Approachable Code: Developing with Abstraction in Mind

CS4991 Capstone Report, 2023

Ian Harvey
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
imh9wz@virginia.edu

ABSTRACT
The Student Game Developers club at UVA, being largely focused on group-based work and comprised of students with a variety of skills, required game scripts that were both functional and easily approachable by less experienced developers. To simplify the use of these scripts, I utilized code abstraction and produced scripts that required no knowledge of underlying code. I exposed public methods and serialized private instance variable fields within the Unity game engine editor that could be accessed and edited by a developer before runtime. I then altered the name for each item displayed within the Unity editor to clarify the purpose and effect that each method and variable would have on the values in the script. Due to this abstraction, both non-programmers and programmers unfamiliar with my code were able to more easily grab the scripts I had developed and apply them to other areas of the game without my direct oversight. This increased the productivity of my team and helped reduce the effect unfamiliarity had on their confidence in their skills. In the future, I will need to write comments and keep better documentation within the scripts themselves so that, in the case of a malfunction, I will not be the only one capable of fixing the code.

1. INTRODUCTION

Technology has become the Atlas that holds the world upon its shoulders. Many of the proceedings of one's daily work-life environment revolve entirely around code written by individuals whose face most will never know. This code is expected to be functional, adaptable, and, most importantly, applicable by these external users without ever looking at a single word from it. Therefore, a layer of abstraction hiding the inner workings of the code must exist between the user and the developer in order for others to be able to adopt said code.

This problem exists for both non-programmers and individual developers. The average person is not expected to understand the underlying code of their applications. However, for other developers, the use of libraries in modern day programming serves as a kind of black box for them to send their input and receive the correct output without understanding what is happening within. This, again, demonstrates the need for the original developer to add a layer of abstraction when writing their code. If they wish for others to use their code, they must produce code others do not need to understand to use.

2. RELATED WORKS

According to Gross and Kelleher (2009), one of the main reasons people without much programming experience tend to struggle to understand code is that unfamiliar methods are difficult to interpret at first glance. In their study, they evaluated the course which such individuals took when trying to understand the code of a program. When it came to method interpretation, participants would not correctly

map value to methods and variables which had been improperly named or presented. This poses the danger of an individual completely misinterpreting the purpose of said methods and variables causing them to be unable to understand their use cases. My approach to my problem utilized this fact and applied it to the development process. Inside of the Unity engine, important variables can be serialized and seen by non-programmers as they apply the scripts to objects within the engine. In the frequent case that names are poorly chosen, it is likely that an individual will not understand how to set said variables within the editor, creating a hard stop in approachability.

Sadowski, et. al. (2015) discussed the developer aspect of unfamiliar code, outlining the process a developer takes to explore and understand it. They found that of the ways developers come to understand code, sample segments of similar code are the most prevalent. This information can then be applied to the team-based development environment of the club as those who are experienced with Unity can produce such code. So long as their code is reasonably readable, it can be used as an example for others to base their code off, making the code more approachable. This, however, relies on the assumption that the sample code written follows standards for readable code, which can be achieved by conducting periodic refactors of the codebase.

3. PROJECT DESIGN

The following sub-sections outline issues present with the club's structure and my proposed solutions to said issues.

3.1 Past Issues

The following sub-sections outline issues present with the club's structure.

3.1.1 Club Composition

The Student Game Developers club at UVA is primarily comprised of individuals of varying skillsets brought together for the common purpose of developing a video game. Many new members do not have any previous game development experience. New members can be split up into two sub-groups: Those who know how to program; and those who joined the club for more artistic reasons such as providing art or sound design. These new members contrast with older members that are highly experienced in working with and coding in the Unity game engine, the club's game engine of choice. Learning how to use Unity itself takes significant amounts of time and dedication. Thus, it is somewhat implausible that individuals with lesser skills will be able to contribute as much as they may desire in the semester-long lifespan within which each game has to be completed.

3.1.2 Code Quality

The code being written by those with more experience in the engine is often presented as hyper niche and completely closed off from the rest of the functional elements within each game scene. This is due to either poor naming practices of the fields exposed or the writing of unnecessarily private methods inaccessible beyond the code of the script. Because of this, newer users will see these scripts and believe that they are not allowed to be used beyond their declared purposes. They treat brand new code as if it were legacy code and refuse to approach it which results in some programmers writing near identical scripts and non-programmers fearing their ability to apply what they have developed to the game. Code bloat is quickly introduced, time is wasted on an already tight schedule, and many individuals simply leave the club because of their fears of inadequacy. All these factors significantly reduce the likelihood that the final product at the end of the semester is anything like what had been initially outlined.

