Australian National University

Towards a Clean Architecture

For

TechLauncher Projects

Zhanhui Lin

U6022913

A Report submitted as the assessment for an Independent Studies 12 units

(COMP4560) in

The Research School of Computer Science

Australian National University

2019/5/30

Acknowledgement

It takes a long time to finish this report. I would like to show my gratitude to those people who have helped me and contributed to the project.

First, I would like to express my deep gratitude to Dr. Henry Gardner my research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. Also, I would like to thank Mr. Jay Hansen my research tutor, for his advice and assistance in keeping my progress on schedule.

I would also like to extend my thanks to the staff in the Research School of Computer Science for providing the resources of TechLauncher projects as cases.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Abstract

Clean architecture is a popular topic in the area of software development, that has been proposed by Robert C Martin, a.k.a. Uncle Bob in his book “Clean Architecture” [13]. The idea is to use a collection of design patterns to develop an architecture that keeps important properties of the software, such as the business rules, isolated from implementation details. By employing a clean architecture, the interconnections between software components can be kept small and manageable. This means that the software will be more easily modified and maintainable: whether in its database section, its use of code frameworks, or its user interface.

In this report, I will describe two TechLauncher case studies. These case studies are software for the “ANU Fifty50” and “FutureYou” projects. Both of them use the same framework which is Django, and these two projects are analysed via the concepts of software design patterns and clean architecture. Recommendations for ways in which the applications might be better structured are made.

Table of Contents

List of Figures

List of tables

Introduction

Introduction to software architecture

With the development and progress of society and computer technology, the demand for high quality software has increased dramatically. There is no doubt that the world is more and more dependent on software. Software has become an indispensable part of human life, whether in transportations or mobile phones. Even in traditional fields, such as supermarkets, organisations are increasing the use of software – such as trying to use AI robots as cashiers.

Software architecture aims to convert software characteristics into a solution which meets the requirements on the level of technology and business [1]. A software architecture defines the overall structure of an application. When people want to understand a software architecture, usually, they will try to find a diagram of structure which will have architectural layers, components, or distribution nodes.

There is an example provided by Peter Eeles on website “IBM” [6], the UML class diagram, in Figure 1, shows some structural elements. In this diagram, there are three classes which are called OrderEntry, CustomerManagement and AccountManagement. It is easy to see that CustomerManagement and AccountManagement depend on class OrderEntry. A software architecture also defines behaviours between each element in a software. In Figure 2 (a UML sequence diagram), the Sales object needs to use an OrderEntry object to create Orders first. Secondly, OrderEntry uses CustomerManagment to achieve a customers’ information and then uses AccountManagement to activate and place orders [6].

Therefore, software architecture plays an important role during the process of developing a TechLauncher project (or any other projects).

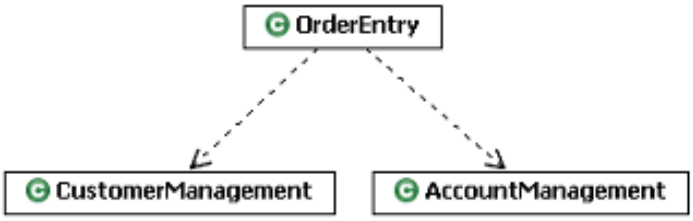

Figure 1. Class Diagram [6]

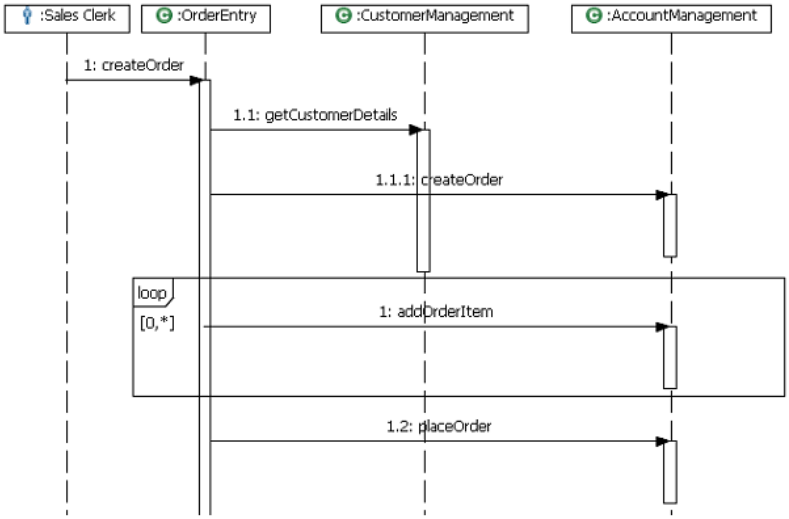

Figure 2. Sequence Diagram [6]

Introduction to TechLauncher

TechLauncher is a program that enables students to develop their research and professional skills, so that they can bring great ideas to life and make a positive impact on society by using technology.

