CMPT 371 – TEAM 3

OVERALL PROJECT REFLECTION

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

April 6th, 2016

1 Continuous Integration

1.1 Build Success Rate

Our continuous integration was the Unity Cloud Build (UCB) service that is offered for free for Unity projects. As of April 4th, we have had 112 builds: 73 successful, 33 failed, 2 cancelled, and 4 that have started on UCB and disappeared for unknown reasons. The disappearing builds appeared to have occurred due to glitches on UCB (i.e. one build appeared to hang on the “building” stage for 3 hours, then disappeared with no trace that it had even started).

Our total Build Health lands at 68%, which is much lower than we would have liked. Some extraneous cases occurred where there were successively failing builds that dramatically lowered the overall health of the project.

The most recent of these cases was the period from March 31st to April 3rd, where 5 failed builds occurred in sequence. The first commit in that sequence also happened to be our first commit from the station provided by Luxsonic, so we had thought the issue was related to that. After a few days of stressing and a few attempts to rectify the issue (including a revert to a commit before the Luxsonic machine had been involved), we noticed that coincidentally Unity 5.6 had been released on March 31st. After checking the settings on UCB, we found that the project was set to compile on the latest stable 5.x version of the engine, and that UCB was attempting to compile code for Unity 5.5.1f1 on a Unity 5.6.0f3 engine. After changing this setting to compile on the correct Unity version, the builds continued to succeed.

The other main source of build failure occurred on February 16th and 17th, where there were 10 failed builds in sequence. These were due to a combination of library issues, as well as problems with system/unit tests running differently in UCB than in the Unity Editor. A specific example that occurred a few times was when a test used the camera position as reference. Within the Unity Editor, this is fine; it automatically creates a default position for camera references. UCB, however, does not allow this, and will throw a null exception whenever a camera is referenced.

This issue, and similar cases, were eventually fixed as we gained a better understanding of the differences between the systems, and how we were unable to rely on the consistency of some testing behaviour in UCB. Failed builds due to this inconsistent behaviour are not limited to the builds on February 16th and 17th, however, the builds on those days were in large volume and similarly represent a large amount of these cases.

1.2 Impact of Build Cooldown

Because we have been using Unity Cloud Build on a Unity Personal License, we had a 1 hour cooldown on our project between builds. This would sometimes delay our feedback on a commit, and how it would fare in a build. We have had 43 builds (38% of the total) that have needed to wait for this cooldown to finish before the builder would be allowed to run, totaling in 29.95 hours of wait time, and an average wait time of 41.8 minutes.

From these 43 builds, 15 (slightly over 1/3) of them are failed builds. This means our group had spent (based on the average) roughly 10.45 hours waiting for feedback on failed builds. This may seem like a small number compared to the total hours spent in development, but it did cause significant delay in cases where other groups relied on a feature being implemented successfully before starting/continuing development on their tasks.

1.3 Impact of developing for Luxsonic using educational licenses

Due to issues with Unity licensing we were not able to push from the school computers as they had educational licences. However, the only computers which could run the Oculus Rift headset were the computers on campus. This slowed development down as we were not able to commit the changes we made on the school computers. Instead we would record all the changes we made and try to move them over to our personal computers.

Many small details were missed as we did this, resulting in an increased amount of time spent finding these mistakes and correcting them. Being able to commit from the school computers would have removed the time spent transferring changes completely.

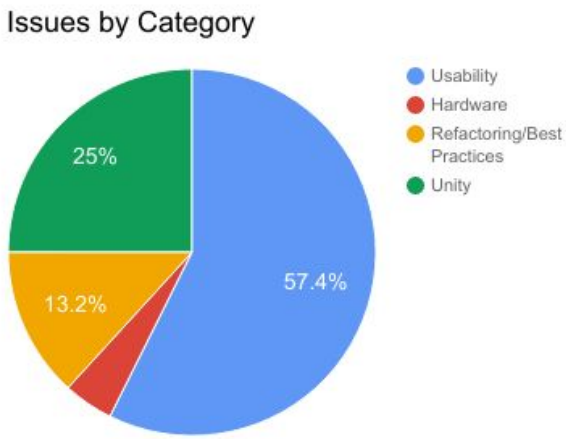
2 Issue Trends

Throughout the project, we have kept track of all issues with the project in the GitHub issue tracking database. The nature of this system yields four major categories of issues: Hardware, Usability, Refactoring/Best Practices, and Unity related issues. Many of the issues discovered during this project were related to usability. The VR aspect of the system poses many new challenges for creating a natural, seamless experience for the user. These issues have been recorded by user testing.

