Understanding "Best Practices" Within Software Development

An Honors Thesis (CS 498)

by

Eli Sokeland

Thesis Advisor

Huseyin Ergin

Ball State University

Muncie, Indiana

April 2021

Expected Date of Graduation

May 2021

Abstract

Software is ever evolving and growing consistently throughout the world. With these changes, there is also progress within the teams behind the development and maintenance of these programs that power the world. In order to understand the effects of the current state of "best practices," in which industries use these frameworks, this project put different features to the test. From CI/CD to containerization, the project utilized newly popular frameworks, languages, platforms, and tools to the best of its ability for developer productivity. By integrating these features and studying the morale of the development team, the results showcased how much the team fluctuated in their own confidence. The research showed that many of the team members felt overwhelmed with the amount of technology that had to be juggled for the simplicity of the product. With a lack of knowledge for new technology along with a lack of technical support from the community, it wasted developer hours and lowered the team's overall morale. Additionally, this report was confirmed with other discussions from other senior capstone projects as well.

Acknowledgements

I would like to thank Dr. Ergin for helping assist with the capstone development teams and being willing to push them to broaden their development skills. I would also like to thank Digital Corps for giving me the confidence and ability to act as a leader among the development field. I would also like to thank those who helped contribute to the study and the project itself. Lastly, I would like to thank my family, who has supported my academics since I can remember.

Table of Contents

Process Analysis 1
Introduction 4
Methods 6
  Application 6
    Requirements 6
    Design Process 6
    Development 7
    Code 7
    Discussions 11
Results 16
Discussion 19
Conclusion 20
Sources 21
Appendix A 22
    Disclaimer 22

Process Analysis

The EduSource Rating Service project was originally assigned to me by my capstone professor. Prior to the project, I held no understanding of EduSource, the client, and what the application's purpose was intended for. However, I knew that I was selected for this project because of my initial knowledge and passion for web development. Specifically, I have had experience within React, CSS, HTML, PHP, REST, and WordPress before the project. However, I had yet to try building anything outside of these areas. Once I saw what the project process would entail, I submitted the research topic and began researching through experience with the development team.

The initial interest for this paper came from my interest to learn and to better understand the developer community. Within my experience of computer science, I have experienced many different perspectives about applying different "best practices." Specifically, I've mainly had two contrasting development roles in the past that pushed me to ponder the difference between the benefits of each system. For example, my first role was within the development of the *Canning Heroes* game was built with the Unreal Engine and Scrum methodologies. Second, I have spent time as a lead developer within several projects at Digital Corps using the Waterfall methodology. These two contrasting processes led me to the current research topic of finding what drives developers using these differences in "best practices."

Throughout the project, I executed the project as the lead software architect of the group. What this meant for the project was that I made the final decisions for the project. For example, I created the final design for the project, I created the initial code frameworks, and I created different parts of key functionality within the software. These key pieces of functionality include,

but are not limited to, creation of ratings, creation of users, authentication of users, authorization levels, and modification of user information.

Overall, the development process for the project was difficult because of the time between iterations. With each progress milestone for the team, the meeting morale swayed greatly, and it was difficult to keep track of how their code progress translated to their morale. To ease the pressure of the team balance and to gauge better ways of aligning with the team, I met with Evan Fischer, who was the mentor for the project. Through these meetings with Evan, I understood that there are limits to the motivation of an overall team. The best way to adapt to a team, is to understand the failures of each task and how to adjust for the future.

