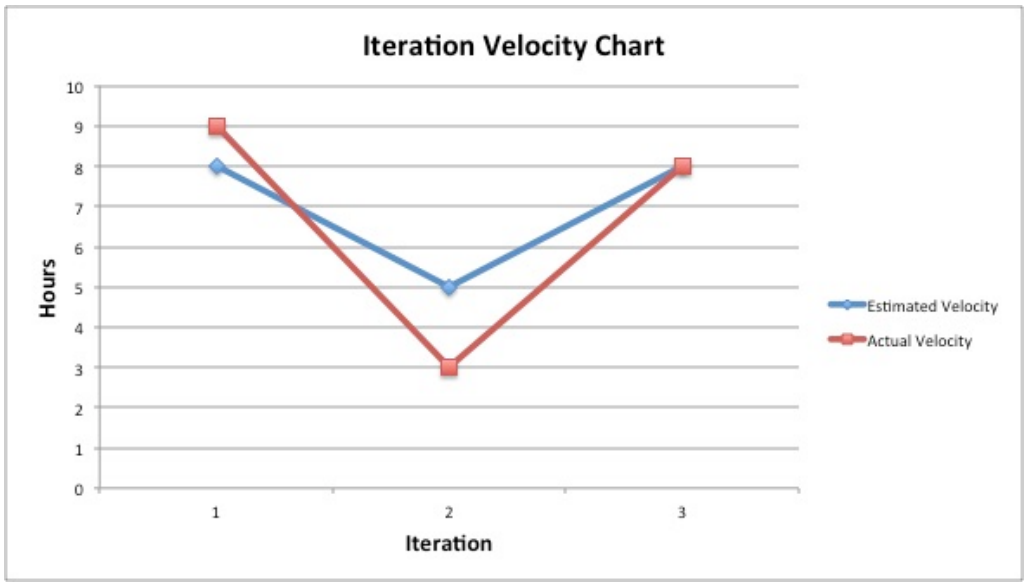
After



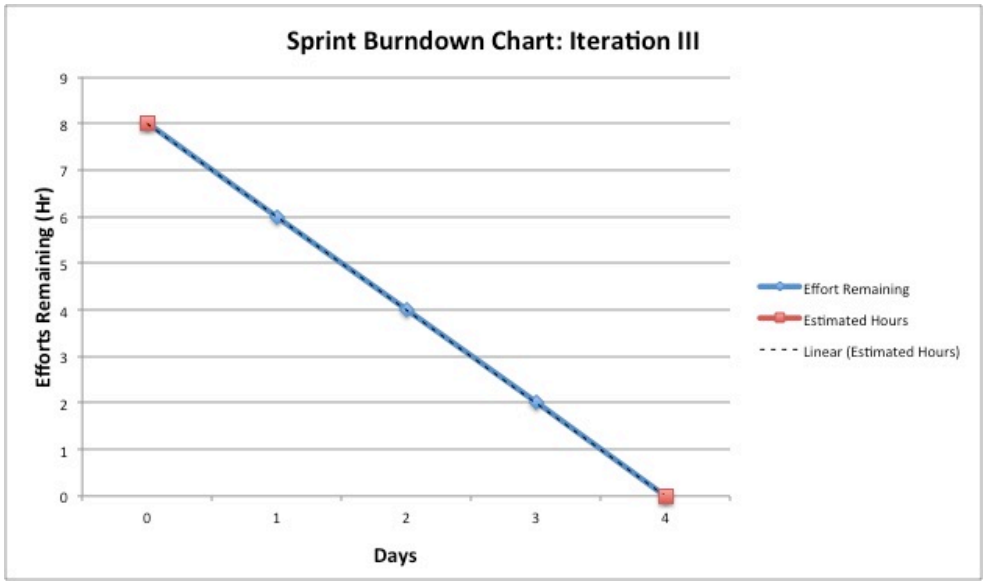
Daily Scrum Meetings and Sprint Review

For our third iteration, during our daily Scrum meeting we discussed enhancements to the application and the changes our user wants. We also discussed how we would present the product to our user once it is finished. We will be doing a 10-minute presentation to show how the app works.

Our third sprint ran from September 19, 2016 to September 22, 2016. Once again this sprint tasks were fairly easy to implement so we did not run into any issues. We made some finishing touches to application and tested it once again on the Nexus 6P emulator. We also discussed our iteration velocity graph and our sprint burndown graph.