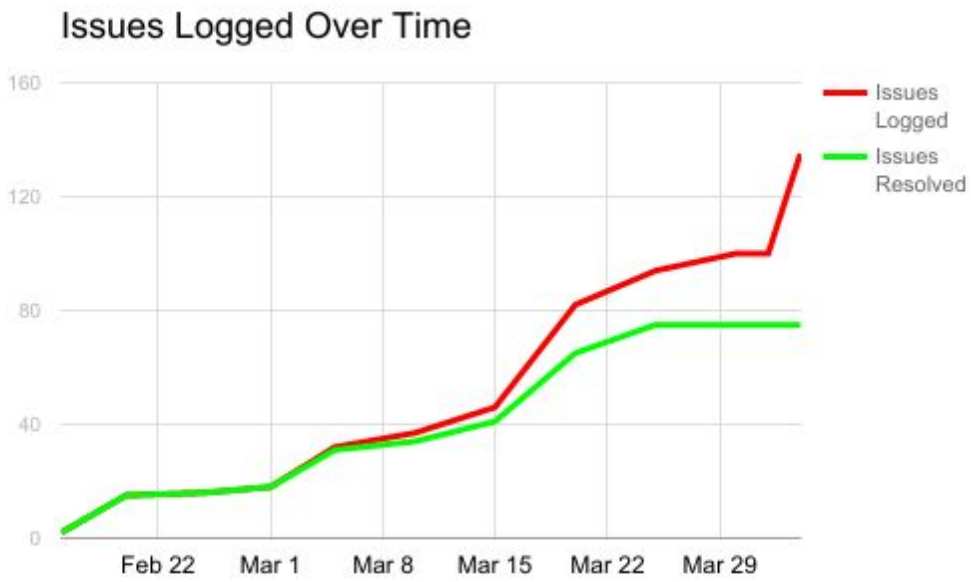
Hardware issues are issues that come with the Oculus workstations required to run the software. There are few of these issues, but the newness of the Oculus rift comes with unknown bugs and issues.

Refactoring/Best Practice issues are issues related to optimizing code and keeping a clean code base that is easy to understand and build upon. These issues arise from misunderstanding of design as well as the learning curve required to develop for the Oculus.

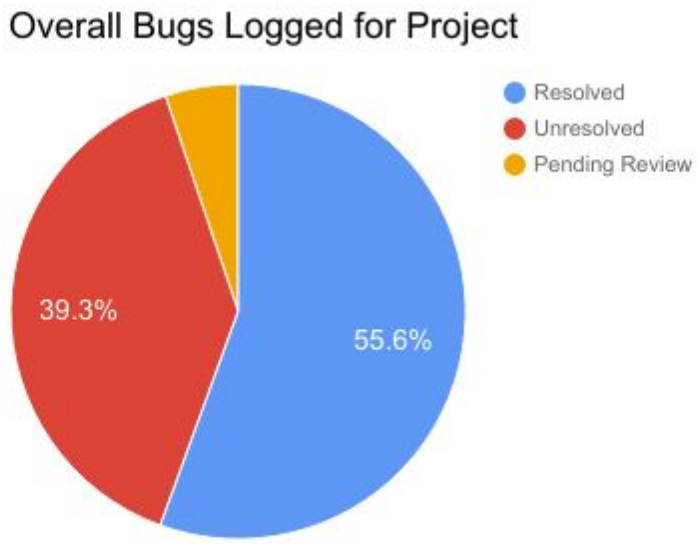
Unity-related issues are issues that are inherent to Unity and the versions of the packages we are using to develop with Unity. These include issues with testing frameworks and continuous integration servers. These issues are generally out of the scope of this project to fix, but still have great effect on the functionality of the system.