Looking back, the project did turn out to be a learning lesson for every member of the team, including myself. While I did learn a new framework, I also learned how to prioritize a team to better suit their own skills. Additionally, being able to discuss the product with the actual client provided a unique perspective that I have not seen before. With these discussions, it allowed me and the team to be realistic about the goals of the project. At the end, the client was satisfied with product and our team performed every task that the client did request. Also, the project helped expand my knowledge and acceptance of the idea of "best practices." Originally, the topic was focused on the perceptions and how these translated to real world developer attitudes. As time went on, I discovered that these practices were not as strict as I initially viewed them to be. Just because a practice is used throughout an industry, it did not make it imperative to complete any project. Overall, the project and the research itself was a success. It was a satisfying experience to know that the application the team created will help an actual company better their employees. In addition, the research helped validate my own knowledge as a computer science student who grew confidence in knowing the different techniques needed to

create collaboration within a team environment. By creating a compassionate environment, the key to clarity and success can be created within a field that can be easily seen as inhuman. By understanding the limits of technology and my own knowledge, the project pushed me to grow and adapt to new situations as a computer scientist. By the end, I know that I can stand behind the ideals of what it means to push for developer advocacy when there are difficult challenges. Overall, this project taught me that I can provide a unique perspective as a computer scientist using an open and empathetic attitude.

Introduction

The EduSource Rating Service was originally a senior capstone project brought to my development team near the end of September 2020. The application is an entirely new program for EduSource, who was the client for this project. Originally, the project was created to serve as a similar replacement to the JIRA system used in industries today. Specifically, the software would be able to have employees create ratings based on other employees based on their proficiency of a specific category. These categories could also be created by employees as well. For example, an employee could rate another a 4/5 on their JavaScript skills. Additionally, managers would be able to view employees' ratings over time. By doing so, the system is built to promote employee growth by nurturing good skills and fixing issues within categories where employee proficiency is lacking. However, what made this project unique was the number of unique integrations that were attempted within the project. According to the 2020 StackOverflow survey, the majority of developers work within the field of web development (either in frontend or backend). Therefore, there are countless developers who are passionately changing the landscape of web development on a daily. For example, StackOverflow has conducted this survey on an annual basis since 2015. One question that has remained through each of these is the "Most Loved Language" of developers. Looking at the 2015 survey, neither TypeScript nor JavaScript made it within the top ten most loved languages for developers. In comparison, the 2020 survey listed TypeScript as the second most loved language along with JavaScript in tenth place. Other popular technologies listed with dramatic growth include PostgreSQL, Docker, Vue, and React. With this capstone project, I wanted to examine how these new technologies would affect developer productivity and morale.

The ability to conduct this research was more fluid than anticipated. Technology within a project can change pretty quickly. However, with the use of the Agile methodology, I conducted team discussions about their mentality after each major iteration throughout the project timeline. For each interview, I would ask each team member their thoughts on the past iteration and their personal confidence in the project. After that, I would ask about the usage/knowledge about the current tech stack used within the software and the thoughts of different integrations within the system. The main focus from these questions was about the confidence and struggles that the team faced overall and how these pieces of software influenced their own mentalities. Therefore, most of the data collected from the research was qualitative in nature.

Methods

Application

Requirements

The EduSource Rating Service was a full-stack application built from the ground-up, so there were a series of requirements that the client requested. The most important aspect of the project was that there was a user system where individuals could rate each other on their proficiency on a selected category of subject. These categories, along with the users, needed to be managed by the users deemed, “administrators” and “full-time employees” of the application. Additionally, the client requested that those with higher authority be able to view other employees’ ratings and their averages throughout time. The last functional requirement, the client specifically requested this application to run on the Chrome browser, being built completely in TypeScript. In terms of non-functional requirements, the client requested that the product would conform to the Material Design standards and be usable on mobile devices.

Design Process

The application process started with the discovery meeting. This meeting was a client meeting focused on what the client envisioned for the product and how it could be beneficial to the company. After the initial meeting, there was a Unified Modeling Language (UML) diagram for the domain model, a summary of requirements, a list of use cases, the tech stack, and the initial mockup was drafted for review.

These documents were first reviewed by Evan Fischer, the mentor for the project. After some feedback, changes were implemented and showcased to the client. With minor feedback on the design, the first iteration requirements were established with the client.