This program trains student in their technical skills, communication skills, critical thinking, teamwork as well as skills of management. Students improve their skills by working in a wider environment outside ANU with industry professionals, technologists and entrepreneurs to complete projects.

TechLauncher is supported by local industry; the local innovation sector; the ACT Government; The Canberra Innovation Network; and ANU.

Motivation

Nowadays, many excellent software architectures exist, however, it can be problematic for people to choose a good architecture when they are developing software. Different software architectures have their own characteristics, and usually developers have little ideas about how to use a software architecture in a properly way.

My project aims to explore how to apply the concept of design patterns and clean architecture in real-world software and how to help developers to recognize how to make their software architectures better.

Specifically, I choose to study two TechLauncher projects that were completed in 2018: ANU Fifty-fifty and FutureYou. In this report. I will analyse these two study cases to show how design patterns and clean architecture might work to improve these two TechLauncher projects.

Background

Software architecture

What is software architecture

Software architecture refers to the high-level structures of a software system [20]. Each structure is interleaved with each other whether in properties or elements. It serves as the blueprint of a system and a developing project, providing the tasks which need to be executed by developers [19]. Without a good software architecture, which is the primary carrier, it will be impossible to implement all of the desired quality attributes (especially maintainability and extensibility).

Software architecture aims to convert software characteristics into a solution which meets the requirements on the level of technology and business [1]. It is the fundamental stage during the process of designing software. The architecture of a software determines how it works; how much it will cost; how long it will take from design to production. When the software architecture is changed, every part of software might need to be remodified at some expense. Therefore, a good architecture is important to the maintenance and evolution of long-life software.

Why software architecture is important

For an organization, a good software architecture enables stakeholders to have a better understanding of the project, and developers can keep communication with stakeholders to make timely and effective modifications which may avoid unnecessary redundancy such as time and money.

For developers, employing a good software architecture is the key to improving their competitiveness. Therefore, a good software architecture is so important when it comes to development. For instance, when stakeholders fully understand the software architecture of the project, they would make important decisions on this stage of development based on their requirements. It is a really essential stage in the process of software development. Because when poor architectural decisions are made, it is hard to make further changes to the continuing work.

The present project aims to assess the software architectures of two TechLauncher applications that were developed using the same framework. It is hoped that in doing so, it will be possible for the good aspects of the architectures, and modifications to them to make them ``clean", to be reused in other projects in the future [10].

How to judge software architecture

It is very difficult to determine whether a software architecture is good or not. Usually, if software performs well, and costs not much money and time to develop, stakeholders will think it has a good architecture. However, for developers and staff who are responsible for update and maintenance, a good software architecture also should be scalable and maintainable. Meanwhile, a good architecture is also friendly to users. In general, all parts of a good architecture are interrelated and dependent.

Here are some conditions to judge software architecture good or not for a developer [21]:

- Functional: Parts of the software that are functionally independent should structurally independent. This means each independent part of the software can be identified, modified and maintained separately.
- Robust: The software architecture should be strong enough so that it can be changed and maintained when there are bugs and failures. All of the problems occurred in this software architecture can be solved.
- Maintainable: Software would be easy to maintain. A good architecture, comments and modular design are extremely important which can save plentiful time and money.
- Reusable: Refactoring should be easy, when developers meet some problems.
- Extensible: Software should be able to add features, when stakeholders present some new requirements. That is why people say that a good software architecture can make a software live for a long life.

A good software architecture should not be only friendly to developers but also friendly to users. Here are some ideas about how to judge it from a user perspective [10]:

- Users adapt to it quite easily

- High performance and runs fast
- Easy to modify or add functionalities
- Easy to run tests on software
- Strong and reliable

Clean architecture

What is clean architecture

The clean architecture, as one of the most popular software architectures proposed and popularised by Robert "Uncle Bob" Martin, has attracted a lot of attention in last few years. It aims to make applications that have extremely low-coupling and are independent of technical implementation [18]. By using clean architecture, software becomes easy to maintain and modify as well as easy to test after it is delivered to consumers.

