COSC345 Project Experience Report

AutoCull (Group 11)

Will Challis
University of Otago 7871182
chawi053@student.otago.ac.nz

Cam Clark
University of Otago 8298806
claca067@student.otago.ac.nz

Marick Malamala
University of Otago 9704746
malma002@student.otago.ac.nz

Daniel Paxton
University of Otago 4936848
paxda981@student.otago.ac.nz

Kevin Wang
University of Otago 7727089
wakev951@student.otago.ac.nz

***Abstract*—This document is intended to detail the experience of Group 11 throughout the development of AutoCull for COSC345. Included is a reflective assessment of the project management, design, implementation and testing processes. Lessons learned and recommendations for future iterations of the project are also discussed. Along with personal contributions to the project as a whole.**

***Index Terms*—Agile, Photo Management, Software Engineering, Teamwork**

I. COSC345 Introduction

COSC345 is a 300-level software engineering course offered at the University of Otago. The course aims to provide students with practical experience in the field of software engineering and delivery, and is designed as a capstone course for both computer science and software engineering students.

This allows for the application of theoretical knowledge gained in previous papers, as well as the development of soft skills such as teamwork, communication and project management. The application of good software engineering practices is also a key learning outcome of the course, with students gaining a better understanding of why these practices are important in a real-world setting.

The course is structured around a single project completed in groups of 4–6 students over the course of a semester. The project is to be marketed as either ed-tech or med-tech, have database persistance and incorporate AI.

The project is largely self-directed, with groups expected to attend a weekly mentoring session with a course mentor; meeting times outside of this are to be organised by the group themselves.

II. Project Overview

Professional digital photography is data intensive. A single photoshoot can quickly amount to hundreds or even thousands of digital images, many of which are similar or near-duplicates. Manually culling these near-identical images can be time-consuming and tedious but is a critical stage in the post-processing workflow of editing and delivery.

AutoCull is a desktop application which makes use of ML/AI algorithms to assist a photographer in culling large quantities of photos. In addition to culling, the application provides feedback to the photographer on the objective quality of their photos, providing a quick and easy way to identify their strong and weak images. This feedback can help photographers improve their skills over time, as well as providing a second opinion on the quality of their work.

Whilst the digital media market is already saturated with post-processing applications, there are comparitively few applications with such a strong focus on the culling process itself. AutoCull aims to fill this gap in the market, providing a tool which can significantly reduce the time a photographer spends on post-processing, and providing the feedback they may not otherwise receive. Subsequently, this allows photographers to spend more time doing what they love – taking (and editing) photos.

A. Objectives and Features

The objectives and features of AutoCull include:

- Assisting photographers in culling large quantities of photos using ML/AI algorithms.
- Providing feedback on the objective quality of photos to help photographers improve their skills.
- Streamlining the post-processing workflow to save time and effort for photographers.

Although objectives and features of the project are rather vague, this is largely due to the nature of the project itself. The project is intended to be a practical exercise in software engineering, and as such, the focus is on the development process rather than the specific features of the application. The option to take any objective further than the minimum requirements is left to the discretion of the group, allowing for adjustments to be made throughout the project as the team gains a better understanding both scope and technical feasibility.

III. Development Process

As COSC345 is a paper with a far greater focus on practical experience than theoretical knowledge, the development process is largely self-directed, with groups expected to manage their own time and resources to complete the project

within the semester timeframe. This section aims to detail the development process undertaken by Group 11.

A. Project Management

Group 11 adopted an agile development methodology, with two-week sprints and regular stand-up meetings. This allowed for team members to put agile methodologies into practice, further cementing the theoretical knowledge gained in previous papers.

The team utilised Taiga as a project management tool, as per the requirements of the course. This allowed for a centralised means of user-story management, sprint planning, and task assignment.