Development

The EduSource Rating Service was designed as a web application for the EduSource company to use for internal purposes. Specifically, the application was split into two aspects: the frontend and the backend. The frontend was focused around the React application architecture, with the use of TypeScript Extendable Markup Language (JSX) files. React was a component-based library that returns HTML-based syntax in a virtual DOM. Additionally, the library integrated with MaterialUI, which was a design library that integrated Material Design into React with CSS-in-TS. This was unique because it allowed developers to craft their components without the reliance of using dedicated CSS/SASS files.

The backend application was developed using the Nest framework. This framework allowed the development team to integrate authentication and the GraphQL language in a modular fashion. Additionally, to assist with GraphQL integration, there was also an additional integration of Prisma. Prisma was an object-relational mapping that was used to ease building queries on the backend application.

Code

Shown below is a piece of sample code from the backend. Specifically, this code provides the authentication for how to create a rating for a specific user. This code can be found at the URL https://github.com/jrbeutler/rating-project-backend. Additionally, the rest of the code and documents for this project can be found at https://github.com/jrbeutler/rating-project-frontend and https://github.com/jrbeutler/rating-project. This code is important for several reasons.

First, the code showcases two pieces of functionality that I exclusively worked on during the project. One piece of that software was the authentication system for users to verify their

login sessions. This included password validation and being able to return user information based from a token passed as well. Additionally, with that verification, I created the functionality where users can create ratings for other users. This verification can be seen with the line "@UseGuards(GqlAuthGuard)." With this code, there are also some underlying important principles.

While I have programmed with the REST programming interface before, this application introduced new concepts that pushed modularity and innovation. Within the application structure, the overall backend utilized GraphQL, NestJS, and an object-relational mapping. As seen within the rating creation function, the application called the create function within prisma.rating. This is unique because in a typically library, functions are already defined and not open for modification upon a programmer using the library. However, within Prisma, these functions are created upon compilation to match the proper schema. This presented unique challenges because of the abstraction away from native SQL statements. While this did ease security concerns with SQL injection, it limited the flexible nature of SQL (particularly in table joins).

Overall, the code merged many modern concepts in order to meet the needs and requirements of the clients. With GraphQL, schemas are highly important and need to be translatable to the frontend. By embracing new frameworks such as NestJS, GraphQL is welcomed by default into the ecosystem. However, with any framework, there are limitations and concerns. The more precise the software and its libraries, the harder the integrations and maintenance became for the application. However, the application itself is well functioning and completes everything that the client requested.

```
//rating-project-backend/src/services/auth.service.ts
async login(email: string, password: string): Promise<Token> {
    const user = await this.prisma.user.findOne({ where: { email } });

    if (!user) {
      throw new NotFoundException(`No user found for email: ${email}`);
    }

    const passwordValid = await this.passwordService.validatePassword(
      password,
      user.password
    );

    if (!passwordValid) {
      throw new BadRequestException('Invalid password');
    }

    return this.generateToken({
      userId: user.id,
    });
  }

  validateUser(userId: string): Promise<User> {
    return this.prisma.user.findOne({ where: { id: userId } });
  }

  getUserFromToken(token: string): Promise<User> {
    const id = this.jwtService.decode(token)['userId'];
    return this.prisma.user.findOne({ where: { id } });
  }
```

```
//rating-project-backend/src/resolvers/rating/rating.resolver.ts
@UseGuards(GqlAuthGuard)
  @Mutation(() => Rating)
  async createRating(
    @RatingEntity() rating: Rating,
    @Args('data') newRatingData: CreateRatingInput) {
    return this.ratingService.createRating(newRatingData);
  }
}

//rating-project-backend/src/services/rating.service.ts
createRating(newRatingData: CreateRatingInput): Promise<any> {
    return this.prisma.rating.create({
      data: {
        User_Rating_reviewedIDToUser: {
          connect: { id: newRatingData.reviewedID }
        },
        User_Rating_reviewerIDToUser: {
          connect: { id: newRatingData.reviewerID }
        },
        Rating_Category_categoryToCategory: {
          connect: { id: newRatingData.categoryID }
        },
        rating: newRatingData.rating,
        notes: newRatingData.notes
      }
    });
  }
```

```
//rating-project-backend/src/resolvers/rating/rating.module.ts
import { RatingResolver } from './rating.resolver';
import { Module } from '@nestjs/common';
import { PrismaService } from '../../services/prisma.service';
import { RatingService } from '../../services/rating.service';

@Module({
  providers: [RatingResolver, RatingService, PrismaService],
})
export class RatingModule {}
```