Clean architecture has an "onion-like" layered structure that makes an application have low-coupling. Figure 3 shows a visual representation of this layered structure. In this diagram, there are four layers, from inside to outside. These layers move from an abstract level to general level from the inside to the outside these are the entities layer, the business logic layer, the controller layer as well as the external interfaces layer [4]:

- Entities include enterprise business rules and depends on domain models.
- Use cases are application business rules; they depend on the inner layer of entities. Use cases represent business actions to achieve goals.
- External interfaces: The most appropriate framework for the whole software
- Adapter layer:
    1. Implements interfaces which are supported by use cases
    2. Control data flow
    3. Interacting with applications

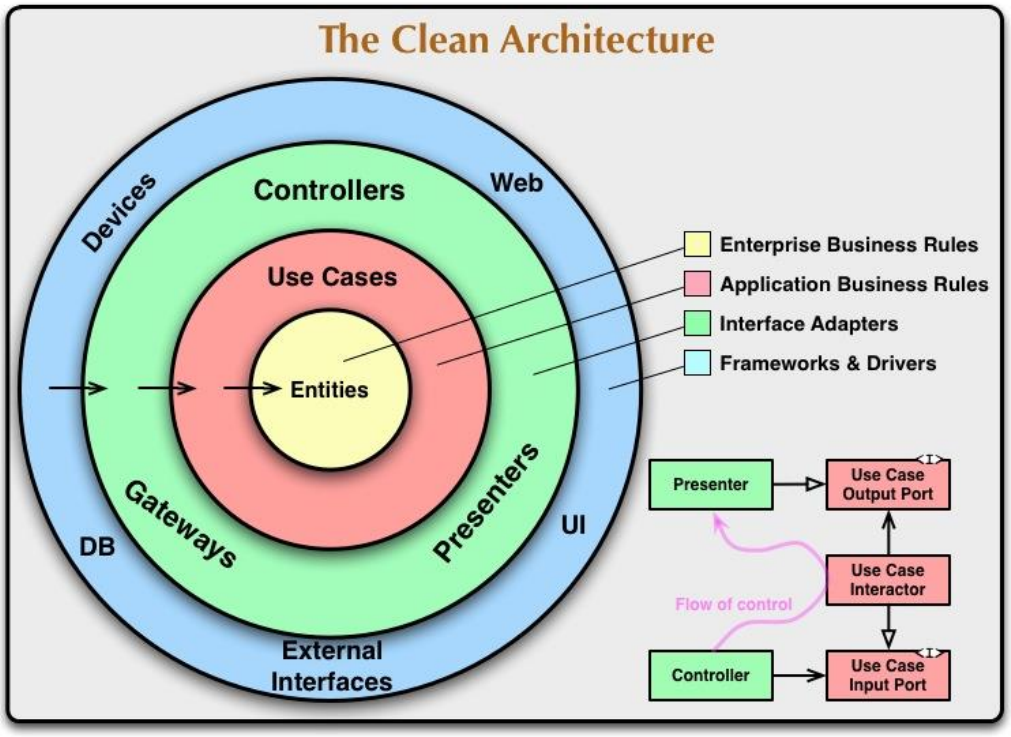

Figure 3. The clean architecture [12]

Features

The main idea of the clean architecture is "separation concerns" which aims for low coupling and independence of technical implementation details (such as databases and frameworks) [17]. It puts frameworks and databases in the outer layers, and business rules and logic in the inner layers [12]. Therefore, it can be simplified to Figure 4, which shows that the outer layer is supported by its inner layer.

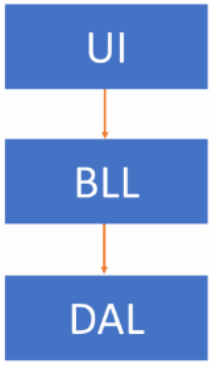

Figure 4. Sample layers diagram [12]

Therefore, there are some characteristics of clean architecture [3]:

- Independent of frameworks, which means the architecture does not depend on the existence of library of feature framework no limited constraints,
- Testable, which means business rules can be tested without the UI, Database, Web Server,
- Independent of database, which means business rules are not bound to database
- Independent of UI. UI can be changed easily and separately from the other parts of software and business rules.
- Independent of any external agency.

In this way, developers can maintain and modify applications effectively and efficiently.

There are many well-known software architectures which apply the basic idea of independency such as Onion architecture by Jeffrey Palermo (2008) and the hexagonal architecture by Alistair Cockburn and others (<2008) [2], however, Uncle Bob reorganized and refined them, and then provided the clean architecture. Therefore, clean architecture has many advantages when it is compared with other architectures. For example, service-oriented architecture is good for testing systems and for multiple implementations, but its cost is more expensive, while the clean architecture is much better which is testable, independent on frameworks and database, and easy to maintain as well as costing less.

