Reflections on the use of Agile practices and associated tools in university settings for an Android project

Elijah Zolduoarrati
Information Science, University of Otago, Dunedin, New Zealand
zolduoarrati@gmail.com

Adriaan Lotter
Information Science, University of Otago, Dunedin, New Zealand
adriaanlotter@gmail.com

Kevin Michael
Information Science, University of Otago, Dunedin, New Zealand
Kevin.vncnt@gmail.com

Rohullah Mohammadi
Information Science, University of Otago, Dunedin, New Zealand
rohullahm313@gmail.com

Nick Alessi
Information Science, University of Otago, Dunedin, New Zealand
n.c.alessi@gmail.com

Sherlock A. Licorish
Information Science, University of Otago, Dunedin, New Zealand
sherlock.licorish@otago.ac.nz

ABSTRACT

Agile methods such as Scrum are held to improve the management of software projects. In particular, there is a popular view that such methods, if introduced properly at the university level, could translate into both satisfied and highly skilled future developers. This paper reflects on the perceived effectiveness of implementing recommended Agile software development practices and tools through the use of Scrum for developing an Android app as part of a university semester-long course. The app captures, stores, uploads, retrieves, shares and manages digital receipts, in overcoming the problems associated with misplacement, storing and organization for manual receipts. Evidence in our reflections demonstrates that implementing Agile software development practices and tools for a university project significantly contribute to project success and quality. Our outcomes provide lessons both for the mentoring of students in the use of Agile practices, and for novice developers using Agile methods in real projects.

Keywords: Agile methods, Android, Project management, Scrum, Reflections, University project.

1. INTRODUCTION

Software project management involves planning and leading software development projects to success. Major areas that have been found to affect project success, and thus, require careful consideration during software development project implementation include: Requirements Engineering, Effort Estimation, Project Planning and Scheduling, Prioritization, Risk and Issue Management (Varajão, Dominguez, Ribeiro, & Paiva, 2014; Nguyen, 2006; Boehm, 1991). The specific flavor of software project management implemented is linked to the methodology that is adopted, which in turn is linked to the nature of the software under development. Software development outcomes may also be derived taking into account the trade-off between available budget (and the availability of time) and quality. It is for this reason that software project management is crucial, as it may help to maximize the quality of software development outcomes within a specified budget, and ultimately, aid with producing software that satisfies the client.

Agile methods, in terms of project management, attempt to manage projects by applying a nimble and fluid methodology (Coram & Bohner, 2005). It is known that software requirements are susceptible to frequent change, and therefore, the process of managing the development of (Agile) systems often needs to inherit the characteristics and tools recommended by fluid methodologies. By using Agile methodologies, teams are able to manage the complexity of the development process through the use of an iterative approach (Beck, 1999), which in turn leads to incremental software delivery, project-wide flexibility, and an emphasis on working code (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003).

However, there is often a necessity for project managers to evaluate the applicability of Agile methods for software development projects, both in terms of the nature of projects and the people involved (Phillips, 2004). In this way the strengths and weaknesses of Agile methods can be evaluated during managerial decision making, with the intention of making informed decisions around which practice to use and which to avoid. As highlighted by Phillips (2004), the applicability of Agile methods is greater for projects that have the following characteristics: requirements are volatile, the software size is small to medium, and the project is not exceedingly complex. In particular, the Agile method, Scrum, aims to improve software development in an environment undergoing constant change (Schwaber & Beedle, 2002). The Scrum life cycle itself constitutes three phases—namely, pre-development, development, and post-development. The pre-development phase focuses on planning and architectural activities; the development phase involves iterative development cycles ("Sprints"); and finally, the post-development phase involves the finalization of the project and the final delivery of the system.