In practice, the agile methodology adopted by the team could have been followed more strictly. The act of breaking down sprints into user stories for the project planning stage was generally well executed. However, the the estimation of task sizes was often inaccurate, leading to sprints being either over- or under-committed. This is almost certainly due to the lack of experience of team members in estimating task sizes, and is an area which could be improved with more practice.

Additionally, stand-up meetings were generally productive provided that an agenda was followed, however this was not always the case. There are instances where meetings would proceed with no clear direction, leading to unproductive discussions and a lack of productive outcomes. This could have been mitigated by having a more structured approach to meetings, with either a clear agenda or objective set prior to the meeting.

If the team were to undertake a similar project in the future, the development methodology would likely be similar, with a few key changes. The team would aim to follow the agile methodology more strictly, with a greater focus on accurate task estimation and structured meetings. Additionally, the team would aim to incorporate more frequent retrospectives, allowing for continuous improvement of the development process throughout the project.

B. Team Roles and Responsibilities

At the beginning of the project, the team operated under a decentralised, emergent leadership style. Where roles are not formally assigned, and individuals were expected to step up and lead tasks as needed. This model offered flexibility and the potential for shared ownership, allowing different members to contribute leadership depending on their personal expertise.

However, the lack of a clear project lead created ambiguity. Without a central point of accountability, the team struggled to establish direction, set priorities, and coordinate effectively. While this approach was able to better encourage both participation and inclusivity, it often led to team members looking to others to take charge, resulting in delays and inefficiencies, especially where decisive action was required.

Upon recognising these shortfalls in management structure, the team opted to transition to a more traditional hierarchical model as presented in COSC345 lectures. The team was divided into two pairs, each responsible for the front-end and back-end development respectively, and a designated project lead. This structure provided clearer accountability and a more defined chain of command, while still maintaining opportunities for collaboration and shared input.

This restructuring brought greater clarity and efficiency. The division of labour allowed tasks to progress in parallel, and the project lead provided much-needed direction to the project. Compared to the initial emergent model, this approach was more effective as it ensured both structure and flexibility.

One limitation, however, was that the project lead emerged through self-appointment rather than consensus, and their relative lack of experience occasionally impacted decision-making. The experience demonstrated that while decentralised leadership can be valuable, software projects often require a clearly defined leader — ideally one selected with regard to competence and team buy-in — to keep the group aligned, especially under time constraints.

Given an opportunity to work in this particular team again, the team would likely adopt a similar hierarchical structure from the outset, rather than expecting leadership to emerge organically. This would provide the necessary clarity and direction for this particular team to function effectively. Had such a structure been in place from the beginning, it is likely that the project would quite easily be on track to be completed early, and with additional features implemented by the final deliverable.

C. Design Process

The design process for AutoCull involved the several stages, including requirements gathering, system architecture design, and user interface design. With certain aspects of the project dictated by the project specifications provided.

Requirements gathering involved consulting with both actively working photographers and those with a keen interest in photography. This allowed for a better understanding of the needs of the target users. Key insights from requirement gathering included:

- Potential users did not want an application to replace the likes of Photoshop.
- The photos should not be automatically culled, the user should have the final say, and the application should not update the local filesystem.
- The use of AI/LLM giving the photographer score and feedback on a collection was well received by the target audience.

System architecture design pulled on the prior knowledge of team members, as well as content throughout COSC345. The architecture was designed to be modular and extensible, allowing for future features to be added with minimal disruption to existing functionality.

The gathering of requirements through approaching actual users was a particularly valuable exercise, likely among the most valuable of the entire project. It provided a real-world context to the project, and allowed for a better understanding of the needs of the target users. This was particularly important

given the niche nature of the application, as it ensured that the features developed were relevant and useful to photographers this application is meant for.

One aspect of the design process where the team performed quite well is the documentation of design decisions. This is particularly important when working in teams, allowing for team members to be quickly brought up to speed on both the application design and changes made throughout the project. Although this team did not experience a turnover of members, it is comforting to know that the documentation is in place should this occur. However, it is unfortunate that this documentation was not referenced more often throughout the project, as it would have likely prevented instances where components required reworking due to misunderstandings regarding the intended design.