As the project has progressed through time, we encountered a wide variety of issues. We prioritized these issues by severity and complexity in order to get the most effective work done in the least amount of time. We held 2 bug parties to help identify defects in the project. Below is a graph of our issues tracked over time.



We can see that around the times of the bug parties (Mar 15 and April 1) we had a sharp increase in logged bugs. This helps us understand the status of our system in terms of remaining defects with the current set of features. We have estimated that there are around 40 undiscovered bugs remaining in the current version of the software. Overall, the team has done a good job of tracking defects and categorizing them. The current state of the project shows the following breakdown of defect status.



3 Developer - Tester Rotation

Halfway through the project, two of our developers and testers each traded roles. This was done to facilitate communication between the teams and share knowledge between them, while allowing more people to gain experience with VR development. The four of them recorded their feelings on the success or failure of the rotation.

3.1 Nhi Phuong

> For iterations one to three, I was a developer and for the remaining time, a tester. With my lack of knowledge in Unity, I often participated in pair programming sessions with developers who had more experience with Unity. In the beginning, I spent most of my time trying to contribute ideas to solve problems, since I was unfamiliar with Unity and C#. Once I got more comfortable with Unity and C#, I was able to develop with or without the help of another developer. Since we participated in the SIIM Innovation Challenge, I was able to spend extra time developing, allowing me to contribute more to the project, but this also meant that I did not spend as much time testing. Since there were many setbacks, we chose not to switch roles during iteration three. If this was not the case, I believe that we could have

used iteration three to familiarize ourselves with the other team's procedures. This would have led to a smoother transition.

Since Unity has complications in regards to testing, we were not able to implement system, integration, and regression tests. Instead, we created a way of manually doing these tests and documented the process. The transition allowed me to experience two different roles during the term, providing me with different perspectives. I learned that there are things that each role could do that would benefit others. For example, writing cleaner code that testers could easily understand would reduce obstacles for creating tests. Overall, I believe that switching roles is a good idea, as long as a good plan for transitioning is in place.

3.2 Heramb Sawant

I was a developer from the first to the third deliverable along with Nhi Phuong, and switched to the testing team for the remaining deliverables. I had no experience with Unity or C# prior to this course, so I spent most of the time as a developer pair programming. My previous experience with C++ made it very quick and easy for me to pick up C#, however the Unity framework required a little longer to understand.

Pair programming made learning very easy and I could watch an experienced Unity programmer work their way around the framework and always had someone i could ask for help. As a result, I picked up Unity relatively quickly and was able to properly contribute to the project. Through ID one to three I gained a lot of experience and knowledge about the code base which made the transition to testing much more productive. I was also aware of the developer/tester switch, so I kept thinking of how I would write code which could be tested. This resulted in a codebase which was much cleaner and consistent.

When I joined the testing team it was relatively easy to think of good test cases as I had so much experience with the code. It was easier for me to test code in Unity as I had already understood the framework through pair programming sessions. I found that the developer/tester switch was a success. I gained a lot of valuable knowledge on both sides of development. It also helped me understand what side of development I was more comfortable and enjoyed more. I would not like to try this in a professional

environment. It made the development and testing process a little longer and did not give me the chance to fully commit to one side.

3.3 Brianne Kostur

For this project I was a tester for the first three iterations, then switched to development for the last two. I found being part of both teams to be very beneficial to my understanding of software development processes. Starting in testing helped me to understand all the different types of testing and how they would be beneficial to our code. It also made me think about the practices I would use to code when I switched to development to allow testable code. There were challenges that came with starting in testing first, as I had no experience with C# or Unity. Having no experience made it difficult for me to fully understand what kind of things can be tested in Unity. So, I did some pair programming to help me understand how Unity works and the basics of C#. The pair programming helped a lot, though it still wasn't as effective as getting hands-on experience with developing in Unity.