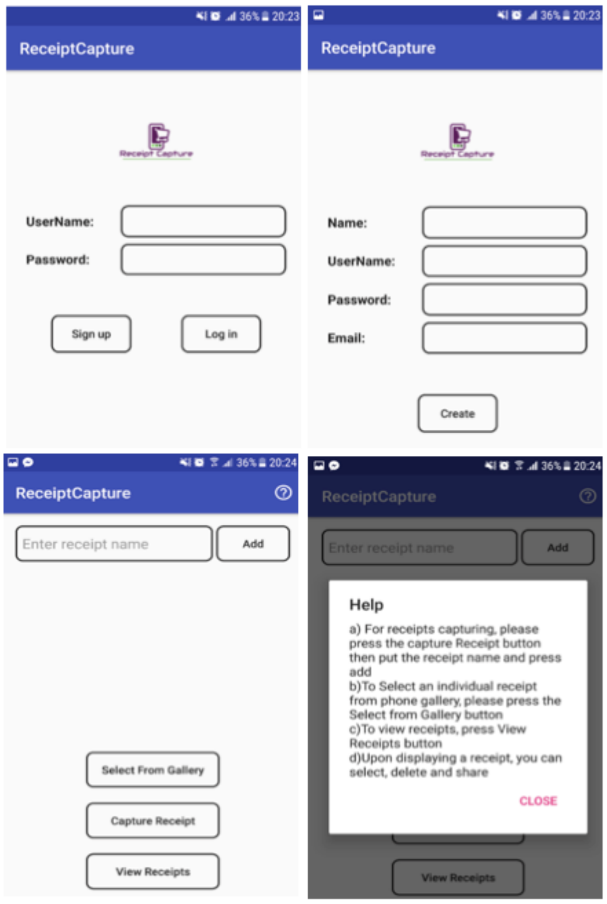
In this paper, we reflect on the application of an Agile method, Scrum, and related practices and tools used to develop an Android app within a university course. As mentioned above, the software's primary functionality includes capturing, uploading, storing, retrieving, and managing receipts on any Android device. The intention was that the software would be developed such that it is scalable, facilitating easy upgrades that may include other advanced features at a later stage. As such, the scope of the system was initially broad. However, given limited resources, Scrum

This quality assured paper appeared at the 8th annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2017) and the 30h Annual Conference of the National Advisory Committee on Computing Qualifications, Napier, New Zealand, October, 2-4, 2017. Executive Editor: Emre Erturk. Associate Editors: Kathryn MacCallum and David Skelton.

provided a framework to best deal with an incremental plan. By implementing Scrum, development iterations resulted in a shippable product, which meant that useable software was available at the end of the project. Figure 1 provides a snapshot of the main interface of the software, covering some of the features mentioned above.

This paper contributes evidence in the form of our reflections for how implementing Agile software development practices and tools for a university project can contribute to its success and quality. Our outcomes provide lessons for the mentoring of students in the use of Agile practices, and for novice developers using Agile methods in real projects.

The remaining sections of this paper are organized as follows. In Section 2 we provide our background, and we summarize our project setting in Section 3. We then provide our reflections in Section 4, before providing concluding remarks in Section 5.



Help

a) For receipts capturing, please press the capture Receipt button then put the receipt name and press add
b)To Select an individual receipt from phone gallery, please press the Select from Gallery button
c)To view receipts, press View Receipts button
d)Upon displaying a receipt, you can select, delete and share

CLOSE

Figure 1. Receipt Forwarding Application User Interface

2. BACKGROUND

Although extensive research has been conducted on the implementation of Agile methodology in software development projects, these projects have predominantly been conducted in various industry settings (Ilieva, Ivanov, & Stefanova, 2004; Fox, Sillito, & Maurer, 2008). As a result, limited information is available on the experiences and implications of implementing Agile methods in projects that are part of a university course. We conducted a literature search where several related research articles were identified. However, most of these articles relate either to university capstone projects or Agile methods theory, with some discussions about both. In the project we undertook, given that we only had one semester (13 weeks) instead of the usual two that is typical for capstone projects, the timespan was much shorter, thus, generating schedule pressures. Therefore, there is room for providing student reflections on the application of Agile methods in a project with a significantly shorter period of time, than in the case of capstone projects (normally done in one year).

In semester-long environments, the time to both develop the software and acquire appropriate skills is scarce and, hence, students potentially face greater issues in terms of scoping requirements, generating accurate estimation outcomes, fulfilling implementation goals, and confirming to specific project management guidelines. In fact, the same environment holds for those developing systems that are released rapidly for environment such as the App Store, where the pressure of targeting market opportunities mean that developers have a very short turnaround period (Licorish, Savarimuthu, & Keertipati 2017).

In the report by Sarang-Sieminski & Christianson (2016), Scrum was found to be a useful method for managing capstone projects. However, even though such projects were conducted over one year, many groups reported that they had not always properly or fully implemented this method (possibly due to lack of understanding of the recommended processes). Earlier work by Strode & Clark (2007) highlighted that findings on Scrum are frequently reported within industry settings, and that capstone projects most often implemented the Dynamic Systems Development Method (DSDM) or Extreme Programming (XP) when Agile methods were exclusively considered. On the other hand, Sarang-Sieminski & Christianson (2016) noted that capstone projects generally use more traditional methods during implementation, which may happen even when the intent is to be agile.

We wondered, *what differences may result in students using Scrum to develop and implement projects with stringent timelines and greater schedule pressure, compared to the much longer schedules that have been evaluated?* While such curiosities persist, there is limited information available around the implementation of Scrum in semester long projects in student settings. When Scrum is used in longer projects there are often reports that these were not implemented as recommended (Sarang-Sieminski & Christianson, 2016).

