Design Patterns and Django MVC

Over the past four semesters in the software studio track, I have learned many things that have helped me grow as a software developer. Each semester, I have worked on a real world web application with a group of qualified student developers in an agile team, and I have had numerous opportunities to apply many software engineering principles that many software companies practice. One significant concept I have learned over the past four semesters is the implementation of various design patterns, particularly the Model-View-Controller (MVC) architectural design pattern using the Python Django framework.

In the second semester of being in the software studio track, I read a software engineering paper about design patterns. The concept of design patterns is relevantly new in the study of software engineering. One can summarize a design pattern as a general, repeatable solution to commonly occurring problems in software design. Design patterns are not a finished design of a software application. Rather, a design pattern is a description or a template for how to solve a particular problem ("Design Patterns" 1). Several design patterns I recognize in code I have written and utilized in software studio include factories, prototypes, and interpreters.

There are several advantages to understanding and implementing design patterns. Software developers should utilize design patterns as a form of communication to share ideas
of how to construct a piece of software. When used as a form of communication, design patterns prevent sublet issues in development and aid in code readability among developers as well. This elimination of miscommunication by using design patterns aids in decreasing development time during the software development process ("Design Patterns” 2).

While design patterns aid in communication and development in an agile software development team, there are some disadvantages to using design patterns. To some software developers, including Paul Graham, design patterns are a faulty solution to a software problem that is not properly abstracted. Moreover, computer scientists like Peter Norvig have simplified or eliminated the need of twenty-three design patterns with solutions written in Lisp and Dylan programming languages ("Design Patterns” 3). Some software engineers further criticize that design patterns are inefficient solutions and require more formal definitions in the study of software engineering.

While there is validity to the challenge that design patterns may not be the most fully abstracted approaches to software design problems, from my own interactions with fellow software studio students, design patterns are beneficial for me to communicate my ideas of how to design code for web applications. In addition, I advocate a formal collection of accepted definitions of design patterns to aid in the communication of software solutions from professional software developers.

One imperative design pattern I had learned before entering the software studio track is the Model-View-Controller architecture. Software engineers define MVC as a concept of encapsulating application data and isolating it from manipulation and presentation. The model represents the data of an application and is not dependent on the controller or the view. The
view displays the model data and sends user actions as HTTP requests to the controller. Finally, the controller provides model data to the view and interprets the HTTP requests sent by the view. The controller is dependent on the model and view.

The MVC design pattern provides several advantages in developing web applications. The MVC design pattern provides flexibility in software development, particularly in modularity of data and presentation. Page design and the user interface are not always dependent on each other, and modifications to the data model are independent to other data models in the application. This low coupling of data and presentation simplifies the development process and affords easy maintenance of a software product (Grove 1).

There are some disadvantages to the MVC design pattern. If a software development team wishes to make a MVC web application, the team must strictly follow the design of the MVC framework. The view and controller components of many MVC frameworks are tightly coupled making modifying the controller and views complicated at times. In addition, while models are separate from views and controllers, changes to the models have the potential to hinder progress of development by requiring software developers to modify the views and controllers as well (Grove 2).

The MVC architecture design pattern is a powerful concept. MVC has revolutionized how I view developing web applications and other software products. Separating the design of data models and the design of web page layouts and application logic is vital in planning tasks to complete. Furthermore, MVC forces me as a software developer to better understand the structure of an application no matter what MVC framework on which I decide to develop. MVC causes me to clearly understand the requirements of a software application in order to avoid
painful side-effects of not properly representing data or business logic. I recognize now that MVC helps software developers to be more intentional about understanding the structure of the software they are developing.

In all four of my semesters in software studio, I have created web applications by utilizing the Python Django framework. According to the Django online documentation, Django is a high-level Python web framework. Django, like other popular web frameworks Rails and Erlang, implements the MVC architectural design pattern. Django emphasizes the importance of the “Don’t Repeat Yourself” (DRY) principle when developing software. Django is helpful for agile software development teams for rapid development and is a popular open source web framework. Other various perks of the Python Django framework are serialization of data, unit tests, ease in handling forms, and middleware libraries that manage sessions and user authentication ("Meet Django" 1).

As stated earlier, the Python Django framework implements the MVC architectural design pattern. The models are stored in the models.py file of an application. Each model is a Python class that is a subclass of a specific Django model object. Implementing a distinct Django model class provides quick database access when making queries. The controllers are function based and are located in the views.py file of an application. This file contains function definitions that receive HTTP Requests and return HTTP Responses for specific types of site pages. Based on the specific request defined in the urls.py file, Django will run a specific function in the views.py file that will send the proper template and data. The view component of the Python Django framework are called templates. These template files are stored in template directories and are a combination of regular HTML text and Django template
language. A software developer working in the Python Django framework should not confuse the terminology that views are controllers, and templates are the views of an MVC framework.

