Computer Science 3307
Ontario Municipal Benchmarking Initiative
Software Design Report

Group 4:
Spencer Hanley
Skylar Jonkers
Travis Laporte
Taylor J. Smith

December 5, 2013

Introduction

This is the design report of group 4 (S. Hanley, S. Jonkers, T. Laporte, and T. J. Smith) for the Ontario Municipal Benchmarking Initiative (OMBI) software project, created as a part of the third year Computer Science 3307 (Object-Oriented Design and Analysis) course at the University of Western Ontario. This report will discuss the software design choices made by our group in designing the OMBI software as well as the group's thoughts on the direction and outcome of the project.

Design Patterns Applied

Our group's project primarily made use of the Model-View-Controller (MVC) behavioural design pattern. Our code was structured in such a way that one set of classes primarily contained and worked with user data and data handling (`main, mainprogramview`), another set of classes was primarily dedicated to the logic of the software (`ombidatabase`), and a third set of classes was primarily used to create the user interface (`introwindow, graphwindow`). The reliance on this design pattern in particular allowed our group to separate similar code into workable, manageable "chunks" so that the process of development and debugging went much more efficiently. Our group also found that the use of MVC alongside the Qt framework allowed us to spend more time on developing the software, rather than fiddling with the user interface. In addition, every group member was already very familiar with the MVC pattern, allowing us to apply our prior knowledge and begin development much more quickly. Our group also incorporated the state behavioural design pattern into some parts of our software. Specifically, when a user is reviewing data from a certain measure, the state pattern is used to determine whether the tabular data or

the graphical data should be displayed. As well, the state pattern allowed our group to display or hide particular options to the user depending on what information was available to them at the time (for instance, displaying the visible/complete mean buttons below a graph). The use of these two design patterns allowed our group to work very efficiently to produce an excellent program that is easy to use.

Opportunities to Apply Design Patterns

Though our software made good use of some classic design patterns, there is still some room for improvement, and our group realizes that other design patterns could have been used to an advantage in select areas of our software. For example, the builder creational design pattern could have been used to some extent to create more customized graphs. By separating the construction and the representation of the graph object, we could allow the user to customize their graph by changing colours, altering data points, and so on. Another possibility was the use of the strategy behavioural pattern to allow the user to manipulate data. By using this pattern, our software could allow multiple encapsulated algorithms to operate on data independently. For example, allowing the user to output data in not just `.csv` format, but possibly `.tex` or `.pdf` for easy document creation. Finally, it might have also been feasible to incorporate the composite structural design pattern into our handling of service areas. For example, it might be possible to select one service area and then compare multiple service measures within that service area, viewing the service area as a "tree structure" with service measures as children of the parent service area. In the end, though many big ideas were thrown around, our group chose to focus on the essential functionality of our software while also leaving the code open and extensible for the future.

Strong Design Areas

In our group's opinion, the strongest part of our software design was the data handling capability of the software, as well as the simplicity of our user interface. Some group members are concurrently enrolled in a third year databases course, and so this project was an excellent way to apply the knowledge of database query languages such as SQL to a real-world situation. The use of databases to store and maintain municipal data was very handy, as it allowed us to manipulate and display data very easily in both tabular and graphical formats. In addition, the ability to save and delete user filters is an excellent addition to the software and makes for a simple way to retrieve previously used data. We were also very pleased with the simplicity of our user interface, and how we managed to achieve it. The MVC pattern mentioned earlier was very valuable to our project as it allowed us to fine-tune the display of our information and data, while also allowing us to focus on more important aspects of the software, such as the underlying logic. These aspects of our software in particular were our personal highlights, and we feel that they help to lend a greater value to our final product.

Weak Design Areas

While our group may be slightly biased about the quality of our program, we do accept that some aspects of the software could be improved. Aside from the unfortunate fact that our software did not fully implement some features, one major improvement that could be made to the software is in the scalability of data provided by the user, as well as the manner in which the data is displayed. Currently, our software can only support a certain maximum number of cities and years to be displayed at one time in either tabular or graphical formats. This is due to inexperience with the

development environment and tools, as no group member could successfully scale either the tabular or graphical data for all use cases. Though we had originally hoped to extend our software to support unlimited data, time constraints forced us to limit display abilities. Throughout the development process, we recognized that OMBI has many more service areas, measures, and years of data than what we were using, and so we strived to make our software as adaptable as possible. However, our efforts fell short, and so the software is somewhat limited as a result. As well, the creation and handling of data graphs is very finicky, and took a considerable amount of adjusting in order to display properly. We are particularly unhappy with these aspects of the program as we feel that, given enough time, we could greatly improve these issues. While `QCustomPlot` was a very handy tool, it had a rather steep learning curve and as such took quite a long time to fully understand and use. We feel that further experience with this tool could definitely produce better results. Though there are countless aspects of our software that we are generally pleased with, we feel that, given the time and the additional experience, we could remove the small blemishes and produce a near-perfect final product.

Conclusion

Overall, this software project was a very interesting insight into the world of professional software development, and an excellent way to apply the skills and knowledge we have learned over the course of the past term. Our group dynamic was very strong as we all had prior experience with working together, and so our software benefited as a result. The fact that our software was based on and uses actual municipal data was also a satisfying thought, and we hope that OMBI finds our project very valuable for their organization.