Evidence in support of answering the question posed above may help to inform both academic staff and students. Specifically, such insights could inform those parties involved about the appropriateness of these methods in short university courses, common issues in these settings, and key findings from student experiences. Consequently, staff members can better adapt courses to facilitate the use of these methods, as well as become aware of common issues related to their implementation by students. For students, this information can assist them in avoiding common pitfalls. This aspect is particularly important since students frequently have limited practical experience in running software projects.

Insights may also inform those developers that operate under very cramped schedule, such as those developing apps. In fact, since many student teams go on to become novice developers, insights gathered by answering the question posed above could be useful for this cohort of practitioners. We thus answered the question provided above in this paper.

3. PROJECT SETTING

Our Android app was developed as part of a software project management final year (year three) Bachelor of Science degree course at the University of Otago. Students complete this course as part of the Information Science, Software Engineering and Computer Science majors. The course covers multiple aspects of software projects, including software life-cycle models, specifying software requirements, software effort estimation, software project scheduling and prioritization, as well as risk and issue management. Other topics include project organization and teamwork, software configuration management, software quality management, managing software contracts, usability, and implementation planning. Specific emphasis is placed on how these aspects are implemented by Agile methods and Scrum teams. Thus, students are required to confirm to such principles throughout their projects (e.g., software requirements are expressed as user stories and group-based effort estimation techniques are used for estimation).

The course was administered in semester one of 2017, and the duration was 13 weeks starting towards the end of February. Group projects started with the forming of groups to pitch projects, finalizing team members (groups of 5), and completing a feasibility study. Thereafter, each team was required to scope their requirements, produce initial estimations, and complete scheduling and risk planning activities. Teams also performed software design and modelling the architecture of their proposed software (concepts learned in an earlier course). These activities then led to project implementation (software coding), which was done over two three-weeks Sprints. Teams attended 4 hours of formalized class sessions each week (2 hours of lectures and 2 hours of labs), and were required to spend another 6 hours each (30 hours altogether) every week outside of the classroom working on their projects.

Sprint (iterative) cycles of the development were implemented under Scrum, and therefore, many of the initial activities were re-evaluated and performed again as the project progressed (in each sprint). Eventually, the project went through quality inspection and testing. To conclude the project, an experience report was written to reflect on teams' experiences while using Scrum and its associated practices and tools to manage their projects. Altogether, there were six teams enrolled in the course.

In terms of the software itself, a need for the transition from manual to digital receipts was identified as a project pitch. This transition to digital receipts, which are stored on a mobile device, has advantages over paper-based receipts. For instance, adding a GPS location tag to the captured receipt can assist in validating proof of purchase at a site. This information can then be used during a range of other activities, such as tax auditing, marketing around that particular area, and, most importantly, warranty claims. The app's core functionality revolves around simplifying and centralizing the storage and management of receipts. Primarily, the core functionality involves either capturing or selecting images of receipts, naming, saving, and, finally, either sharing or deleting them. Further development could see these functionalities integrated with point of sale systems within businesses. The intent is that both the business and the customer will have access to digital receipts for purchases that were made. The app's functionalities are highlighted in Figure 2 (and screenshots in Figure 1).

The key outcomes of the course were to improve students' ability to delineate software requirement, come up with a project schedule, assign resources, and ultimately manage a project through the implementation of a high quality piece of software. As noted above, these outcomes were achieved using Scrum. The project requirement also required students to independently obtain technical knowledge in terms of selecting an appropriate programming environment and using appropriate frameworks for development. Apart from the technical aspects, the course attempted to improve students' skills of working in a team environment, while raising their awareness about topics such as different communication channels and personality types and other implementation considerations. We reflect on five crucial aspects of one group of students' (the first five authors) experiences in the following section.

4. REFLECTIONS

We reflect on the implementation of five recommended Agile practices during our development of the Android app introduced in the previous section. Requirements Engineering is considered in Section 4.1, and Software Estimation in Section 4.2. Agile Project Scheduling and Task Assignment are reviewed in Section 4.3, Agile Risk Management in Section 4.4, and finally, Version Control Management in Section 4.5.

4.1 Requirements Engineering: User Stories

4.1.1 Concept and Implementation

Unknown or unspecified requirements can lead to scope-related concerns during software development. However, it is difficult to elicit and specify all the software requirements at project inception. This is particularly challenging for projects where there is changing requirements. Agile development methods embrace this change, and are frequently adopted in such environments. User stories are commonly used to capture expressed requirements, which are added to a Product Backlog. As physical artefacts (story cards), they provide a convenient way of providing information about the features to be developed, and evidence shows that user stories greatly support the work of Agile teams (Sharp, Robinson, Segal, & Furniss, 2006).