Introduction to Django Web Framework

Django is a high-level web framework written in the Python programming language which is effective and efficient on the development and maintenance of websites [5]. It has an MVT (model-view-template design pattern) for those developers who do not have too much time for a high-load work, as well as, for those low-skilled and low-cost developers. Because of their boost team efficiency and their ease of use, Django is very popular in the field of software development.

There are some advantages and disadvantages of Django [8],

Advantages:

- Fast: Applications can be designed in an easy and efficient way by employing Django.
- Fully loaded: There are lots of extra functions which can help developers to make their products more powerful.
- Secure: It is a mature framework which will remind developers to avoid any potential secure mistakes.
- Scalable: It can meet the high demand and pressure from web visitors.
- Versatile: Good management on code.

Disadvantages:

- Everything must be based on Django ORM, excessive packaging which means a third-party package cannot be used
- Component are deployed together; there is a weak template which is not flexible for some developers to code with.
- Too monolithic

Case Study

Case Study 1: Future You

Description

FutureYou is an online gamifying career planning and development platform which is supported by ANU Student Experience & Career Development (SECD). It is a web-based application which aims to realize undergraduate students to be aware of their career development as well as appeal them to engage in their career life early. It is designed to help and track progress of students' career planning, meanwhile, teaching them how to make right expectations for their career life.

On this platform, students can input their career goals and any other expectations they would like to achieve, and they are also able to input the resources they need such as skills and preparation. Figure 5 shows a screenshot of the sample website.

Figure 5. Screenshot from FutureYou

Future You is written using the Django framework with the Python programming language to implement backend and using a website style for frontend. MySQL is used for the management of database.

Analysis

For this discussion, I consider properties of software following a method of analysis that has been described in the PhD dissertation by Roy Fielding, "Architectural Styles and the Design of Network-based Software Architectures" (in 2000) [7].

Performance

The performance of Software architecture can be described by three main perspectives which are functionality, structure and allocation of domain function [11]. We can identify the performance through the interactions between components.

- Functionality

The functionality means what the software does during implementation. For this case, it is a Web-application.

First, we simulate the flow of the functions implementation which is shown in the sequence diagram. When a student uses the application for the first time, he will register first and then enter his detailed personal information, such as name, university and career goals. And then, during this process, the application will generate his personal characteristics into a chart through a student's profile. And, whenever he completes a goal, he gets a reward.

At the functional level, we can see that when a visitor enters the home page, system calls url.py in the Django framework. Then url.py calls view.py. View.py then calls the class in model.py and html pages in the file which is called templates to implement.

For example, if the visitor would like to register, he will go to the signup.html page to register and call the class Student to store his personal information. When he completes the registration, the website will go to registe_success.html to notice that you register successfully and returns to the login page. After the visitor logs in, he should put in his personal information in detail such as work, major, plan, meanwhile, the class "UserProfile" and class "CareerGoal" is called to store these data.

In general, the functions implemented by this application are simple and clear, such as login registration, looking for employer, forming resume, and so on.