To some software developers, there exist a few issues in Python Django web framework. One criticism of the Django framework is that it is monolithic, or is constructed in one large structure. The Django framework forces a software developer to create highly coupled applications within a project at times, especially when data or functionality is required from another application. Some software developers contend that Django can be overwhelming for a new developer because the framework requires a broad knowledge of how it implements MVC in structuring a web application. Finally, some software developers criticize the Django framework by stating that most, if not all, objects from sessions to persistent data is dependent on the Django object relational mapping.

Based on my own experiences of developing various web applications with the Python Django framework, I find that Django is indeed robust and requires much understanding before developing a web applications. I had to dedicate many hours to comprehending the framework and the best practices for developing web applications. However, once I had mastered my understanding of the Django framework, I found that web applications were somewhat easy to structure in the MVC design pattern. Furthermore, while there are some restrictions and difficulties for some web developers by the Django framework, creating web applications on the Django framework can be effortless and even easier to deploy in a short amount of development time.

In my fourth semester of software studio, the assigned project for the studio team was to work on the Dreamsuite application for Tiny Hands International. The Dreamsuite application
was created as a Django project. However, the Dreamsuite application implemented the Django framework in a different way known as Class Based Views. Class Based Views are an alternative way to develop web applications in the Python Django framework. The key characteristic of Class Based Views that a software developer should note is that controllers handle HTTP Requests using classes and not functions. Class Based Views are specific to HTTP requests the controller receives, and return the required template and data model based on the configuration in the urls file. Moreover, the Class Based Views utilize object oriented programming with the use of Mixins used to factor Python code into various components.

There are several advantages for a Django application developer implementing Class Based Views. Class Based Views provide flexibility for software developers by allowing subclasses for previously defined classes and redefining methods for specific application cases. Class Based Views also provide the affordance of clearer code and readability with class implementations of controllers. Class Based Views also follow the goal of creating web applications on the Django framework by following the “Don’t Repeat Yourself” principle when creating specific Class Based Views for a certain HTTP Request.

While there are some significant advantages to developing Django web applications with Class Based Views, there are some disadvantages. While Class Based Views enhance readability of code, often times Class Based Views require more lines of code to define a view class. In addition, there are numerous default Class Based Views defined within the Django framework. If a software developer does not correctly implement the Class Based Views, there is an increased chance for improper configuration or errors in instantiating a class.
Based on my own experiences with exploiting the functionality of Class Based Views in the Dreamsuite application, I believe that Class Based Views are a powerful alternative to the practice of Function Based Views in Django views. For me as a software developer, Django Class Based Views are easier to read than traditional Function Based Views. Class Based Views offer the affordance of flexibility to be specific for certain URL configurations. Furthermore, the various pre-defined Class Based View methods are helpful in manipulating common variables and processes like the context, form, and request variables. Class Based Views are indeed robust, and therefore require a higher level of understanding in order to avoid errors in implementing Django Class Based Views. However, I believe that when a Django web developer becomes comfortable with his or her understanding of the Class Based View definitions, that developer will find that Class Based Views offer more advantages when designing the architecture of their web application.

I find that the knowledge and skills I have exercised the past four semesters in the software studio track have helped me mature as a software developer. Through studying various design patterns, I have developed an enhanced vocabulary and understanding of various object oriented constructs in software engineering. Along with an enhanced vocabulary, have gained a new language to communicate various ideas to fellow software developers on development teams with whom I work. Having a deep understanding of the various components of the Model View Controller design pattern have helped me understand a successful approach to structuring a web application. Furthermore, the Model View Controller paradigm has been beneficial to my understanding on the importance in separating data, data processing, and data presentation in web applications in order to simplify the development
process. Mastering the Python Django framework and implementing Class Based Views has aided me in creating web applications for numerous software studio projects and personal applications as well.

From all of my experiences and study of the various topics aforementioned, I believe that successful web applications I develop in the future will need to implement well-defined structure from various design patterns. Furthermore, web applications I develop will in some way be modeled in the MVC architectural design, based on the benefits of separating data, logic, and presentation. While I may not always work on applications with Django framework, I have the skills and experience of a popular and successful framework to develop web applications. In software engineering, all projects require structure and design and my experiences with design patterns, MVC, and Django are advantageous to my future as a software developer to create well structured and designed software.
Works Cited