User stories serve as high-level descriptors, resulting in simple, clear, and brief descriptions of proposed system functionalities (Cohn, 2004). User stories are written on a card, where the front serves to document the story of a feature/constraint, while the back serves as the space where the information on the front can be broken down and analyzed at a more granular level. This often helps with further analysis of unclear requirements if needed, leading to the discovery and creation of more features/constraints. User stories can be used throughout the systems development life cycle, as they are a quick and straightforward tool to use for expanding, refining, and consolidating what is required of the system.

The recommended format of User Stories is as follows:

As a <role of user>,

I want to <define activity to be performed>,

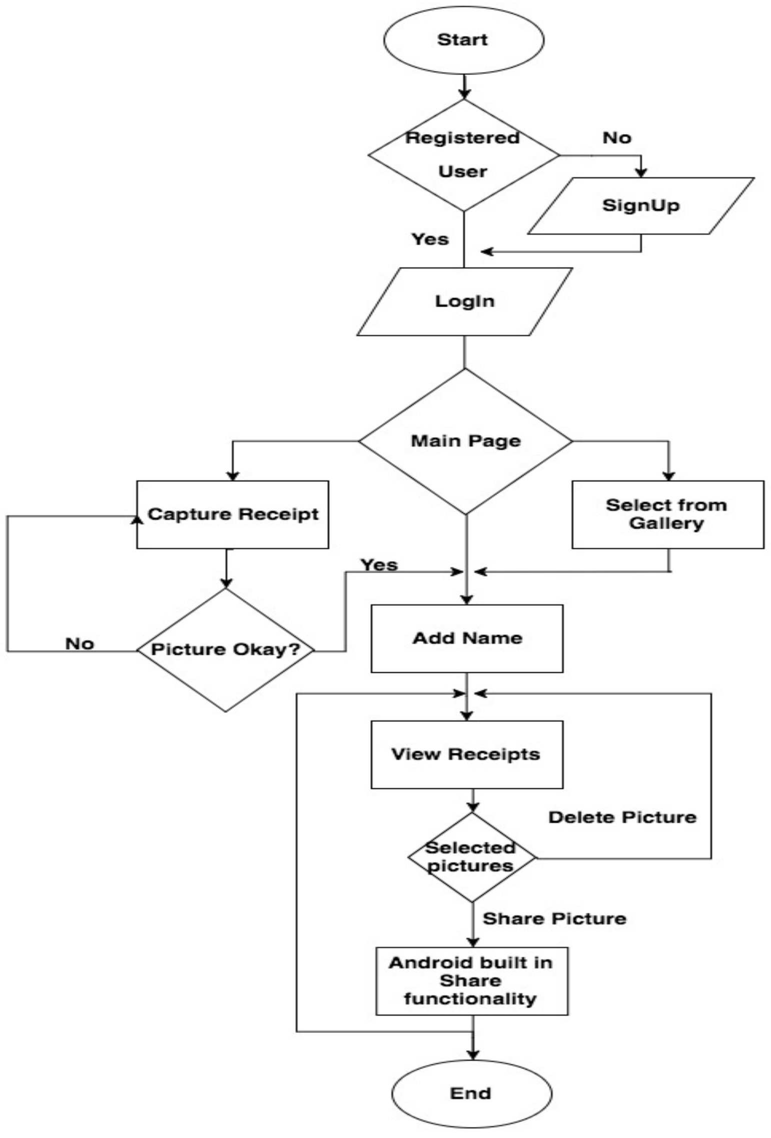
So that I can <define goal/end result>

This format is used because it places the interviewees into the mindset possessed in that role, allowing them to identify the main activities they need to do and highlighting why they need to do this. To this end, the interviewee takes on the viewpoint of the user's, which allows them to engage in a conversation that is structured towards identifying integral pieces of information (Cohn, 2004). User stories also serve as a structured center point of communication between clients and team members and acts as a foundation for specifying and documenting both user and system requirements.

4.1.2 Key Lessons From Implementation

User stories were used to establish both the functional and non-functional features required for our Android app. Within an uncertain and constantly changing environment, making use of an Agile based requirements gathering tool was preferred. During requirements gathering it was found that user stories are a simple yet effective tool. It requires minimal time investments to produce a significant amount of the key features that the system should have. In retrospect, we found that too many features were produced and added to the Product Backlog. This was due to the effectiveness and efficiency of the user story notation, combined with the team's lack of experience in Android development. To this end, the team experienced scope creep very early in the project, not knowing *when to stop*. Scope creep is the continuous or uncontrolled growth of project scope, often experienced in projects with changing requirements (Prabhakar & Ouah, 2008). The project was highly susceptible to both frequent requirement changes and the addition of extra features that were not initially considered in the feasibility assessment at project initiation. As a result, the team was required to invest effort in refining the project requirements and narrowing the scope of the Android app.