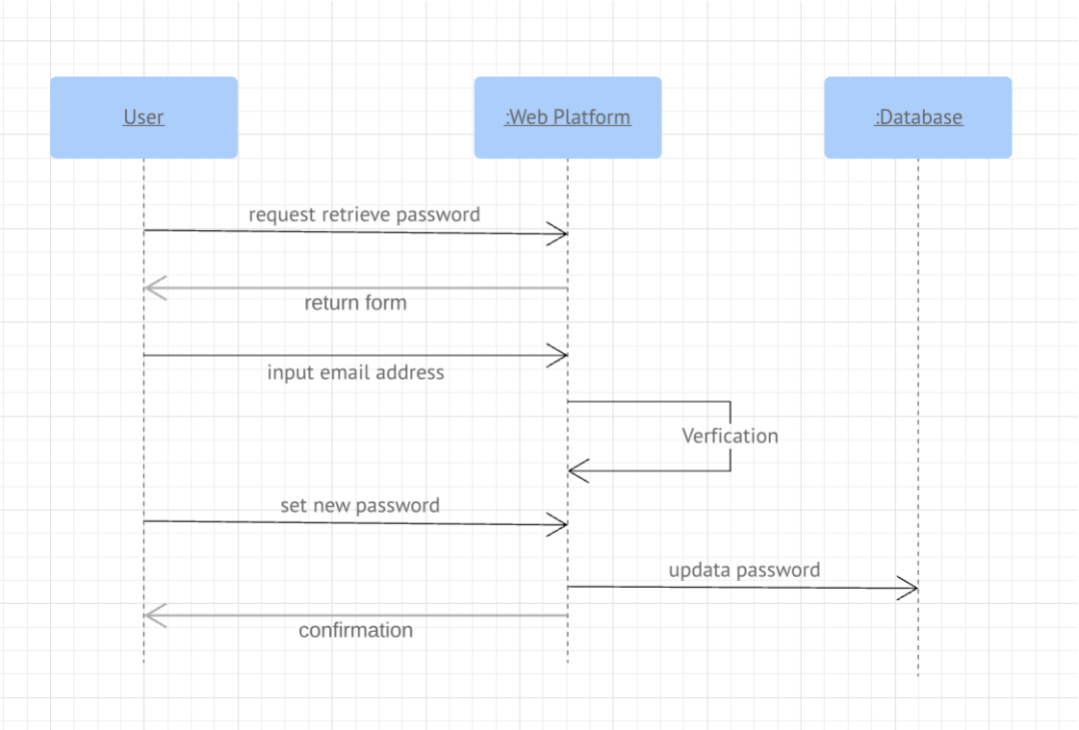

Figure 6. Sequence diagram of reset password for FutureYou

- Database

The diagram blew, provided by the FutureYou development team, it shows the relationships between classes. Table “Student” is related to other tables with foreign key it “studentId”. This kind of design simplifies the burden of the student table. By decomposing the student table, each web page applies with different tables to enter the database. The advantage of this is that when the database table is changed, it will not cause data confusion, and the unnecessary table can be easily added or deleted, which avoids the redundancy of data and the trouble of later maintenance.

For example, when a student uses “ResearchEmployer () “to query an employer, the application only needs to access the table “reseachemployer” to make changes to the Status property, which does not involve other tables. The independence between tables provides the basis for clean architecture.

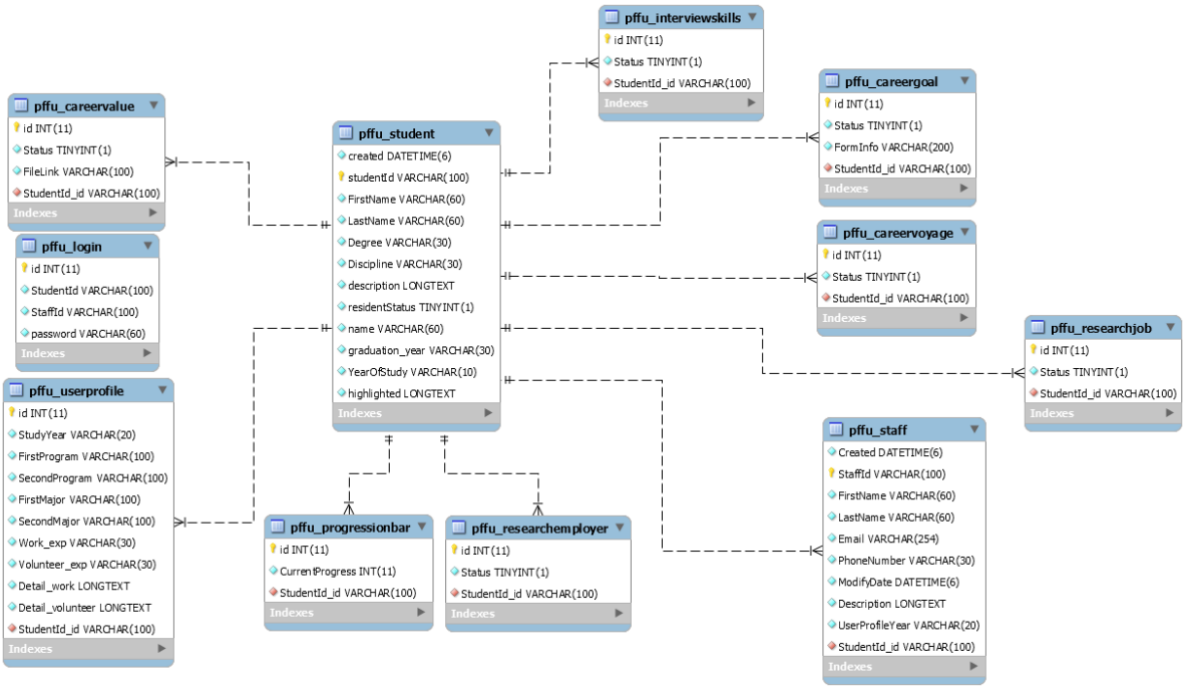

Figure 7. ER diagram for FutureYou (provided by development team)

- Allocation of domain function

The allocation of domain function happens when a user does a series of operations on the application. Whether the architecture of an application is clean depends on whether the functionality of the software is simple and fluent.