User interface design was approached through extensive research into existing post processing applications, as well as general UI/UX design principles. The goal was to create an interface that was both intuitive and efficient, allowing the target users to quickly learn the application through implementing familiar design patterns shared by other popular photography applications.

While there was initially some friction within the team regarding design decisions, where it was unclear whether a certain UI design would make sense in the context of this application, this was largely mitigated through gaining feedback from potential users. This feedback was invaluable in refining the design, and ensuring that the final product was both functional and intuitive to the end user. With the understanding that we wanted to create a desktop application, the team decided to use Python as the primary programming language, leveraging the Tkinter library for the user interface.

User interface design focused on creating a simple and intuitive interface, with a focus on usability and user experience. Wireframes and mockups were created using Figma, allowing for rapid prototyping and iteration based on feedback from potential users. This was a particularly effective approach to the problem as it allowed for quick and easy changes to be made to the mockups while obtaining user feedback.

Upon reflection, the design process could have been improved by involving potential users earlier in the process, and having the entire team present for both the gathering of requirements and the user interface design. This would have allowed for a more cohesive design, with less misunderstandings and miscommunications between team members. It should be noted that this could possibly also be worked on through higher quality documentation of design decisions, allowing for team members to be on the same page regarding the design of the application. This would have prevented instances where components required reworking due to misunderstandings regarding the intended design.

D. Implementation

While the tech stack was detailed during the design process, the actual implementation initially differed from the proposed design. The start of the implementation process saw inadequate planning and task allocation put into practice. Leading to team members working on tasks they were not best suited for, and important decisions being made without the input of all team members.

One area where this was particularly evident was in the prototyping of the GUI. The proposed technical stack included the use of PyQt for the GUI, however this was almost immediately changed to PyEel after a team member determined that PyQt was too complex for the scope of the project. This decision was made without the input of all team members, and ultimately led to a full rework of the GUI when it was later decided that Tkinter would be a more suitable choice for the application. This could have been mitigated by having a more structured approach to decision making, with all team members being involved in important decisions. Additionally, the lack of experience of some team members with certain technologies led to suboptimal choices being made, making this project a particularly valuable learning experience for all team members.

The beginning of the implementation process consisted of a lot of research into the various technologies and libraries that would be used in the project. The prototyping which followed was not as well structured and directed as it could have been, with minimal input from members of the team which were not working on the specific prototype. This led to a lack of cohesion in the early stages of implementation, with different parts of the application being developed in isolation from one another.

This resulted in a lack of integration between different parts of the application, and a lack of consistency in the overall design. The team solved this issue by taking a step back and re-evaluating the implementation process. In doing so, large amounts of existing code was discarded, and many features were reimplemented in a more cohesive manner.

While the initial stages of implementation were somewhat disorganised, the restructuring of the team into a more traditional hierarchical model led to a more structured approach to implementation. The division of the team into front-end and back-end pairs allowed for tasks to be completed in parallel, and the project lead provided much-needed direction to the project.

E. Version Control

One notable topic in every software engineering course offered at the University of Otago is version control. It allows for collaboration between team members, tracking of changes, and the ability to revert to previous versions of the codebase if necessary.

For development, the team used both Gitbucket and GitHub for version control, while this provides the unintended benefit of redundancy in the case one service may go down, the decision to use both services was made to allow for the use of GitHub Actions for continuous integration and deployment, while also adhering to the course requirement of using Gitbucket.

Good practice in the use of version control is something that the team was well aware of, and was put into practice throughout the project. The use of feature branches allowed for team members to work on specific features without affecting the main codebase, and pull requests were used to review and merge changes. This ensured that the main codebase remained stable, and that changes were thoroughly reviewed before being integrated.

The use of version control is something that team was already relatively well versed in, and was put into practice effectively. It is unlikely that any changes would be made to the use of version control if the team were to undertake a similar project in the future. Given the option however, it is unlikely GitBucket would be used again.