Based on our experience during the project, we would use user stories again, as it was a useful tool for stimulating thought and communication. However, we would spend more time brainstorming and creating a clear picture of the system (a high level view) before creating user stories. Alternatively, user stories could be done initially, directly followed by extensive discussion, analysis, and refinement of the requirements to ensure that scope creep is minimized, the aim of the project is satisfied, and the project has clear direction.



Flow Chart of Receipt Forwarding Application

Figure 2. Receipt Forwarding Application Flowchart

4.2 Software Estimation: Planning Poker

4.2.1 Concept and Implementation

Software estimation is the process of predicting the time and effort required to complete software development tasks (Clayton, 2014). Some argue that the purpose of estimation should not be to predict duration, but to control the project in order to meet the targets set (McConnell, 2006). That said, project success is sometimes measured against initial estimates. Estimation thus allows project members to produce concrete ideas in order to gain insight into how long features will take to implement, which features to prioritize, and which features may need to be removed. Through establishing estimations, teams can begin to more efficiently and effectively schedule, plan, and manage the project. In addition, estimation serves as a fundamental tool for client interaction, since its input for scheduling and planning purposes allow project managers to establish arrangements, agreements, and contracts. Therefore, estimation serves as a foundation for improving the likelihood of success for a project by providing a product that satisfies the needs of the client in a resource-effective manner.

In Agile development, group estimation is essential to both initial planning activities and subsequent iterations. Group estimation is frequently used as it has been proven to be more accurate than individual estimation (Molokken & Jorgensen, 2003). However, group estimation still leaves room for issues to arise. For instance, dominant personalities and anchoring effects frequently reduce group estimation performance and, as a result, can severely hinder project success. To this end, a group estimation method called Planning Poker was developed to reduce personality dominance and anchoring, as well as to enhance communication. The effects of this method have been noted and, previous evidence has shown that the Planning Poker estimation process improves teams' performance (Haugen, 2006).

Planning Poker is a collaborative approach to effort estimation in Agile software development projects. It is enacted through the use of cards, either physical or digital, where the results are integrated into planning and scheduling activities (Prabhakar & Ouah, 2008). The steps and rules of Planning Poker are shown below and are largely based on the outline highlighted by Molokken-Ostvold, Haugen, & Benestad (2008).

1. Assign members a deck of estimation-value cards.
2. Construct a baseline estimation scale by collectively agreeing on a low-value, small feature, and providing an estimate for it. A high-value, a large feature can also be used.
3. Descriptions of features are read.
4. Discuss all questions that the members may have.
5. Privately select an estimation value for the feature at hand.
6. Signal to all members so that all estimations are revealed simultaneously.
7. Discuss high and low estimates.
8. Re-estimate after discussion.
9. Repeat until a consensus is reached and repeat the process for all features.

4.2.2 Key Lessons From Implementation

Planning poker was used to estimate the effort required to complete each User Story, which was ultimately used to estimate the overall project duration. This technique improved the team's confidence regarding estimates and, by doing so, aided in improving the efficiency of the team's time use. It also significantly removed bias in the group and allowed for better estimations to be made concerning project risks and feature effort requirements. Most importantly, however, Planning Poker facilitated and encouraged greater levels of communication, discussion, and idea sharing. Through this approach, the members were able to combine their previous experiences and select estimates that reflected the entire team's competence. Also, it allowed all members to take responsibility and accountability for their estimates.

While using Planning Poker worked well for the team, due to the lack of experience, the estimations were found to be incorrect to a remarkable extent in the beginning. That said, when members provided estimates outside of the planning poker technique these were normally worst. Planning Poker worked sufficiently well and we would be happy to use this technique in future projects, as, even in an inexperienced team, Planning Poker still improved estimations. Also, Planning Poker was noted to work better when a more granular view was taken and when the discussions were detailed. In particular, in discussions, emphasis should be placed on why estimates are given and on what was learnt from previous experiences, including programming in other languages and on different projects. At points, particularly at the start, team members were producing estimations, but the discussions were not sufficiently detailed. Therefore, we recommend that, in future use, more emphasis should be placed on the debate than on the actual initial estimations of members. This would allow final estimation agreements to be based on specific evidence and thorough analysis, rather than on pure guesswork.