Graph 3: Iteration Velocity Chart



Graph 4: Sprint Burndown Chart Iteration III

In this sprint burndown chart, it shows that we assigned 8 hours to our sprint. We were able to tackle our tasks each day with minimal effort. We put in the finishing touches and were able to complete the app by the end of the sprint.

4.4 Acceptance Testing

We conducted phases of acceptance testing after each iteration to make sure we addressed our user stories and satisfied the user requirements. In each acceptance testing phase, we pulled up the relevant user stories for the iteration. Each member of our team ran the application and ran tests to emulate the user stories. The detailed tests we ran for each story can be found under the “Testing” section of each iteration.

By having each member run through and emulate each user story, we were able to not only test our work once, but multiple times. Our process of acceptance testing allowed for comprehensive testing to be conducted each iteration.

Upon completion of our third iteration, we ran acceptance tests for the user stories that were assigned to Iteration III. Once each member confirmed that each Iteration III user story was satisfied, we proceeded to run through acceptance testing for the entire application at that point. Each member ran acceptance tests on their own and ensured that every user story in our project was satisfied. After this final phase of acceptance testing, we determined the application had satisfied all the user requirements and user stories. With the final phase of acceptance testing complete, we deemed the application approved for delivery.

5.0 Release

5.1 Documentation

`MainActivity.java`

This class begins immediately when the application is opened. It is the main class where the majority of the actions are handled.

```
protected void onCreate(Bundle savedInstanceState);
```

The onCreate method initializes the Activity. When the Activity is started and the application is not loaded, then onCreate is called automatically. It creates input field, input listener, sort button, sort button listener, and the output display field.

```
public boolean onCreateOptionsMenu(Menu menu);
```

This method creates the menu containing the help and exit buttons.

```
public boolean onOptionsItemSelected(MenuItem item);
```

This method is called whenever an item in the options menu is selected. When the help item is selected, the help description pops up. When the exit item is selected, the exit function is called, exiting the application.

```
public void sort();
```

This method checks the input to see if it is in the correct format. If the input is not in the correct format, it will display an error message. If the format is acceptable, it calls the sortWithOutput method in the InsertionSort.java class. The output is displayed in the output text area.

```
public void exit();
```

This method kills the InsertionSortApp process, exiting the application.

```
public void random(View view);
```

This method generates random input into the input field that is recognized by the sort method.

```
public void reset(View view);
```

This method clears the input field and the output field.

`InsertionSort.java`

This class contains the methods used to perform an insertion sort and outputs the result along with snapshots of each sorting iteration. It also provides the option to highlight (bold) the comparison value.

```
protected static String sortWithOutput(String[] input);
```

This method sorts an array of integers (as strings) using an Insertion Sort algorithm and returns a formatted output of the sorting phases and the sorted result.

```
private static StringBuilder takeSnapshot(String[] input, StringBuilder sb, int
highlightIndex);
```

This method appends each integer within an array to the resulting output. It will also add a bold styling around an integer within the array at a given index.

```
private static String highlightValue(String value);
```
This method adds a bold HTML style to an integer value and returns it.

`activity_main.xml`
This xml file properties for the layout of the application. It contains information for the input label, input field, button bar, string output, and credits.

`app_menu.xml`
This xml file contains the menu options and the features of each option.

`colors.xml`
This xml file contains the colors used in the application.

`strings.xml`
This xml file contains the hard-coded strings in the application.

5.2 Training

After showing our user the Insertion Sort App, we gave them a run through of how to use the application. We started by holding a meeting with each other to plan how we want the training session to go, what we want the users to know about the application, and what kind of goals we have for this training session. The training session should be able to teach users how to properly navigate all of the application's functions. Once we ironed out a training game plan, we held a training session for our users to attend. During the training, we shared a video that had a demonstration of all the application functions. After the video demonstration, we proceeded to manually show the users how the application works. The goal of the training session was to guide our users through the Insertion Sort App and teach them how to sort the numbers and successfully exit the application.