Discussions

While the end goal of the project was the product for the client, it also served as a testing ground for the team's personal challenges in a real-world project. To gauge the effectiveness of the applied technologies, two forms of data were collected: interview data and survey data. Regarding the discussions, these were focused on developer morale and were conducted on an iteration basis. Those who were interviewed were three of the four members of the EduSource Rating Service project, two members of the trust management capstone project, and one member of the offline video editor capstone project.

During the discussions with the members of the EduSource Rating Service, there was a pattern of knowledge, research, and application that fell into repeated procrastination. During the meetings of the first iteration and the second iteration, three of the members requested clarification on the core concepts surrounding the frontend and backend architecture. When the interview talked with each member on the team, there was little to no experience with web development before. Comparatively, each member did have experience within Java and Python,

but admitted that the switch to a new language with a new framework was a difficult transition. Additionally, the team faced issues regarding integration of new technologies.

At the beginning of the project, the team brought up the key point of stability within modern frameworks. The project was originally intended to be developed in Vue and TypeScript. However, the Vue version requested was still in beta until early 2021. Therefore, the configuration on local developer machines presented issues that caused the team to switch to React. According to Joshua Johannsen, a developer on the team, stated that, “When we switched to React, it was a lot easier to learn. It was a lot more straightforward than Vue.” This was a key building base for the team. Additionally, there were other challenges as well.

Docker and testing are key pieces of software that can greatly assist with software quality assurance. However, integrating these services can prove difficult depending on the overall software structure and application. When interviewing Clayton Mercer, who was a developer for the trust management application, he stated that his team decided to ignore Docker. Because the project was a mobile application, the developers used local smartphone emulators on their machines to achieve the same effect. In comparison, the EduSource Rating Service was able to successfully use Docker for local machine testing of the application. Regarding testing, there was a greater consensus on how testing was completed through each project timeline.

While there is a variety of testing forms within software development, the usage of each type warrants serious discussion within each team. Within the EduSource Rating Service team, there was attempted user interface (UI) testing and end-to-end (E2E) testing for the overall application. However, due to knowledge and time constraints, the team never managed to successfully integrate into the software architecture. In comparison, the offline video editing software team integrated some parts of testing. According to Hunter Line, who was a developer

on this team, he stated, “UI testing was minimal because we mainly used libraries that weren’t meant to be tested. Additionally, we didn’t do much unit testing because of our lack of familiarity with the code.” He stated that UI testing would have been similar to testing Xamarin (the framework the team used) itself, which is already tested by the maintainers of the library. Lastly, the teams looked back at how their mentality was affected by the software they developed.

Within the initial EduSource Rating Service team, there was a sense of imposter syndrome that came with the amount of knowledge required by the project. Looking back, Nicholas Burrell, who was a developer on the team, stated that, “I think a majority of best practices are good, but unit testing is a little difficult to warrant. With the class, it was understandable because we worked on a small team, but there is more in the development pipeline in an actual company that ensures that code is kept up to quality.” This was a key perspective that showcased how project sizes also affected the use of modern practices. Additionally, within the survey that was sent out to the rest of the Ball State computer science capstone groups and within the IndyHackers group, there was an interesting perspective as well regarding these integrations.