For example, let us assume that a student forgets the login password and chooses to reset the password. In this series of operations, first, the student pressed the button which is called “forget password”. Then, the system will send a link to the registered email address. After student clicks the linked page, there will be new password verification functions to make sure that student enters the same password twice. When the setting is successful, the system updates the new password corresponding to the student user name to database. Finally, the student successfully logs in with the new password.

In this series of resetting passwords operations, many functions are involved, such as sending a verification email to a registered mailbox, verifying that the two new passwords are entered consistently, automatically updating the database user password, and so on. This series of operations uses the most succinct way to change

the password, instead of the user looking for an administrator to modify the password for him, which simplifies the controller part in software architecture.

Scalability

Scalability means when the workload increases the architecture can be scaled up to meet them [9]. There are many ways to improve the scalability in software and hardware.

For the software, we can simplify the component and make the interactions distributed. In the FutureYou, we can find in the Figure 7, there are many tables linked to each other by "studentId". It maximizes the independence of each table. In this way, developers can add other tables to distribute the interactions when necessary. Moreover, when the number of users rises, the processing speed of the database on the system will also become a problem. If too many attributes are placed in each table or all the attributes are concentrated in one table, there will be confusion when users operate on the application. Suppose the system can process up to 100 people per table per time when processing database information and "password" and "status" are in the same table. If one hundred people change their passwords and another hundred change the status at the same time, that means there are 200 people who make changes on the same table in the database which will cause the system to get stuck. However, for this project, the developer has solved this potential problem well and prepared for the future improvement by simplify the tables. There is another way to solve the problem of scalability which is task parallelization. It also can be called multitasking which means tasks can be run on separate computers or CPUs on the same computer or separate threads on same CPU [9],

For the hardware, vertical and horizontal scalability can be used such as faster CPUs or better machines [9].

Modifiability

As the time passes, most of the software has to be upgraded to avoid being eliminated. Therefore, modifiability is an important property in the software

development which aims to enable applications to make modification instead of shutting down or replacing.

Modifiability means that there are no side effects on other components when the component changes. This is also the main idea in the clean architecture which is called "separation of concerns". To prove this property whether exists in FutureYou, we imagine that FutureYou needs to make some functional changes over time and see how such changes could be implemented.

In our first hypothetical scenario, FutureYou needs to upgrade the function which is called “research employers”, for example, automatically pushing a relatively appropriate employer to students based on their personal information instead of students do researches on an employer. In order to achieve this function, first, we need to modify the database. It needs a table which contains some attributes such as the company name, whether it is being recruited, the difficulty of being employed, and the number of people recruited.

In terms of programming, we can add a function, for example, we can do an automatic research algorithm which is managed by staff. This algorithm can retrieve all the companies that are hiring and conduct machine learning to classify companies, for example, big data in computer science field, artificial intelligence in computer science field, or financial and accounting. Then, this algorithm can output the required data, for instance, the difficulty of being employed can be calculated by the percentage of recruits per year. After that, these retrieved and calculated results will be stored in the database. This algorithm is executed manually by the staff regularly such as once a month. And the results are automatically recommended to each student who uses the function "search employer" based on the results of machine learning.

In this process, we found that this function enhances the usability of this application, but it does not have a negative impact on other components of the application.

In our second hypothetical scenario, FutureYou needs to add a new function which can make it performance better and more powerful. For example, the student's comprehensive ability score, this function will use data from some tables in the database.

The first step is that we need to use a table containing a series of personal information such as student grades, ages, majors, etc. Then, we even need to add some attributes to tables, such as students' hobbies. As mentioned before, using the machine learning method, the relevant information of students in the database is trained, and the student's comprehensive score will be calculated. The score will be displayed on student's personal homepage including the score details of various aspects.

During this process, database and code changes are involved. We add a function which is called “calculate_student_overall_score”. At the same time, we make some small changes to the database, such as adding attributes such as student personal interests in the "profile" table. However, these operations did not have any negative impacts on the entire application, while enhancing its functionality.

In a conclusion, we find that there is no side-effect when we modify functions on the application in these two imagined scenarios. Therefore, the FutureYou appears to be reasonably extensible.

Reusability

Reusability is a property of software architecture, which means that the main part of an application (data structure and algorithm) can be ported to another similar application without a lot of modifications [16]. This kind of operations not only does save time and cost, but it can also provide greater benefits for stakeholders.