When I switched to development, I pair programmed at the start to familiarize myself more with Unity and I picked up on it fairly quick. Once I became more confident, I was able to program features on my own and found being on the development team quite satisfying, especially when a feature gets implemented and works. Overall, the switch was a beneficial thing to do for this class because it taught me about what both teams do and the processes they follow to do it.

3.4 Brady Lang

I was on the development team and early on switched over to the testing team as they needed additional assistance. After the third deliverable, I switched back to the Development team. In both cases, I was learning things on the fly and reading documents to keep up to date on the latest plans going forward. My experience with Unity was limited to 2D, so taking on 3D in VR was a whole other challenge. It was nice to be on the testing team early, as that team ironed out a lot of the VR best practices, and early challenges. Pair programming was essential to gaining knowledge in both testing and development

in the early phases. It saved a lot of headache, and the stress caused by communication was much lower.

We were able to cover each other's knowledge gap about the testing or development processes, allowing us to shift to more solo-oriented work as we went on. Observing the development process and testing around it gave me a strong foundation to build off of when I became a developer. I felt like I had built up an understanding of the dev's system as a whole, and was in a good place to switch into development. Being a part of the testing team also helped me understand the different types of tests and how they assisted our system. This focus also allowed me to consider testing as more than an afterthought when switching to development.

In ID2, there were challenges in communication between testers and developers. A later development freeze and adding too many system-changing features required us to redesign our tests time and time again. We had to iron out the communication channels and describe how the development process was affecting us. While we made a point to communicate challenges and work through them, and things definitely improved, the tester-developer rotation really solved these communication issues.

As the tester team became developers, we were developing with the testing team in mind. We knew to call freezes early, what small things we could change without breaking or interrupting the work of the testers. The tester-development rotation also resulted in cleaner, well documented code going forward to stop the head scratching of the testers when it came time for them to work through our code. It also gave me a greater knowledge breadth about the project. By understanding both testing and development, I was able to learn more things, albeit, not to the depth I'd have preferred. Overall, I have a better understanding of both sides of the development process and a better understanding of the system because of it. I found the developer-tester rotation to be beneficial.

4 User Testing

Over the course of the project, user testing was performed with a range of different users. Characteristics we looked at consisted of experience in VR, Computer Science, technology in general, and the medical field. We also took into consideration the user's eyesight and susceptibility to motion sickness. We consistently found that all users enjoyed the interactions that they were able to

perform using different motions with their hands and the virtual objects. It allowed them to be more immersed into the workspace. This included motions that we were not able to fully implement such as resizing with two hands. All users had difficulty with the buttons in the workspace since they did not provide feedback indicating success. This was also an issue with the copies until selection indication was implemented.

We also found that each individual imagined something a little different in regards to how the workspace should be arranged and the restrictions that the workspace should have. Some users enjoyed the idea of a free workspace, allowing them to display copies in whichever location desired while others wanted a restricted environment to precisely place copies side-by-side.

Additionally, we could see that those who are used to the current programs offered for viewing medical images were more inclined to prefer something a little more restrictive. These users expect to have monitors similar to their current workstation instead of images in a free space. Although they felt this way, most saw the potential for the ability to create infinite copies in the workspace with certain restrictions in place, such as a grid to snap the copies to.

Age (along with other influences) also seemed to play a role in the user's ability. Younger users had an easier time adapting to the touch controllers; this could be the result of using other devices similar to the touch controllers. Older users had a harder time understanding how the basic controls worked, increasing the difficulty to also understand new features such as different motion controls (e.g. motion of grabbing an object). As testers, we had a harder time trying to explain the controls to an older user because we did not take into account the complications of learning how to use touch controllers to those who are inexperienced with controllers in general.

5 Impact of Testing Shift on Development

The testing tools provided by Unity within the Unity framework made it extremely hard to create tests. Switching the test plan to more manual test meant we spent less time trying to force tests in the system. This resulted in much less time being wasted finding a productive method of testing. We instead used this time to run more manual tests and make sure that everything worked as it should.

Even though manual testing takes much longer, it was still more productive for the team in the long run, as more time was spent actually testing the system instead of trying to come up with a clever workaround for system and integration tests in Unity.