5.3 Operating

To operate the Insertion Sort App, you unlock your Android device and navigate to the icon. Click on the icon to start the application. The user interface will display upon opening the application:

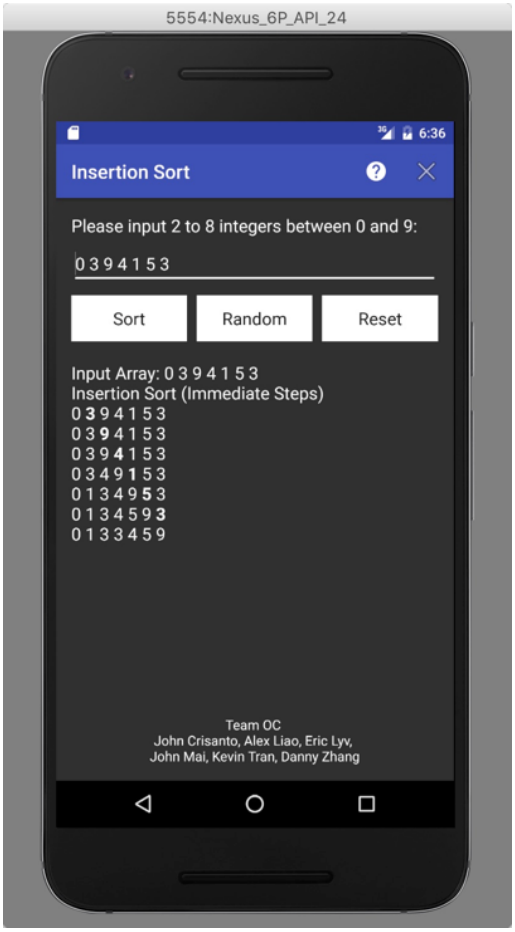


Figure 21: final product user interface

Insertion Sort

In order to run an insertion sort, you must enter integers into the input field. You may enter integers from 0 to 9, separating each integer in the input field with a space. You must enter a minimum of two integers and may not enter more than eight integers. Once you have entered a valid input, press the "Sort" button to run the insertion sort.

The insertion sort will run and each iteration of the sort will be displayed within the application. If an error message popped up after pressing "Sort", you entered an invalid input. You must correct your input before the application will properly sort your input.

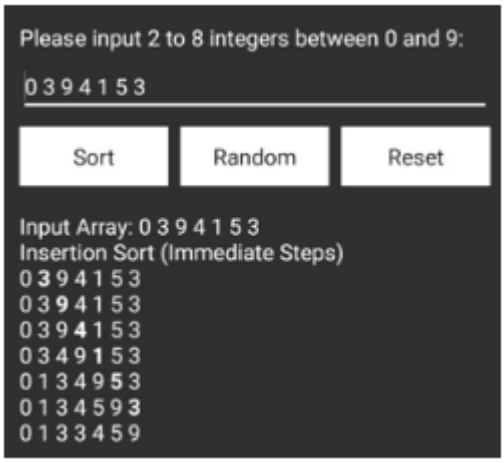


Figure 22: final product insertion sort

Help Screen

The help screen can be accessed by pressing the question mark icon in the upper right side of the application UI. When pressed, a help pop-up will appear on the screen telling the user how to operate the application. Press the “OK” button to close the popup and return to where you were previously.

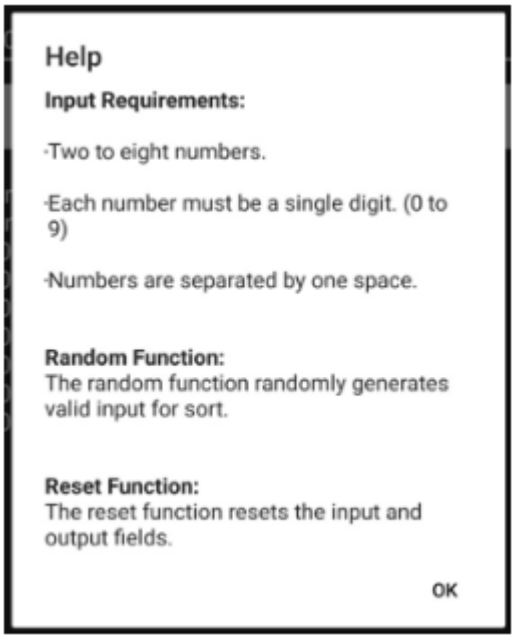


Figure 23: final product help screen

Exit

To exit the application, you may click on the white X icon on the upper right side of the application, next to the help icon. Additionally, you may type “quit”, without quotation marks, into the input field and the application will close.