For FutureYou, its system is similar to a blog, everyone has their own home page, personal information and photos on the home page. Its main function is to show users’ career plans and goals, and to assist users with some functional queries, such as "employer research." In this application, it uses some affiliated links, such as the website called "ANU Careers". Suppose that, we also use this application for the University of Canberra, we will find that we only need to make minor changes, such as changing this affiliate link to "UC Careers" or other links similar to "ANU Careers". This allows code reuse.

Based on the architecture and style of FutureYou, it can also be used for commercial purposes instead of applied to universities. For example, it can do workplace

planning for an employee of a company, and the changes need to involve some data structures in the database. However, the overall structure and functionality do not need to be changed too much. We can change some functions, for example, we can change the "research employer" function to "research the company you work for" to increase the staff's understanding of the company. At the same time, we can make some changes to the scoring algorithm mentioned above, and then apply it to the calculation of the employee's annual year-end bonus. This also reflects the company's fairness.

In conclusion, we can find that FutureYou can be used not only in university, but also in other commercial fields. Therefore, we can say that FutureYou is a reasonably reusable application.

Extensibility

Extensibility is a design principle in which extensibility is considered for future growth. Extensibility refers to the ease of adding and removing features to an existing application, which can be achieved by adding new features and modifying existing features. The more extensible the software is, the stronger the structure is. Therefore, when you modify the software with high extensibility, it will not affect the existing functions.

In this project, FutureYou is a relatively more extensible software. For example, its data structure and implementation of functions are relatively independent, thus, it reduces the complexity of the modifications. It uses the Django framework to implement some functions in HTML that implement the code implementation in Python. However, it has both advantages and disadvantages to doing this.

The benefit is that developers can easily modify or remove features by changing the code in the HTML file. For example, when a student needs to generate a resume, the developer can add some frameworks to present the student's characteristic attributes, which is more intuitive for the user. However, these are difficult to implement in Python. Because this requires the developer to program in Python first, and then store the data in the database, which is finally called by HTML. Therefore, using the Django framework can increase software independence and reduce development complexity to achieve clean architecture.

However, excessive reliance on the Django framework also has a downside. As mentioned in the introduction to the Django framework, it is not suitable for large, complex applications. Python can better implement these functions when the program needs to add complex functions, which HTML cannot do. For example, the function of the machine learning classification mentioned above, which cannot be implemented in html, is too complicated. Therefore, developers need to program in python and then call this function in html. And, when an application is too functional, the Django framework doesn't run it very well.

In short, for FutureYou, the application of the Django framework is very appropriate. It reduces the complexity of the structure and makes the entire application scalable, such as the addition and modification of functions.

Portability

Portability refers to the usability of the same software which is implemented in different environments [22]. In other words, portability means the convenience of applying an application to different operating systems. In this section, I will discuss in two aspects which are operating systems and data sources.

Regarding the operating system, we only discuss the operating systems that most people use, such as Windows, Mac, Android and iOS. Because FutureYou is a web application based on the Django framework, it doesn't involve a lot of mobile or computer operations and processing. It simply uses the web page to show users information and the results they retrieve, and it does not involve complex calculations. Meanwhile, including electronic devices such as mobile phones and computers, they come with a browser which can handle this web application easily. Therefore, for operating systems, this web application is convenient and portable.

Regarding the data source, in FutureYou, all the data is stored in the database, presented on web pages which applied to the Django framework and then to the database. Users can easily get the relevant information they want. Also, the calculation of the data is implemented on the background server. Therefore, in general, the data source does not have a negative impact on the portability of the project.

Reliability

Reliability refers to the extent to which an architecture is susceptible to failures in the event of an unexpected application. This is not just accidents that may occur in the design, but also unexpected errors that occur in the application, such as data errors, data loss, and so on.

In order to evaluate reliability, we can also imagine two hypothetical scenarios. In the first scenario, we assume that a table called "profile" has been corrupted in the database. The ensuing problem is the loss of user information which is a real serious accident. When the user's data is lost, it followed with the chaos throughout the entire application. The only solution to this situation is to periodically back up the system database, which is not mentioned and designed in FutureYou.

In the second scenario, we assume that the table "researchjob" which is not as important as "profile" in the database and corruption has occurred. We can solve this problem by using redundancy, such as adding a property called "rj_Status" which is same to “Status” in table “researchjob” to other tables as a backup of this table.

We also need to consider some factors, such as whether the error is recorded and whether there is an error message notification to users. For these factors, FutureYou did not consider it. The correct way is to create a record that will record each unexpected error and stop servers and make modifications and debugs in a timely manner. At the same time, users should be prompted with error notifications to avoid unnecessary disputes.