We also noticed that manual testing helped find more bugs and issues within the code as we spent so much time working with the system.

From the fourth deliverable onwards, the developers started closing bugs with testers present. This often resulted in better code quality and fewer errors in the code. The testers would often find that the code requires minor refactoring which resulted in clearer code. Testers also found many areas where assertions could be thrown into the code where the developers missed. The acceptance tests after closing bugs also helped identify the impact of the fix on the rest of the code. Testers closing bugs ensured that the fix did what it was supposed to, without affecting the rest of the system.

6 Client Relations

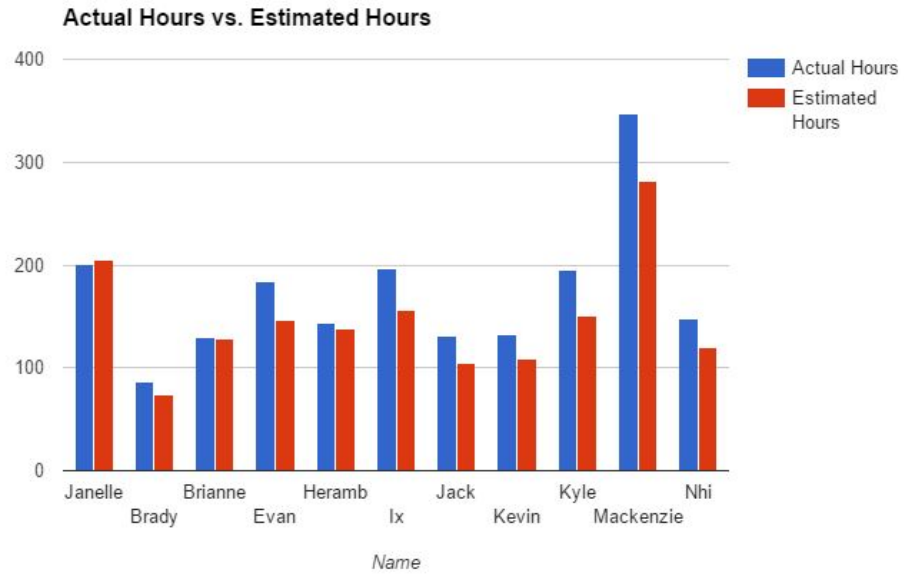
Our client made it clear from the start of the project that the most important part of the project was getting what we need to for the class. He was very cooperative and responsive with the team. Mike took the time to understand the problems we faced, as well as why we were implementing our features the way they are. This resulted in an environment with less stress leading to a more productive and cooperative team.

The original scope of the project was very reasonable. We reached most of the goals we set in the start of the semester except for one, the import and manipulation of 3D volumetric scans. This was a significant portion of our project; however, many features took longer to implement than anticipated, which resulted in less time for the team to work on volumetric scans. We made the decision as a group to refine our current project rather than add an extra feature at the end without much time for testing. This allowed for a project which contained most of the requirements as well as tested and debugged. Mike was very understanding of how we prioritized these aspects of the project.

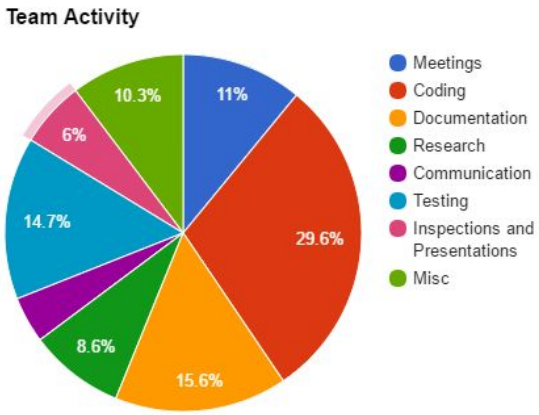
Mike gave feedback on his experience with the project at the end. He expressed concerns about CMPT 371 in general regarding a lack of intellectual property policies and wished there were more formal guidelines in place. Aside from that departmental hiccup, he was pleased with his experience with our group, praising the team’s professional behaviour and maturity. He also mentioned that he felt the team did an admirable job considering the technical challenges faced, and appreciated the team’s passion for the project as a whole.