Figure 24: final product exit button

5.4 Feedback

We welcomed user feedback after delivering the application to our customers. Overall the feedback we received was positive. The users reported that the application was quick to open, straightforward to operate, simple to use, and free of bugs. There were no reported crashes. Our customers were not only satisfied with the performance and ease of use that our application delivered, but they made comments on their positive interactions with us. We received positive feedback for our entire process as well. They were impressed with our communication and the fact that we kept them informed throughout the process. They particularly enjoyed our small releases that came with each iteration. The users felt that testing was very efficient and kept the project on the right track.

6.0 Other Activities

6.1 Estimation

Estimation helps give us an idea of how much time and resources we would need to allocate for such a project. We came across estimation when we had to create our product backlog. During the product backlog creation, we had to assign an estimated time to each task. This gave us an idea of how much time we would need to complete the given task and the amount of resources we would need to allocate to the sprint planning. Further identification of architectural spikes and high level risks for each task led us to give a more accurate estimation of the tasks. Estimation is helpful to use as a baseline so that we can easily measure our progress throughout the project. Furthermore, we can see if there is too much work given to a sprint based on the numbers that we have estimated.

In *Managing the Software Process*, the book mentioned an estimation procedure that would help give accurate estimations by the group. The estimation procedure is called Wideband Delphi Technique. This procedure allows a group of experts to come together to discuss the product being build. After the discussion, each person from the group will get an estimation form where they anonymously write down their estimation. All estimations will be anonymous except for their own estimation. Further discussions and revisions are made after everyone sees the results. The procedure ends when all estimations are agreed upon. The Wideband Delphi Technique helps narrow down confusion and provides a more accurate project estimation.

Although we did not follow the Wideband Delphi Technique, we went through a voting process during our product backlog time estimation task. In our process, we rated each task based on how long we thought it would take for us to finish it. The average of the votes was taken and assigned to the corresponding task.

The book also mentioned possible estimation inaccuracies that could occur during the estimation process. There were definitely common estimation inaccuracies that had occurred throughout the course of our project. Some of the time and resources that we had estimated for the task turned out to take much longer or shorter than expected. There are two types of estimating inaccuracies: normal statistical variations and estimating bias.

In normal statistical variation, estimations are usually too low. This causes last minute crisis to happen, late delivery of the product, and poor quality product as well [13]. In order to avert this issue from happening, having multiple estimators will help.

In the second type of estimation inaccuracies, estimating bias, team members often estimate the project way too early in the project stage. This causes members to lack information or evidence to support their estimation. The earlier the estimation, the less is known about the product, which means there is a higher estimation inaccuracy [13]. To avert this issue, one would need to collect all user stories and information before making an educated estimation on the product.

When estimating the production of the product, there should always be some kind of estimating contingencies. These contingencies refer to setting aside a budget and resources that would help cover unforeseen circumstances that arise during the project. While most programmers will argue that this will not be necessary since they believe they know how much time and resources should go into the coding, it would still be ideal to have some sort of insurance in case any issues do arise. Since looking out for contingency errors can increase cost, time and resources, following the Agile process can alleviate some of these worries. The Agile process allows us to prioritize important tasks that need to be completed first in the sprint. If any issues show up, it will at least be early enough in the process to be able to have time to fix right away.

Overall, estimation is good to have so that we can monitor how much time and resources we would need during each sprint. This allows for projects to run more efficiently. Without estimation, we would not be able to deliver a quality product in a timely manner.

6.2 Project Planning

It was necessary for the team to do some project planning in order to determine what was asked of us. The team found that investing time into project planning to define what our work will be and how it will be done was well worth our time. In our project plan, we defined each of the major tasks, estimated the time and resources required of each task, and created a framework for the team to review and control the project progression. With the completed project plan, the team was able to utilize it as a point of reference to compare our actual performance with the expected performance.

The team followed the iterative plan negotiation process to develop our plan. We started with negotiating the initial requirements of the project. It took us some time to deliberate which features of the application were critical. In our project, we decided that one of the essential functions for the application was that it would run on the Android operating system. We also found that other essential functions for the application would be to have an input field for the values, a button to run the insertion sort, and a display of each sorting step. Once these initial requirements were determined, we developed a conceptual design of the application and decomposed the work into key elements in a Work Breakdown Structure. During this step, we broke each initial requirement into smaller product elements so that we could estimate the amount of time and resources that would be needed to accomplish each task. After estimating the time and resources needed for each task, we were able to develop a plan using user stories. The product owner created the user stories based on his needs. He then had a meeting with the team to go over the user stories and to break them up into smaller tasks. The smaller tasks were used for planning what we wanted each iteration to contain. By having all of the tasks and the estimated number of work hours for each of them before us, we were able to spread the workload evenly across three iterations. We used this information to create a sprint backlog and a sprint burndown chart to track the team's progress. The backlog and burndown chart were valuable references at the end of the project, when we needed to see how we could plan better for the next project.