4.3 Agile Project Scheduling and Task Assignment

4.3.1 Concept and Implementation

Traditional project management techniques tend to embrace planning the project schedule as far ahead as possible, and thus, employs a deterministic approach (Newby, 2012). This means that the project is not very adaptable to future changes. Agile, and Scrum in particular, however, rely on a high-level plan, while the low-level plans and tasks remain flexible to changes (Newby, 2012). Since project requirements are very likely to change in the future, the Agile approach is more realistic for project management.

In Scrum, development phases are known as "sprints", each producing a usable deliverable to the client (Adi, 2015). Therefore, Agile focuses on detailed scheduling for each sprint, rather than on the overall project. This approach was more suited to the small-scale nature of the present project, where the goal of the project was to have a working application in 13 weeks, but it was necessary to demonstrate progress at various points over this period. Thus, there was a requirement for a tool that would aid the team to successfully deliver project tasks rapidly.

Agile scheduling was initially applied to the project by scheduling the dates for the sprints, without delving into lower level details such as task assignment. This was after the Planning Poker method was used to estimate the number of story points (in hours) for each of the user stories. These story points were used to guide the scheduling and prioritizing of tasks to sprints. Tasks with high story points (> 10) were assigned to two or three members of the group, as it was estimated to take longer to complete. After finishing the first sprint, we recorded the team's velocity, which reflected the performance for that sprint. This velocity was then used to guide subsequent task assignments in the next sprint. In previous research, tracking past decisions and actions has been shown to be valuable in stabilizing new projects. In addition, when compared to other practices, tracking can also lower overheads (Talby, Hazzan, Dubinsky, & Keren, 2006).

4.3.2 Key Lessons From Implementation

It was predicted that Scrum would work well, since the incremental planning and scheduling would improve adaptability. Our hypothesis was confirmed, as we were able to complete all the tasks for the project within the allocated timeframe. This was because Scrum allowed us to be adaptive and responsive. It was noted by the course lecturer (the sixth author) that it is best to implement a high-level long-term plan, and to develop the project incrementally based on short-term scheduling. This is an advantageous route, because there are likely to be changes, which were actually experienced in the project, as members had other courses and commitments interfering with the project schedule. It was also noted that allowing members to self-select tasks would be beneficial, as members would choose tasks based on their strengths. We experienced this to be useful, and especially gaging task difficulty using story points, as members were aware of the amount of estimated effort for their task and, therefore, they were able to effectively plan, schedule, and prioritize work.

In the future, it could be beneficial to consider other scheduling techniques, such as proactive, stochastic, and reactive approaches (Newby, 2012). The team also noted that, in the future, estimates will be made larger for activities surrounded by uncertainty. Also, in order to improve the performance of the team and enhance the project outcome, it is recommended that members spend sufficient time on planning sprints. The high-level plan the team produced served well to inform members of the overall goal and scope of the project; however, more emphasis should be placed on the planning of individual sprints. Although sprints were planned, on reflection, the team believes that, if more time had been assigned to this stage, then organizing and scheduling could have been improved and, ultimately, communication would have been better. The team also recommends that sufficient time should be spent on updating sprint details in the Product Backlog. Again, this function was used, but more time invested could have improved the management of the project. By and large we recommend a focus on improving communication by being clear, concise, up to date, and communicating using a central tool.

4.4 Agile Risk Management

4.4.1 Concept and Implementation

Teams go through the process of identifying and analyzing risks to come up with a possible plan of action in order to mitigate the effects of the risks should they happen. Agile risk management is no different, which considers risk identification, analysis and prioritization, planning and treatment and monitoring. Risk identification involves brainstorming what might occur, risk analysis and prioritization involves assessing risk importance, risk planning and treatment involves deciding how to react to the risks, and monitoring involves observing the risk over the project lifetime (Chin, 2004). Sometimes risk burndown charts are recommended for risk monitoring.

In an early Scrum meeting, the team identified time as the most valuable resource and identified several risks unique to the project. This involved a reflection on theory, as well as brainstorming potential hazards. Due to the lack of access to sophisticated tools and algorithms for a precise estimation of risk effects (which tends to not align with an Agile approach), we resorted to using Planning Poker for analyzing identified risks. This provided us with a means of getting a group estimate, which was averaged across the individual estimations gathered.

The process involved going through project-specific risks, which had been listed beforehand, followed by the steps listed below:

1. Estimating and discussing the probability of the risk occurrence (measured as a percentage).
2. Estimating and discussing the effect of the risk in the case of its occurrence.

