Reflection

FDM06

This reflective essay will evaluate how the FDM05 and FDM06 project was managed, along with the positive and negative experiences within this. It will also explore how the challenges faced were approached and overcome, and what thought process was used in making important decisions throughout. The essay will use this analysis to assess the learning outcomes achieved in relation to many of the learning objectives, and what can be taken forward from this experience into designing software solutions in the future.

Initially, two designs were proposed to expose a wide range of architectural opportunities, which strayed from focusing too much on the specifics of their implementation language. This approach allowed exploration into good low-level design and good practice (monolithic) as well much higher-level considerations (microservices).

The major challenge in using these strategies is that placing such a great focus on architecture retracted from implementation details in the microservices approach. The idea was to use architectural patterns and processes as a major comparison point, but the additional time taken to design a complex high-level system meant that many low-level details were skipped.

However, this made for an interesting contrast between the two designs. On one hand, there is a proposal with simple architecture and a well-considered codebase, on the other hand, there is a proposal with detailed architecture but a theoretically simple implementation.

Despite this, there was a form of parity between the two, where easy comparison could be drawn between components of each design, but they just sit at different levels in the architecture. For example, many of the classes in the monolithic approach were one-to-one representations of a microservice in the opposing design. As a result, this challenge in itself brought up an interesting decision to forget about UML diagrams and leave implementation up to interpretation as and when features were built.

As the microservices design was chosen in the end, the decision to leave these details out of the design was refreshing. The process was very reminiscent of a real-world agile environment, whereby unit tests are written to support a requirement and the code is written to pass those tests. In addition to this, the risk of oversight in the design phase is completely removed as no time was wasted worrying about programmatic logic until it really matters.

Another challenge faced in the design phase was the comparison of many conflicting factors in elegance and efficiency within software design, and evaluation of which are the most integral in accordance with the given functional requirements. The main struggle was trying to understand the importance of elegance and efficiency in relation to the project requirements.

The first step to overcoming this hurdle was making use of various academic and online sources to determine what factors make up these areas. This research led to new ideas of how these factors can impact the development process, the final solution, and most importantly, one another. These findings were used in conjunction with assessment of the project and its specific needs. The nature of which pointed towards a performant application without much need for the benefits in which elegance metrics provided, as this was an application that would never exist in a production environment with long term support. However, this process was viewed with a real-world lens to try and ground the subsequent decisions in reality.

This experience has embedded the importance of consideration into the context of a given project, both functionally and from a development perspective. It was found that there is importance in considering the answers to such questions:

- Who will be writing the code?
- What is the deadline?
- What is the software development life cycle?
- Is performance integral to the application?
  - If so, use relevant benchmarks to test against.
- Who is the user?
- Will they be frustrated by low performance?

To decide which software solution would be implemented, elegance and efficiency needed to be evaluated for each. It was a challenging process to assess and compare these findings, as it was tough to determine how and why a design meets the given criteria. The extreme differences in the design proposals led to an even more difficult comparison process.

Both designs were created to meet the design goals of which they did in their own ways. However, at this point there was a decision to make a trade-off and choose which design goals were more crucial. In the end, a more reliable, secure application was more desirable. This was mainly due to the nature of the application and how the integrity and safety of customer data must be paramount above a more elegant codebase.

This decision process shed light on what really matters in software: the user. Building a great product is important to customer retention, and poor security can ultimately lead to legal trouble. There are always trade-offs to be made, and software developers will always aim to write good code, but certain factors cannot be avoided or ignored.

Once a design was chosen, it became evident that the software solution was extremely complex. It was a challenging task to build and thoroughly test such an application. Much of the complexity was self-inflicted through over complication of various components. Admittedly, the client-side codebase took a long time to build, as much of the functionality was 'nice to have' and wouldn't have been missed from the final application.

Once this realisation was made, it was important to take a step back and re-evaluate the priority of the remaining tasks. The development approach was reorganised to make use of a Kanban board. This meant that only one feature or part of a feature would be built at one time. It helped to remain focused on the task at hand, by breaking each functional requirement down into manageable user stories.

It is important going forward to work in a granular way, by making purposeful steps toward to final goal. It is too easy to believe there is plenty of time in the beginning of a large project, and even when setting smaller internal deadlines, it can be difficult to stay on track. Tasks need to be manageable, and the extra time it takes to break requirements down into 1-day tasks can be saved twice over by an efficient development. process.

Upon completion of the application build, it was found to be difficult to effectively test the performance of the application based on the test cases that were built. There were various unforeseen barriers as the benchmarks were not grounded in reality, and while it was not integral to reach these milestones, more in-depth research should have been performed to understand the processes in achieving the set benchmarks. In the end, the software solution was subject to many bottlenecks induced by the host system.

Upon reflection, this issue could have been better approached by making full use of the benefits provided by a microservice architecture. Containerisation of each service and database would have allowed for the system to be distributed between physical machines or throughout a cloud platform such as Azure or AWS. In addition, this approach would have allowed for use of more 'quality of service' features such as load balancing or autoscaling of services.

Tackling the project with a kind of 'architecture first' mindset should have placed these features at the forefront and building to support these in future would benefit application performance capabilities. It is worth noting that there are simpler ways to build architecture to support high concurrency and low latency, and these are things that must be assessed alongside the rest of the design to ensure deadlines are sufficiently met. To reference points raised earlier, it is important to build high-priority features first and build 'nice to have' pieces when time permits.

Overall, this project provided plentiful opportunities to learn, both through research and practical application of these theories. There are many growing considerations to be made in the application design phase; with ever-changing technology and new ideas there is always debate to be had in the worst and best ways to solve problems. This project has highlighted the importance of instinct and contextual information to make decisions as there is no correct way to build software, but there are many ethical considerations that cannot be ignored. And finally, as ever, project management is a key factor to the successful delivery of a software solution. Sacrifices must be made, or no software product would ever be deemed complete.

1403 words