In the textbook, Humphrey states that there are five essential elements in a project plan. These five elements are: goals and objectives, Work Breakdown Structure, product size estimates, resource estimates, and project schedule.

The project's goals and objectives were decided upon by the product owner during the requirements phase. It was agreed upon by both the product owner and the team that the goal of the project was to create an Android Application that shows the insertion sort. The product owner created the user stories that the team used to develop the application. The team worked with the product owner to prioritize the objectives, or in our case it was user stories. The team developed the application through each iteration based on the determined priorities. During the iterations, issues and roadblocks regarding objectives were addressed by following the Scrum and XP methodologies. All changes made to the requirements were tracked and documented.

The Work Breakdown Structure was used to estimate the size of the project that we were to produce. We started by breaking down the product into smaller work elements. In our project, we created a product backlog that accomplished the same purposes of a Work Breakdown structure. Our product backlog contained all of the tasks that needed to be accomplished for the project to be in a deliverable state.

Product size estimates for our project were done when we broke down the user stories into smaller tasks with the product owner. Some typical ways to measure the size of the software are lines of code estimates and function points. In our project, we felt that estimating in hours was appropriate, which is very similar to estimating by function points.

In our project's resource estimate, we had to utilize the skills of our team. We did not have any other options for resources, so every skill that was needed had to be developed within. The team did not have enough experience to gauge the amount of resources we would need to complete the project. Even if we did have that ability, it would not have made much of a difference because the teams were already set by the time we got to estimating.

The project schedule is developed by dispersing resources over the development phases. The team's product owner had the responsibility to set the project schedule. He accomplished this by determining the length of each iteration and by selecting which user stories would be assigned to each iteration. While the product owner did not have any historical experience with similar projects, he used publicly available factors as guidance for setting the schedule.

6.3 Project Monitoring and Control

The combination of project monitoring and control is an effective way to ensure that projects can proceed as planned, and that change requests are clear, appropriate, and agreed-upon by all members.

Project monitoring refers to the use of methods, such as checkpoints and reviews, to strategically monitor the current status of active tasks. Once a project has grown in complexity, it becomes much harder to maintain. To ensure that all aspects of a project continue to run smoothly and deadlines are

met, it is important to monitor both the project tasks and resources. Project monitoring also ensures that the appropriate resources are assigned and quickly adjusted as needed. In terms of communication, monitoring provides the development team opportunities to alert the stakeholders to any potential delays ahead of time.

One such method of project monitoring is known as earned-value project scheduling. In this method, there are milestones that must be achieved within a given timeline. Each milestone acts as a checkpoint for the current stage of the project. When a point in the timeline is reached, the team or individual monitoring the project status is able to compare the completion of the defined milestones to the expected progress. In our project, we created tasks from user stories and assigned them to various iterations. For each task, we created a series of acceptance tests that had to pass in order for the iteration to be considered complete. While doing this, we also created estimates for the completion of each task and deadlines to complete the iterations. These iteration deadlines served as our checkpoints and allowed us to maintain progress towards completing the project. Burndown charts were also used to keep track of our checkpoints and the rate of progress towards those deadlines.

In addition to earned-value project scheduling, another method for project monitoring is a quarterly review. These reviews serve as “a forum for conflict resolution and progress monitoring against period and product objectives”. Since we elected to follow an Agile process, we held weekly Scrum meetings that allowed us to keep track of each task’s status during the iterations. During these meetings, we held discussions about potential obstacles that could impede our progress and determined what steps were required to overcome these obstacles. We also employed the use of a Scrum board to monitor the state of each task. The Scrum board included tasks that were in the backlog, in testing, in progress, and listed as completed. Using this tool made it easy to tell at a glance what tasks were being worked on and what tasks were still pending.