Within the survey conducted within this group, there were a total of 13 respondents. The questions that were asked were the following:

1. Which development methodologies have you used/currently using (checkbox answer)?
2. What aspects of these methodologies do you enjoy? What aspects do you not enjoy (short answer)?
3. What area of software engineering do you currently practice (checkbox answer)?
4. How does your techstack affect your productivity (short answer)?

5. Do you use testing (unit/integration/ui) in your software (yes/no answer)?
6. How does testing affect your work productivity (short answer)?
7. Do you use CI/CD in your software (yes/no answer)?
8. How does CI/CD affect your work productivity (short answer)?
9. Do you use containerization in your software (yes/no answer)?
10. How does containerization affect your work productivity (short answer)?
11. When do you think that it is acceptable to adopt a new library/language/framework? What kind of requirements do you look for (short answer)?
12. What are your thoughts on the state of modern development (short answer)?

While software development has grown rapidly over the past, there is a greater sense of mixed messaging that has left the community confused about the direction of software development. For example, frontend applications can be built on the basis of just HTML, CSS, and vanilla JavaScript. One humorous example within the industry is the story of RemoteOk.io, which is a website that allows companies to post and connect with potential employees who would like to work remotely. The team behind the creation of this business was one man: Pieter Levels. According to his Twitter account (https://twitter.com/levelsio), the whole application is built on one PHP file. While this may be absurd, it does complete all the necessary requirements that make it a successful website for remote job findings. One respondent to the survey conducted stated a similar notion when asked about the state modern software development:

> Modern development is all over the place. The industry has a list of 'best practices' but they are only contextually useful. I think as an industry we're (very slowly) learning that product development can't be reduced to a list of steps to follow, or even a list of practices to follow, although some people will fight that tooth and nail despite the growing evidence.

> It's a mixed bag. The industry doesn't seem to know what direction it wants to go. Most everything is moving to web/cloud computing, but there's no definite answer of what modern/near-future web development should look like. Frameworks like Ruby on Rails still prefer monoliths/large backend applications while "lightweight" language like Go are proving useful for microservices, which is great when considering front-end development concepts like adopting the JAMstack to improve performance. Additionally, there's some blurred lines in job descriptions now. It used to be that you could "specialize" in frontend, backend, devops, etc. But without enough software engineers (which there will always be a shortage of), there's an unspoken need for everyone to become a jack of all trades who can work on application logic and tread water into devops to deploy said application logic. It's particularly jarring when I first graduated college 2 years ago where I focused on writing web applications to suddenly having to learn concepts like PaaS providers such as AWS or networking/application tools like Kubernetes.

This respondent showed that while new technology is exciting, modern development has blurred the lines of developer roles and duties. Instead of writing purely user facing logic, a frontend developer would also need to understand DevOps to fully contribute to the team in a proper manner.

Results

From the capstone projects and the discussions conducted across teams, there was a variety of mixed emotions regarding the juggling of multiple areas of knowledge. According to the survey, all of the 13 respondents have found that modern software tools and techniques like Docker, testing, and CI/CD can be beneficial to modern projects. However, there is a gap in the knowledge, usage, and theory behind these areas.

Based off of the qualitative analysis behind each of the groups interviewed during the study, each one of them admitted that lack of knowledge was the biggest factor behind motivation within the project. For example, Joshua Johannsen stated, “If there was one thing that was above all else in fear, it was how far behind I felt. Everyone else I think had something to bring to the table and I have no experience to bring to this. The lack of experience had me worried. I want to feel like I had something to contribute.” This, along with the initial inability to integrate a framework in a stable manner, proved that these were the key foundations behind a project.

Additionally, the use of integration and testing was an important factor for the projects as well. Within the discussions conducted with the trust management system and the offline video editing system, the issues regarding integrations lied within the type of software used. Since both projects were natively based, any form of Docker would not have been useful to the project. Additionally, the use of some frameworks causes different aspects of testing to be redundant.