7 Activity log breakdown

Over the course of the project, we tracked the team's time spent on various activities. Each member of the team was required to give a time estimate for their tasks, as well as an actual figure after they were completed.

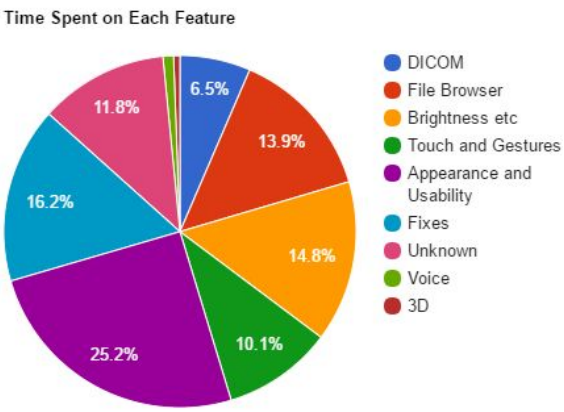


As you can see in the above figure, almost every group member spent more time on their tasks than they estimated. As the project went on, it became easier to predict how long a task would take, but estimations were still below actual hours overall. Bugs and unexpected issues were the main reasons that actual hours were so much higher than estimated. Working with virtual reality made it hard to build and avoid potential problem, as no one on the team had previous experience with this technology.



The majority of time spent by the group consisted of development activities, such as coding. Coding and research for the majority of this project went hand-in-hand due to the lack of Oculus Rift experience on the team and the unique requirements of our system, such as parsing DICOM files.

Our team spent a significant number of hours on documentation. One thing that we could have changed over the course of the project was actually spending less time on documentation. It seemed appropriate to invest time into making good-quality documentation for a university project, but in a real-world project where many of these documents would be for internal use only, spending as much time on documentation as the team did would be a poor decision.



Due to the lack of best practices and resources online, and the unique challenges of VR interfaces, the majority of time spent on features consisted of usability and appearance issues. We learned a lot about best practices as we went through this project through manual testing. These manual tests were performed by both developers and other users external to the team, such as medical professionals. This form of testing required a lot of organization and time. We recorded all the feedback we got and worked towards a more user-friendly system.

8 Pair Programming

Since the Oculus Rift was located on campus and some developers weren't familiar with Unity, most of the development was done in pairs or even groups. Because of this, almost all of the code in the project was understood by multiple people, improving the shared knowledge base and preventing one individual from being a lynchpin in the system.

Working with multiple people allowed for productive discussion about different ideas, methods of implementation, and debugging. This led to many implementation options to explore and quicker defect resolution. Not as much time was wasted on small errors such as grammar or syntax, instead this time was allocated to resolving larger issues that the team was facing. With the wide range of skills and knowledge, every member was able to learn something new during development.

As those involved in the developer-tester rotation expressed above, the decision to do the majority of our programming as pair programming was a positive experience, and didn't reduce the productivity of the team overall.

9 VR Programming

9.1 How it felt to learn VR

Working with the Oculus Rift along with Unity was both exhilarating and frustrating for group members. It was exciting to have the opportunity to work with new hardware that is in increasing demand. The Oculus Rift is an expensive piece of hardware, requiring a high-end workstation to run, and many members of the team would not have the means to purchase one and experience VR development otherwise.

Team members found the experience frustrating mainly due to the lack of integration with Unity built-ins, along with the lack of resources available online. Working with cutting-edge technology meant that there were not a lot of well-known best practices, which resulted in a lot of learning through trial and error. User testing was a huge

part of this project. We had to learn a lot of the best practices in a virtual environment through the feedback we received from users.