Project control is the process of maintaining a project’s velocity by carefully considering changes in requirements and where they should be implemented. As projects progress forward, there are inevitable requests and modifications that will come up. These must be integrated and managed properly to prevent the disruption of a project’s flow. Adding additional requirements to a project has the potential to extend the project’s duration and can create chaos in the process. Therefore, change management is necessary to ensure that projects do not deviate from a given plan. We used a sprint backlog to control change requests and to ensure that they did not affect the current iteration’s velocity unless we determined the changes to be in scope. During our Scrum meetings, we held discussions about whether to implement these backlog items in the current iteration or in future iterations. By making careful considerations, we were able to prevent scope creep and minimize disruptions to the current development progress. This also allowed us to develop the product efficiently and meet our planned deadlines.

6.4 Configuration Management

Change management is an essential part of good software engineering practices. When managing changes in the ever evolving software, Software Configuration Management is needed. Software

Configuration Management helps us reduce software errors by tracking and controlling changes made to the requirements that would affect the overall product source code. During development, problems can occur and occasionally team members may run into conflicts with one another. For example, when two programmers are working on the same project, one programmer might work on a prior version of a program and makes changes over another programmer's work in a simultaneous update. This may result in bugs in the final release. In addition, there might be disagreements within common code, shared code and errors that need to be fixed in future versions of the program [13]. Therefore, we need to come up with a management solution that would handle all of these problems from happening.

Our solution to Software Configuration Management was to use GitLab. GitLab is an online application that allows users to upload their projects and deploy code collaboratively. We decided to use Gitlab because it provides versioning management and helps us keep track of any changes that are made to the software. Another reason why we use GitLab is code control. By using GitLab as part of our SCM, we are able to organize and build a private code repository. With pull requests, it makes it easier to manage our code while allowing us to see the current changes to the branch. All of our team members have access to the latest code base. Members collaboratively create their own branches locally as working copies, and then push back to the development branch as the code is completed. Members can provide inline comments within the branch to inform everyone of any changes made.

Each member of our team can pull, push, and commit changes using GitLab. But before committing code to the master branch, we conduct multiple tests. According to Humphrey, there are 4 tests that are needed in the SCM implementation process: unit test, integration test, functional test and regression test. Unit test is a separate test of each individual module. Integration test which tests the interfaces and interdependencies of the modules as they are integrated to the system. Function test ensures functionality of each successful build. Regression test tests the system to ensure that each build hasn't regressed or lost functions after each integration [13]. Running our code through these tests will ultimately ensure changes are properly implemented.

Software Configuration Management code management requires the team to establish the baseline. We consider the first successful product as the baseline [13]. The baseline is a milestone in the software development process. After successfully delivering our product with all the basic functions (US-1, US-2, US-3, US-6) in the first iteration, we set that project as our baseline. We committed project codes to our repository master branch. From then on, we add only tested and approved codes to the (master branch) baseline while being protected [13]. We protect any changes made the could affect the source code. If we manage changes before they are made, and record changes before they are implemented, we can improve quality of the final product and reduce future errors. By managing our code in this way, we keep our baseline pristine. The baseline is frequently updated after each product enhancement and it serves as the official source in which all subsequent work is based on [13]. Developers must use the official source so that all work is consistent with everyone on the team.

As development continues to grow in size, the need for an automated system is a must. It is extremely difficult to remember what is done, what is updated, what is taken out, by whom and when during software development. In final software tests, valuable time can be lost in a frantic search to correct

unaccountable last-minute regression [13]. GitLab is our main automated tool that we use to keep track of all these important details. GitLab keeps track of our change request, CCB actions, activity log entries and reports.

There are six important tasks in Software Configuration Management: configuration control, change management, revisions, versions, deltas, and conditional code. Configuration control is very important. We only have one official copy of the project files and keep a separate official copy after each revision. It is the simplest way to protect system revisions [13]. Next task, change management; we keep changes separate by assuming that the separate copies are different and only keep one single official copy [13]. Revisions is our third task; we keep track of revisions by forming a numbering system to identify each module, test, component, product, and system [13]. There are potential issues with module changes so we must set versions; create different versions if they have different functions, even if they are implemented by the same module [13]. The fifth important task in the SCM is implementing deltas; because the use of versions in the prior task introduces multiple copies of the same code, we must implement deltas to store the base module together with the revisions required and to make it into another version [13]. Lastly, the use of conditional code is to handle slight variations between modules. The use of conditional code has the advantage of providing one complete copy of the entire program [13].