Lastly, to finalize a project’s proposed set of tools and processes, it requires additional support from around the community. For example, the usage of the website StackOverflow is synonymous with developers sharing their questions and answers to difficult scenarios. Because of this, the EduSource Rating Service team found it difficult when there were not any previous

developers creating a unique integration that blended the Nest framework with GraphQL and testing within the same system. Additionally, this area of support was also the majority of reasoning from the survey for choosing a new tech stack element when starting a project.

Therefore, the results from the research are a proposed triangle model of needs (inspired by Maslow's Hierarchy of Needs) that a project must achieve within its development team for the project to successfully use its tools to the fullest. At the base of the model is the foundation of knowledge and stability. If a team is completely unaware of the language they are going to use, there must be a path for the team to gain knowledge before the project gets started. Additionally, the project tech stack must have clearly documented steps of instruction that can be easily troubleshooted if there are any issues.

For the next layer of the developer needs, there needs to be a form of agnostic capabilities within the software for it to be understandable to developers. By only relying on the simplest implementation, developers should have easy access to add and remove third party libraries and tools that extend an application's sophistication.

Lastly, tools and modern practices need community support. While there are countless unique projects being developed on a daily, the standardization of many web frameworks today has provided key steps for developer progress. With the continued push for open-source communities within the web and mobile, this is guaranteed to expand even further. Therefore, it is essential for developers to have clear areas of knowledge that they can look towards for any assistance.

While this triangle hierarchy of needs provides a basic sense of cohesiveness within a team, it's not a guaranteed means of success. Every project has it downsides, such as procrastination within the teams that were interviewed. What the goal of best practices is that

they provide a basis of learning for developers to feel comfortable with new technology. While there is no official list of best practices, there should not be a list. Best practices are formed through experience, which helps strengthen a developer through time.

Discussion

While the project did have occasional challenges, both the product and the research did confirm some initial beliefs regarding best practices. Initially, best practices were thought of as a layer of tools, processes, and architectures that every team must have conformed to be successful. However, best practices are more fluid than initially expected. The type of application, for example, can have a major impact at deployment and containerization. Additionally, because of this fluidity, there is greater sense of multitasking within the developer community.

Within the EduSource Rating Service application and within the other capstone groups, each member of the team was expected to know about the entire tech stack of the project. While this is understandable for small teams, the push for DevOps within frontend and backend roles for developers has caused developers to struggle between too many different fields of study to understand software. However, the reason why these new areas are being pushed is the growth of open-source simplicity.

With the survey from developers regarding modern development practices, there was optimism within the community. While it has been difficult for DevOps in the past (such as configuring servers completely and handling deployment), the introduction of Docker and CI/CD has allowed for easier management of software. However, there will almost always be knowledge gaps within development teams. Therefore, it was beneficial to see that these processes can be formulated into official steps within companies if they deem it essential enough. Overall, the strictness of best practices lies within the development teams of each company. With diligence and proper guidance, best practices can save countless of hours of development time on testing and deployment. The right process just requires team collaboration.

Conclusion

The project and research showed that, while integrating some best practices and modern tools could assist with development, they are not always required. Within the EduSource Rating Service, there was significant amount of time and energy poured into testing. However, testing was never implemented by the time the product was handed to the client. Despite this failure, the client was still very pleased with the outcome of the project. This showed that developers should always focus on the functionality first and try to find the best tools that enable the team to meet any project goal.

Sources

1. StackOverflow. (2015). “2015 Developer Survey”. StackOverflow: https://insights.stackoverflow.com/survey/2015 [2015].
2. StackOverflow. (2020). “2020 Developer Survey”. StackOverflow: https://insights.stackoverflow.com/survey/2020 [2020].

Appendix A

Disclaimer

Those who were interviewed and consulted for the research of this project gave their consent to use their words for this thesis.