To measure the level and effect of the risks, a number ranking system from 1 to 100 was used: 1 would be for little to no effect on the project if the risk occurred and 100 if the project would have failed or been required to change significantly. We then used this information to calculate the risk exposure by multiplying our risk effect with the probability of its occurrence.

4.4.2 Key Lessons From Implementation

In retrospect, the value received from the risk exposure was almost an arbitrary number that we used for prioritizing the risks, but it did not give any real information on time duration. Hence, risk management for future projects should ensure that the effects of the risks are measured as real values related to either time or money. By placing emphasis on producing more practical information, the team would be more equipped to schedule and plan events in a manner that would reduce the impact of risks. In the teams' opinion, Planning Poker (for risk analysis) mainly served as a channel for facilitating and directing the conversation, and it was out of these conversations that our mitigation plans were conceived. This mechanism thus may be useful as a tool for risk evaluation, similar to its utility for estimation.

4.5 Version Control Management

4.5.1 Concept and Implementation

Version control software enables developers to work together while archiving the history of their work (Sink, 2011). The primary goal of a version control system (VCS) is to allow developers to operate simultaneously rather than individually, as it gives them more control over their source code (Sink, 2011). The main reason behind teams using version control is to centralize work into one repository while having local copies in a decentralized fashion. This enables the entire team to operate without disrupting each other's tasks. Version control system such as GitBucket and GitHub also provide practitioners the ability to share documents and perform issue tracking. Hence, GitBucket and GitHub were used during all of our project phases, including specification, design, and implementation.

4.5.2 Key Lessons From Implementation

We received several benefits by implementing version management practices. The team was able to archive work in a central repository and roll back to any historical point, gaining access to all previous versions. Regularly checking in work to the GitBucket repository was also important, as it aided us avoiding conflicts on the tasks that were being completed. Specifically, GitBucket's functionality of highlighting code changes, its issue tracking functionality, and file storing ability enabled all members to get up-to-date and readily available information about each member's responsibilities and the current state of the system. GitBucket proved useful to back up the central repository online instead of having local repositories only, which, in our case, helped avoid delays due to technical issues. GitBucket facilitated development collaboration, enabling code reviews to pull the code from the repository. Thus, pushing or merging code back to the trunk (master) was extremely beneficial. The Issue Tracking features used by members on GitBucket and GitHub aided in identifying, publishing, and following challenges and features of the project that needed to be addressed. Finally, the Wiki was used to document progress and enable files to be accessible by all members and allowed access to specific key parts and tools used/created in the project, such as the Product Backlog. The ability to Branch source code also allowed the members to implement specific functionality without disrupting code in the master trunk.

Overall, the team discovered that GitBucket and GitHub allowed them to share and version files easily. In addition, these tools improved communication and teamwork and allowed for better control over the project and the development of the App. In particular, the pushing, pulling, merging, branching, issue tracking, and Wiki features allowed the group to be more efficient and effective. In terms of possible improvements opportunities, members initially performed prototyping development in a decentralized fashion, without integrating version control software. This meant that members were developing different features on their local repositories only. Due to this, the team experienced difficulties in manually merging the work that individual members had done. To overcome this problem, we would suggest using version control software at the inception of the project, even for prototyping. We also discovered issues with GitBucket, which then caused us to shift to using GitHub. The transition was not only time-consuming, but also required extra effort to be spent on attempting to work with GitBucket. Therefore, we would recommend creating an original master repository at the start and then ensuring every team member is able to successfully clone, push, and pull, etc. before development begins. With regard to collaboration, the team used various methods to share files. Although GitBucket was mostly used, the team also used emails to share files. To keep the project clean, consistent, and well organized, we would recommend using only GitBucket, or any version control management software that enables file sharing.

5. CONCLUSION

In conclusion, the success of the project reviewed was largely influenced by the implementation of key Agile project management practices and tools. In general, these tools helped to improve our understanding of the project's requirements, their size and priority, how to schedule them, keep track of the development, collaborate in an integrated way, and how to minimize the effects of risks. As shown in our reflections, these concepts, tools, and methods were vital to making our Android project a success. In fact, when tools and principles were not used as recommended we encountered issues that in hindsight we could have avoided.

As such, we firmly believe that Scrum can greatly enhance the outcomes of software projects in a university setting, and that the utilization of key Agile tools can benefit both professional developers and students. In fact, there are incentives to employing Agile practices in student settings during university courses, as this environment offers students the opportunity for trial and error, which may not be possible in a professional environment.

Furthermore, with the opportunity to practice such techniques repeatedly, students would develop key skills that could translate into professional practice, and especially given that Agile methods are largely used in the software development industry today (Licorish, et al., 2016). To this end, we strongly encourage the use of Agile techniques for semester-long student projects beyond the common use of such methods in capstone courses (Sarang-Sieminski & Christianson; 2016).