9.2 Impact on development environment:

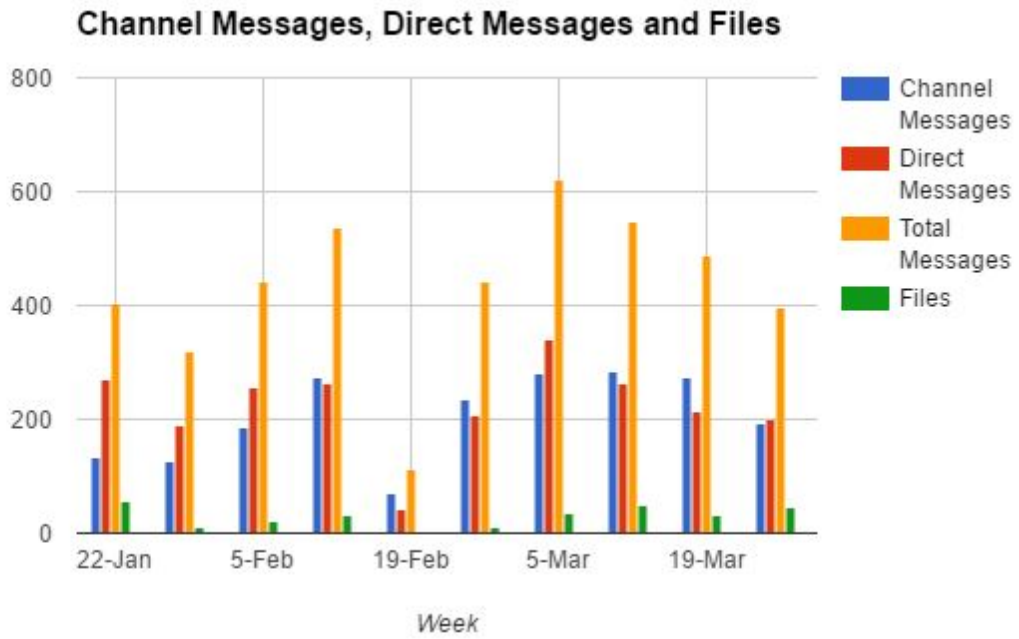
The nature of the project had a significant impact on the team's development environment. Because the Oculus Rift hardware could not be removed from the U of S campus, that meant that a significant chunk of development and almost all testing could only be performed with the team physically in the lab; after development had progressed to using the headset, no one was able to do much development or testing work while at home or traveling. The hardware and workstations had to be shared among other software teams at the U of S, such as the CMPT 406 Oculus team, reducing our development time further.

Working with cutting-edge technology came with the risks and difficulties we expected. Packages, SDKs and updates to Unity and Oculus did not have a long history of development, which made it more difficult to diagnose issues, and sometimes led to situations where fixes and patches for our problems were mere days old. Resources we could consult were limited, and documentation was thin compared to more established frameworks. There were also no experienced developers we could turn to in person when we encountered problems. We did our best in the project to anticipate these difficulties and manage the risk behind them.

10 Slack Statistics

Our team communicated primarily in-person and through Slack; other methods such as email and SMS were rarely, if ever, used. This helped keep communication centralized, and provided records for us to consult in case of confusion. Over the course of the project, we sent about 6000 Slack messages, 55% in public channels.

Activity increased prior to deliverables, but only slightly. Our rate of communication was consistent across the project, with relatively steady numbers of messages per week; the one exception is mid-February, during the reading break, when several team members were traveling or ill.



One unexpected element that allowed our team to gel was the use of Slack's emoji reactions to messages. We initially started using these as confirmation that important messages had been read, but over time it evolved into a morass of memes and inside jokes that brought levity to urgent bulletins and status updates. This might not have been 100% professional if viewed by outsiders, but given that the Slack was for internal use only, there were no drawbacks to using this system.

We broke our Slack up into several channels, to separate concerns, but a few group members expressed confusion about what should go where. We should have had clearer goals for some channels. There was also some concern that members who fell behind returned to find a large number of messages, getting overwhelmed and not absorbing important information they missed while they were away. This could have been mitigated with better awareness of absences and follow-up by the project manager and subteam leads.

Our development process itself led to fewer Slack messages overall. Pair programming increased face-to-face time between developers and testers, which reduced the need for electronic communication.