This significantly harms both the reputation of the club and the morale of those who participated throughout the semester, greatly reducing the likelihood that people return to the club in any subsequent semester.

3.2 Code Management

The following sub-sections outline my proposed solution to these issues.

3.2.1 Standards Definitions

In order to fix these issues, the project I directed was regulated under a system of code management. A system of code standards was established, requiring code in the main branch to follow certain patterns. 1) All public and/or serializable variables have a unique and comprehensible name; 2) All methods and method fields describe their exact purpose; 3) Certain re-usable scripts have generic, less specific names to indicate their use flexibility; 4) Interface classes and sample interface templates are used for developers to copy as a base; 5) Code syntax and whitespace are similar between files; 6) Scripts interacting with similar facets of the project are placed in designated folders. These changes make the code easier to find and read both inside and outside of the scripts being written.

3.2.2 Standards Enforcement

The GitHub repository which we worked under was set up to echo repositories of industry standard. Protections were placed on the main development branch of the repository, forcing all members to produce their changes on separate branches. These branches could then be reviewed and edited to force conformity to code standards before reaching the main branch. In the case that the code did not conform to standards but was considered "close enough," I would either edit the code or message the author with clarifications and/or suggest looking at similar scripts.

3.2.3 Code Awareness

Each week, during our team-wide meeting, I would present and explain the new scripts that had been developed, including the purpose of each script and when/where they could potentially be used within the game. I would then show examples of how they had already been implemented within the codebase and answer any questions team members had about them.

4. RESULTS

I evaluated success on the observation of three quantities. First, I evaluated the rate at which non-directors were able to add code to the main branch of the repository. Previous projects within the club would have an average of 5.333 merges from non-directors to their development branch per month. Within one month of my project, there were a total of seven merges to the development branch from non-directors. The increase of these values served as a good indicator that individuals were comfortable delving into the codebase that had been constructed despite not necessarily being familiar with it.

Second, I evaluated the retention rate of individuals new to the club throughout the project's lifespan. On average, the club has recorded 8.25 programmers per project in the past who do not contribute anything before the semester ends. For my project, a total of 11 members did this. This would seem to indicate a failure in regard to retention rate, but at the same time this project was conducted in the fall semester when the club expects a higher membership drop due to a higher initial member count.

Finally, I evaluated the number of times I had to step in and help a member complete their assignment. Nearly every time a member was asked to complete something for the project, I would have to step in to help them out. Most, if not all, of these issues seemed to stem from

an unfamiliarity with Unity rather than the code that I was providing. When it came to the code I was providing, most members were capable of understanding each script they were using after I pointed them in the right direction and gave them a brief explanation of the purpose of the script.

5. CONCLUSION

My project has explored methods of reducing the barrier to entry in the particularly niche field of game development. However, my project's discoveries overall have a greater application to organizations with complex barriers to entries in the programming field in general. The methods I have applied increase the likelihood of individuals attempting to contribute to a project in a field they have no experience in. While I do not achieve the express goal of rapidly turning new members into independently contributing programmers, their exposure to any form of work on these projects enhances their abilities. The application of formal programming standards and overall greater code awareness could prove to be a beneficial practice to apply to any programming club to increase member interaction.

6. FUTURE WORK

While successful with the increase of contribution, my project still fails at creating an environment where code can be used and understood without assistance. Further work on this project would entail finding a solution to this aspect. In order to fix this, new members would have to be provided with a way to understand existing code more easily by solely looking at the scripts the code is written in. Some potential solutions may include a better method of maintaining documentation of the code being written. This mainly entails writing more comments which are more descriptive of the exact purpose and functionality of the code within each script. Furthermore, it may entail the implementation of Javadoc-like documentation to make code descriptions more easily readable through a browser view page.

REFERENCES

Gross, P., & Kelleher, C. (2009). (tech.). *Non-programmers identifying functionality in unfamiliar code: strategies and barriers*. Retrieved September 27, 2023, from https://openscholarship.wustl.edu/cse_research/19.

Sadowski, C., Stolee, K. T., & Elbaum, S. (2015). How developers search for code: A case study. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 191–201. Retrieved September 27, 2023, from https://doi.org/10.1145/2786805.2786855