6. REFERENCES

Abrahamsson, P., Warsta, J., Siponen, M., & Ronkainen, J. (2003). *New directions in agile methods: Comparative analysis*. 25th International Conference on Software Engineering (May 3–10), 244–254.

Adi, P. (2015). Scrum method implementation in a software development project management. *International Journal of Advanced Computer Science and Applications*, 6(9). doi:10.14569/ijacsa.2015.060927.

Beck, K. (1999). *eXtreme Programming Explained: Embrace Change*. AddisonWesley, Reading, MA.

Boehm, B. N. (1991). Software risk management: principles and practices. *IEEE Software*, 8(1), 32-41. doi:10.1109/52.62930.

Chin, G. (2004). *Agile project management: how to succeed in the face of changing project requirements*. New York: American Management Association.

Clayton, R. (2014). *Software Estimation is a Losing Game*. Retrieved from: https://rclayton.silvrback.com/software-estimation-is-a-losing-game

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Boston, AL: Addison-Wesley.

Coram, M., & Bohner, S. (2005). *The impact of agile methods on software project management*. In Engineering of Computer-Based Systems, 2005. 12th IEEE International Conference and Workshops on the X (pp. 363-370).

Fox, D., Sillito, J., & Maurer, F. (2008). *Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry*. Agile 2008 Conference, 63-72. doi:10.1109/agile.2008.78

Haugen, N. C. (2006). *An empirical study of using planning poker for user story estimation*. In Agile Conference, 2006 (pp. 9-x). IEEE.

Ilieva, S., Ivanov, P., & Stefanova, E. (2004). *Analyses of an agile methodology implementation*. Proceedings. 30th Euromicro Conference, 2004., 326-333. doi:10.1109/eurmic.2004.1333387.

Licorish, S. A., Holvitie, J., Hyrynsalmi, S., Leppanen, V., Spinola, R. O., Mendes, T. S., . . . Buchan, J. (2016). *Adoption and Suitability of Software Development Methods and Practices*. 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). doi:10.1109/apsec.2016.062

Licorish, S. A., Savarimuthu, B. T. R., & Keertipati, S. (2017). Attributes that Predict Which Features to Fix: Lessons for App Store Mining. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (pp. 108-117). ACM.

McConnell, S. (2006). *Software Estimation: Demystifying the Black Art*. Redmond, Wa.: Microsoft Press.

Molokken, K., & Jorgensen, M. (2003). A review of software surveys on software effort estimation. 2003 *International Symposium on Empirical Software Engineering*. doi:10.1109.isese.2003.1237981

Molokken-Ostvold, K., Haugen, N. C., & Benestad, H. C. (2008). Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software*, 81(12), 2106-2117. doi:10.1016/j.jss.2008.03.058

Newby, E. (2012). *Project Scheduling in Software Development*. University of the Witwatersrand, Johannesburg, 1(12), 1-1. Retrieved from: https://www.wits.ac.za/media/migration/files/cs-38933-fix/migrated-pdf/pdfs-2/RCPSPMISG2012.pdf

Nguyen, T. N. (2006). A decision model for managing software development projects. *Information & Management*, 43(1), 63-75. doi:10.1016/j.im.2005.01.006

Phillips, D. (2004). *The Software Project Manager's Handbook: Principles that Work at Work*. X: IEEE Computer Society Press.

Prabhakar, G. P. & Quah, J. (2008). Scope creep in software development. *Journal of Social Management*, 6, 45-59.

Sarang-Sieminski, A., & Christianson, R. (2016). *Agile/Scrum for Capstone Project Management*. Presented at the Capstone Design Conference, Columbus, Ohio.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 18). Upper Saddle River, NJ: Pearson.

Sharp, H., Robinson, H., Segal, J., & Furniss, D. (2006). *The role of story cCards and the wall in XP teams: A distributed cognition perspective*. In Agile Conference, 2006 (pp. 11-x). IEEE.

Sink, E. (2011). *Version Control by Example*. X: Pyrenean Gold Press.

Strode, D.E. & Clark, J. (2007). *Methodology in software development capstone projects*. Proceedings of the 20th Annual NACCQ, Nelson, New Zealand.

Talby, D., Hazzan, O., Dubinsky, Y., & Keren, A. (2006). *Reflections on reflection in agile software development*. In Agile Conference, 2006 (pp. 11-x). IEEE.

Varajão, J., Dominguez, C., Ribeiro, P. A., & Paiva, A. (2014). Critical success aspects in project management: Similarities and differences between the construction and the software industry. *Tehnički Vjesnik*, 21(3), 583-589.