Furthermore, there should be an established module ownership. In our project, we chose Kevin Tran and Danny Zhang to be our module owners because they are the most experienced programmers within the group. The Product owner, Alex Liao has to ensure that every member in the team have followed the SCM process in keeping the module's integrity.

6.5 Quality Assurance

The goal of quality assurance (QA) is to ensure that the established standards and procedures for a software process are being followed [13]. QA works closely with the development team to ensure that the defined software process is being followed. As its name suggests, QA assures quality. A QA team would not actually produce or design any product of quality; this responsibility would fall on the developers. However, QA monitors the development process to assure that the software process is being followed, which would ideally result in a quality product being developed.

QA is involved in many different roles of the overall project. In the beginning of a project, QA must make sure quality practices are planned. This is essential to the overall project. In order to have a quality product, you must have a high quality base to build upon. If quality practices are not planned in the beginning of a project, the final product will likely not be a high quality product. Once a quality software process is established, QA should review the requirements for a project. These requirements need to be reviewed in order to make sure they are quality requirements. Low quality requirements are also unlikely to result in high quality products [13]. Upon reviewing the requirements, QA needs to make sure the proper coding and development processes are being followed. QA should ensure that the development team is following the defined process, as deviating from a defined process can result in low quality work and delays in order to address and fix the low quality work produced [13].

Having QA so deeply integrated within the process means they need to work with (and not against) development. Because QA monitors the overall development of the product to ensure that a defined process is executed and carried out throughout the life of the project, they need to be viewed in a neutral light. If they are viewed in a negative manner, QA cannot be effective [13]. To help avoid this issue, QA should not report to the project manager. Instead, they should report to other management individuals who have a vested interest in software quality [13]. No matter who QA reports to, it is essential that they have the backing of upper management. If they have no managerial backing, they have no ability to enforce a process or to have development adhere to a process. Without the proper backing, QA has no authority and is ultimately ineffective.

Throughout our project, we made sure that QA was involved during each applicable phase. In development, we utilized pair programming to ensure that we were adhering to standards and processes we had defined. One person would write code and the other would review the code to make sure the code was written in a way that adhered to our standards. As tasks in our backlog were completed, they were passed to QA for review. Prior to being marked as completed, QA made sure that our process was adhered to and was consistent. Once it was approved by QA, we marked a task as complete. We also involved QA in testing phases. As we discussed testing and created tests to use, we included QA as well. This was done in order to make ensure that our tests were actually testing the right thing and to make sure our results were recorded properly. If action was needed as a result of a test, we needed to make sure that the proper action was taken.

6.6 Risk Management

Risks are uncertainties that can potentially affect the overall project outcome. It is important to understand the risk environment, the project context, and project objectives clearly. The risk environment establishes the acceptable levels of risk within the project and the project context points out the potential contributors to risk and how to interpret them [12]. Carefully studying the risk environment and the project context can help ensure that activities deemed necessary to achieve the project objectives are not impeded.

It is common that while the Scrum team works on their respective tasks during each sprint, they know the importance of each task and how to complete them, however, they might not be able to express the implications of all the tasks in the context of risk management [12]. This is why it is very important to observe and make an attempt to recognize all possible threats, opportunities, and contingencies throughout the entire process of building a product. Identifying the potential risks ahead of time is very useful so that the team can either prevent them, if possible, or prepare in advance in the event that these risks do turn up. The figure below provides an overview of the Agile risk management process.

