ECE 458: Engineering Software For Maintainability

Senior Design Course

Spring 2015

Evolution 1 Analysis

Brian Bolze, Jeff Day, Henrique Rusca, Wes Koorbusch

Contents

1 Introduction

Good software design is often seen not only as a science but as an art. It is a craft and, like any other type of engineering, is only mastered over time. The fundamentals of good software design, however, remain sound. In this senior design course, we plan on synthesizing our four years of knowledge through the development of a robust and long-lasting software application implementing a web-based calendar. We plan on applying the core principles of good design to our code and to our design process, while continuously evaluating, refining, and improving on our skills of the software engineering trade.

For our project, we built an application using the Ruby on Rails framework. This framework aided us in developing highly modular and reusable code due to the MVC architecture and Rails' powerful web application stack. With the additional help of great documentation and a strong community, we were able to develop a functional product within weeks, despite minimal domain expertise. While our current application has noticeable areas for improvement, our robust model, well thought out API, and extensive front-end templates promise a maintainable foundation for future development.

2 Project Plan

2.1 Design Goals

In order to set ourselves up for a successful project, it was really important to us that we lay out our design goals and priorities before moving forward. We aimed at grounding our design discussions in the fundamentals, which we hoped would help us set up a solid foundation for a project of such a significant size. The core design principles that we focused on primarily were modularity (Open-Closed Principle), re-usability, and the DRY principle. Another consideration that we put a lot of thought into was our language and framework choice, which will be expanded on in the next section. Besides promoting good code design, a well-chosen framework would allow us to get up to speed quickly, which was another important criteria for our team given our experience level with web applications. During our initial stages, a particular focus was placed on developing a robust, accurate, and extensible model for our data. This would lay the groundwork for our API with the front-end components, which we saw as vital to our success down the line.

2.2 Language Choice

The language decision was extremely important for our group for a few reasons. Firstly, no one in our group had previous experience in developing web applications. Accordingly, we stressed finding a language and framework with a strong community, good documentation, and abundant resources. Secondly, seeing as our project would be maintained over an entire semester, we strived to find an environment that would support and promote good design practices. To do this, we would need a language with strong object orientation and a framework with a highly modular architecture. Lastly, we also needed the flexibility to implement our application with high customizability to meet any obscure requirements. After extensive research and testing, we found that Ruby on Rails was able to meet our needs better than any other framework.

2.3 High-Level API

stuf stuf stuf

3 Design Review

3.1 Status

stuf stuf stuf

3.2 Design

As a result of utilizing the Ruby on Rails framework, we separated our program into four distinct large scale sections: the Model, View, Controller, and Database. A majority of our first evolution design consisted of developing our models and database schemas from the model. In the design of our model, we wanted to make sure that each component we created could have a clear relationship to every other component that was necessary in the design requirements. In order to make these relationships flexible and dynamic, we took to heart a computer science principle that has been driven into us by many of our professors: the solution in a level of indirection. One key idea that we believe helps significantly with easily allowing one user to follow another user's calendar is the subscription model. Instead of directly associating a list of other users and events of those other users that a single user can view on the calendar, we gave each user many subscriptions that cleanly encapsulate the subscribing user, the user being subscribed to, the event being subscribed to, the visibility of that event, and the ability to send email notifications regarding that event. We created a similar model for managing groups called memberships that help to relate various users to various groups. These two models gives us the flexibility to easily and logically add memberships to groups and visibilities while being able to apply rules to them.

One strong aspect of our overall system design is the ability to easily create new views for every aspect of the model. Using the rails framework, a single terminal command can create controllers for every aspect of the model along with associated views for creating, deleting, editing, and viewing model information. However, currently our implementation repeats a lot of code for user forms and other aspects of the view. In order to fix this problem in the future, we will attempt to utilize what are called Partials in order to more easily reuse aspects of our front end design.

3.3 Alternate Designs

stuf stuf stuf

4 Design Process Notes

4.1 Designed and Conducted Experiment

Brian's Contribution

One of the components of the project that I worked on heavily was the model. One of the few annoyances of Rails, however, is in how it separates out its model files from the underlying database migration files. This made it very difficult to keep track of the attributes and associations with each of the models. In order to ensure that the work on the model was accurate and would not break any other parts of the system, I decided to write a collections of tests. One of the benefits of RoR is the integrated test environment. It allows a developer to write tests for models, controllers, and other components at varying granularities, as well as seed the database with sample data using *fixtures*. After writing a suite of tests for the model and running them against a few fixtures, I was able to quickly identify and resolve some errors in the way we were performing associations in the model. After running this experiment I was motivated to continue writing tests for controller components and others, which we can check in on continuously.