F. Testing and Quality Assurance

It is a well known fact that test driven development is a highly effective software development practice. Although this is something that the team was well aware of, it was not put into practice until the later stages of the project. While this is something that could be seen as a shortcoming of the team, it was convenient that less time was spent on tests which would have been discarded due to the significant changes made to the codebase throughout the project.

If this were to be done again, the team would likely work towards ensuring that best test driven development practices are followed from the beginning of the project. Although it worked out better to not do this in the context of this project, it could easily be argued that this was the result of poor planning to begin with, which led to the need for significant changes to the codebase.

Where implemented correctly, tests should be written before the actual implementation of a feature. With features being properly planned out before any code is written. This would allow for a more structured approach to development, and would likely lead to a more cohesive codebase, which would not require significant changes to be made later in the project.

In addition to manual and automated testing, Group 11 experimented with static analysis tools to improve code quality. The team primarily used `pylint` and `flake8` for Python code linting. These tools highlighted areas of improvement in naming conventions, documentation, and adherence to PEP8 style guidelines. While not all warnings were critical, they encouraged the team to maintain a cleaner and more readable codebase. In particular, the process helped identify missing docstrings, unused variables, and instances of duplicated logic, which were subsequently refactored.

Overall, static analysis complemented testing by catching issues earlier in the development cycle, before they could manifest at runtime. If more time had been available, the team would have integrated these tools into a continuous integration pipeline, ensuring that code quality standards were automatically enforced on every commit.

Testing is something the team has identified as a key area for improvement, and is something that will be focused on in future projects. The importance of testing in software development cannot be overstated, and the team is well aware of the cost in both time and effort that could have been saved, had proper testing and planning been put into practice from the beginning of the project.

IV. Lessons Learned

The undertaking of this project has been a valuable learning experience for all team members involved. The practical experience gained thus far at week nine of the project has been invaluable. The experience of working in a team, managing a project, and developing a software application has provided insights that cannot be gained through theoretical knowledge alone. The importance of good software engineering practices, such as version control, testing, and documentation, has been reinforced through the practical application of theoretical knowledge.

Additionally, it has been a worthwhile exercise for the team to make mistakes, understand where things went wrong, and learn from these mistakes. While these mistakes were often not trivial, and rather time consuming to rectify, they showed that getting things correct from the start prevents a lot of wasted effort later on. This has been a particularly valuable lesson for all team members, and is something that will be taken forward into future projects.

Plenty of lessons have been learned throughout the course of developing AutoCull so far, although many of these lessons were preventable through better planning, communication, and adherence to good software engineering practices. The team now has a far more comprehensive understanding of both the software development process, and the importance of good software engineering practices.

V. Individual Contributions

A. Will Challis

- Introduction
- Project Overview

B. Cam Clark

- Version Control
- Implementation

C. Marick Malamala

- Project Management
- Team Roles and Responsibilities

D. Daniel Paxton

- Testing and Quality Assurance
- Objectives and Features

E. Kevin Wang

- Report structure
- Design Process

VI. Conclusions

The development of this application throughout the course has been both challenging and rewarding for the members of Group 11. The project proved to be an opportunity to put into practice the theory learned throughout time at university. While also highlighting the complexities of teamwork, communication, and technical decision-making.

Although the project encountered some setbacks during the earlier stages of development, they were an important learning opportunity for all team members. The fact that the team was able to identify these issues, and make changes accordingly shows that learnings have been put into practice right from the start.

Important lessons for the team include the importance of clear communication, structured project management, and adherence to good software engineering practices. Most of these things were applied to a reasonable extent, however, there will always be room for improvement.

Although not yet complete, the project is well on its way to delivering a functional photo management application that meets COSC345 requirements. More importantly, the process has equipped each team member with practical experience in project management, collaborative development, and the application of professional software engineering practices–skills that will carry forward into both future academic work and industry practice.