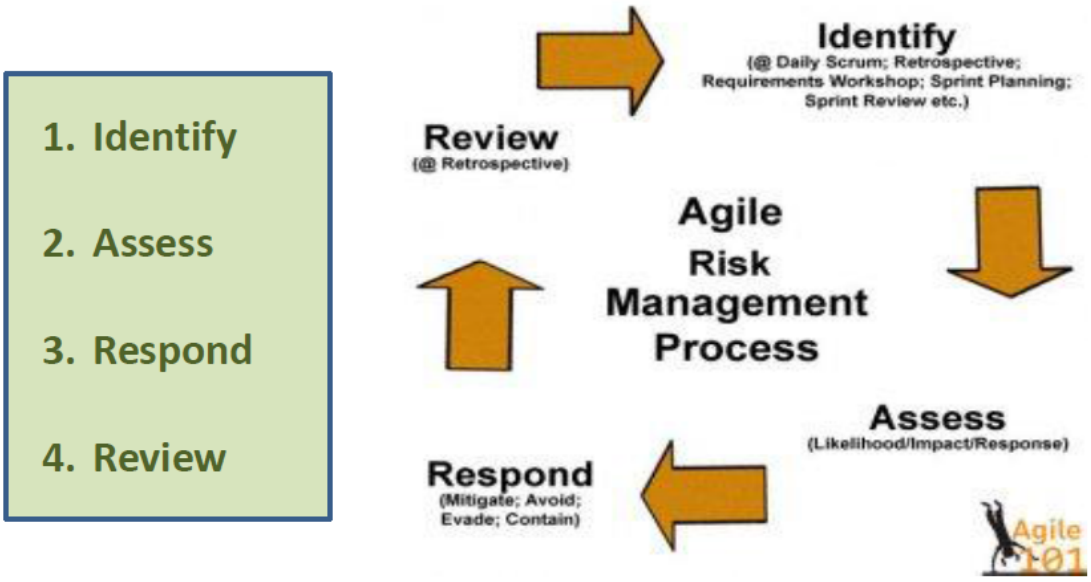


Figure 25: overview of the Agile risk management process

Our group followed the risk management process shown in the figure above. As part of the identification phase, our team identified certain risks in our architectural spikes. The first risk that we have identified was related to validation of user input because wrong inputs can cause the program to crash if not handled correctly, and our group had never dealt with input validation when it comes to writing an Android application. The second risk we identified was related to satisfying the need to highlight the integer that is being compared when running the sort. This was considered a risk because we were unaware whether or not bolding individual characters within a string object was allowed in Android programming. Once both of these risks were identified, we then moved onto assessing them.

We performed a risk analysis and assessed the probability of these risks occurring and the impact they may have on our team's capability of reaching our project goals. Assessing both of these risks as a group, the risk with the highest priority pertains to input validation. If this was not handled properly, the entire program crashes due to a NumberFormatException. On the other hand, the risk connected to highlighting a part of a string will not necessarily cause the program to crash since the sort will still execute, the user just will not see it. As far as the probability of the risks go, both were likely to happen so preparation was highly necessary to tackle these possibilities. Our group then moved onto responding to these assessments.

In order to help prepare and mitigate these risks, the first thing we wanted to do was create a message to the user and notify the user to input numbers from 0-9 and to choose at least 2 numbers to a maximum of eight. Although this will help guide the user to enter the correct input type, it does not stop the user from entering other inputs such as special characters, letters, etc. The team started doing more research on Android APIs related to string parsing functions and regular expressions while also adding exception handling into the mix to ensure that only valid inputs will be accepted. The same approach of doing intensive research was performed to help mitigate the risk related to bolding individual characters

in a string. As part of risk management, it is crucial to monitor and review risks even after they are mitigated and resolved. Upon going through the later stages in development such as improving user functionality and enhancing the app features, we had to ensure that these errors will not occur and the app still functions as intended because sometimes addition of new functions can cause errors in other parts of the application.

Risk management is very important in building a successful product. Recognizing risks in advance and designing the appropriate steps needed to address them are crucial in order to promote a balanced ratio of risk and reward within the confines of the project [12]. However, it is extremely important to keep in mind that in order to properly implement an effective risk management process, communication between everyone involved in the entire project is key.

8.0 References

[1] http://www.extremeprogramming.org/rules.html

[2]https://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming

[3] http://agilemethodology.org/

[4] https://developer.android.com/training/basics/firstapp/creating-project.html

[5] https://developer.android.com/training/basics/firstapp/running-app.html

[6] https://developer.android.com/develop/index.html

[7] https://developer.android.com/reference/classes.html

[8] https://gitlab.com

[9] http://www.scrum-institute.org/The_Scrum_Product_Backlog.php

[10] http://www.scrum-institute.org/The_Sprint_Backlog.php

[11] http://www.scrum-institute.org/Burndown_Chart.php

[12] http://institute.agileriskmanagement.org/wp-content/themes/iarm/publications/AgileRiskManagementScrumWhitepaper.pdf

[13] Humphrey, Watts S., Managing the Software Process, The SEI Series in Software Engineering, Addison-Wesley, 1989. (29th Printing, May 2003) (ISBN 0- 201-18095-2)