Jeff's Contribution

Once the team finalized our schema for the database, we began developing individual aspects of the entire design on each of our own git branches. However, we noticed that when testing newly added features on our own individual branches, the database would for some reason roll back to an older version of the database that would cause the website to break. For a very long time, we thought it was because the database schema and model files were being transferred over to new branches incorrectly. However, after trying to implement new features on both individual branches and on the master branch, we realized that in order to update the database layout using rake db:migrate (the rails command to automatically update the database) on an individual branch, both the migrations and the database files needed to be committed to master whereas we were only committing the schema files. After running the experiment of developing on two different branches and seeing the different outcomes (along with some help from stackoverflow), we were able to solve this bug using an experiment.

4.2 Analyzed and Interpreted Data

Brian's Contribution

Most of the analysis and interpretation I performed during this stage of the project was in the tests that I wrote. One of the tests I wrote for the User Model included generating 1000 new user emails and passwords and ensuring that all 1000 saved successfully. Even with encryption of all passwords, the test was able to complete in less than a second, which was a promising result when considering the deployment environment where there may be hundreds of new users signing on every second.

Jeff's Contribution

Included in the rails framework is the ability to write tests for all of the different models that we create. In order to confirm the functionality of our models and the associated database,

we developed a number of test files that would create instances of our models, manipulate them in some way (add, edit, or delete specific fields of the model), and then display the result. In confirmation of our model design and function, we developed different data schemes to test and analyzed their outcomes to make sure that our implementation was sound.

4.3 Designed System Component to Meet Desired Needs

Brian's Contribution

One of the core components that our team collaborated on heavily was the data representation in the back-end. Henrique and I were the only two members with experience in database programming, so we were able to guide the discussions in the right direction. After many white-board sessions, as a team we were able to establish a robust model representation that we later translated into a series of Ruby models and database migrations.

Jeff's Contribution

Our team as a whole spent many hours designing our database schema and model to make sure that our design had the ability to meet all the desired needs specified by the evolution document. Although not every single requirement was implemented perfectly by the specified deadline, we our confident that our design deals with every possible relationship necessary to facilitate the completion of the requirements.

4.4 Deal with Realistic Constraints

Brian's Contribution

One of the crucial design considerations for our group was the selection of language and framework for our development environment. No one on the team had had much (if any) experience with web development, so given the time constraints and project requirements, it was important that we used an environment with extensive libraries and resources, a large community, and good documentation. Initially, we ended up starting with Django and Python due to two team members background in Python. However, due to the reasons stated above we came to realize that Ruby On Rails met our needs far greater than Django and Python, and justified our switch.

Jeff's Contribution

The fact that we made a decision part way through the development process to switch from a Django framework to a Rails framework made the time of total development much smaller. This restraint required us to quickly design our new model in Rails while still having to consider good design principles in what we decided to implement.

4.5 Contributed to Team Work and Interacted with Team Members

Brian's Contribution

I often acted as the mediator and organizer of the group. I set up a Trello page to help us

with Project Management and task delegation. I also hosted most of our major hacking sessions at my apartment, which helped foster collaboration in the group. I also attended all meetings and contributed greatly to the design and implementation of the back-end model, business logic, and model-controller API.

Jeff's Contribution

I attended every single team meeting and highly contributed to the initial model design. Specifically, I helped white board out the model and collaborated with Henrique on integrating aspects of how events are displayed and how the overall look and feel of the application should be.

5 Next Steps

One key aspect of our current implementation that needs to be fixed is code reusability. In rendering views in Rails, we can implement Partials which allow us to specify a piece of repeated code, and refer to the partial in the actual view files instead of the actual source code (basically like creating methods for rendering front end pieces of the application). This allows for less lines of code in general and better module design. We can do the same thing with the controllers of the application using what are called Filters. Implementing Filters and Partials into our framework will result in fewer lines of code to get the same job done and more modularity in our design, two key aspects of good design that we discussed in class.

We also need to finalize some of the front end views for the Evolution 1 requirements. Although we have implemented the appropriate models and database schemas, we unfortunately did not have enough time to implement an elegant front end for interacting with rules and repeating events. These must be finalized before moving on to significant aspects of the Evolution 2 requirements.

Finally, we want to embellish our user interface more to make it look more elegant. Because each member of the group has had very little front end experience, we initially focused more on the back end design and database implementation. Our front end implementation we really only wanted to be functional. For our final product, we want our website to not only be functional, but cool and fun to use. More research into Javascript abilities and front end frameworks like Angular.js are probably necessary.