Summary

The following table lists the positive and negative aspects of this architecture.

Table 1:

criterion	***Good***	***Bad***
Performance	Good function design: clear and powerful	Some function on websites cannot be implemented
Scalability	Easy to scale up in both hardware and software	None
Modifiability	Easy to modify and no side-effects due to its clean architecture	None
Reusability	Can be reused for most of similar projects	Serval limitations exist for reusability
Extensibility	Relatively easy to extent	Some functions implement in both HTML and Python are difficult to modify
Portability	Small web-application and good portability on different operation systems	Usability issues for mobile devices
Reliability	Having notifications and solutions for expected issues	Not enough solutions to unexpected errors

Conclusion

In general, FutureYou is a project that is not fully completed, because some features are defined in the web page but not implemented on code level. Through attributes analysis in each web page template and related code, we can find that the architecture of FutureYou is clean. It is easy to modify, scalable, and reusable.

However, there are still some problems, such as handling unexpected situations in reliability.

Case Study 2: ANU Fifty50

Description

ANU Fifty50 is a student-run online mentoring platform which aims to help mentees (junior students) to search, find, and pair with their mentors (senior students) automatically. The intent of this platform aims to replace the Wattle website that is being used for this functionality at present. Users can sign up anytime they prefer. After they are paired with their mentors, tutorial and coaching content will be released every two weeks.

Like FutureYou, ANU Fifty50 is also based on Django framework with the Python programming language implementing the backend. For the user interface the developer team has used a website style, using Bootstrap, HTML, CSS and JavaScript for frontend. MySQL is used to manage the database.

For the implementing functions details [14]:

- Mentors can have multiple mentees.
- Admins can upload coaching content such as Facebook and Twitter and edit the digest page with summary.
- Admins can post news and events through Blog in Django.
- A setting page is created for users to update their personal information at any time.
- Feedback pages are provided to users to send their feedback to Admins.
- For the pairing algorithm, shown in Figure 8:
    1. Mentee request a “find mentor” function
    2. System check the availability of mentors in database
    3. Available mentors are collected to a list.
    4. Find appropriate mentors based on the requirements which are added by mentees
    5. Mentors, mentees and Admins are notified.
    6. If both mentors and mentees accept, Admins will pair them. Otherwise, new mentors will be provided to mentees.

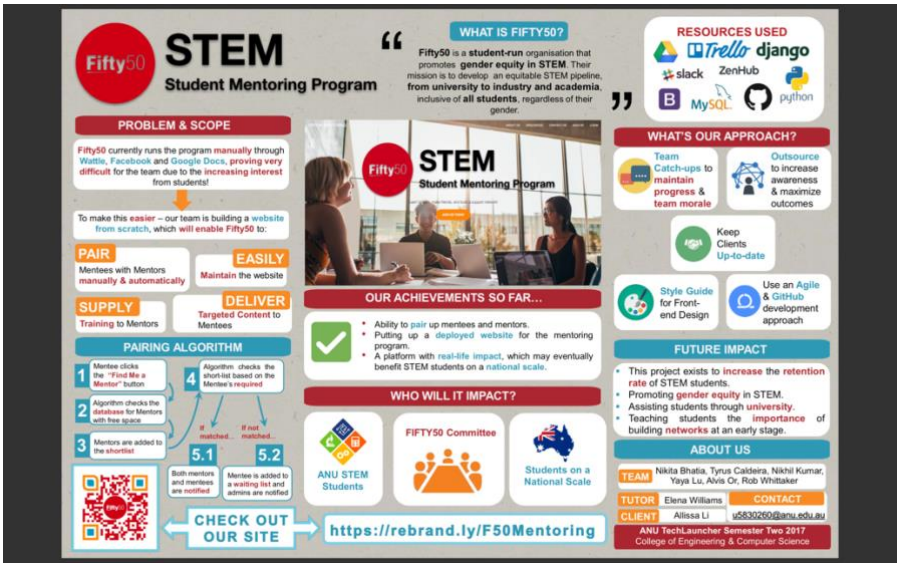

Figure 8. ANU Fifty50 post [23]

Analysis

For this discussion, I use the same method which I used on analysis FutureYou. I refer the analysis method which was provides in "Architectural Styles and the Design of Network-based Software Architectures" [7]. For the analysis of attributes, I imagine hypothetical scenarios, assuming in some specific conditions, how well the software handles the problem to analyse whether its architecture is clean and strong.

Performance

Performance is the standard for evaluating the quality of a software. Generally, the performance of a software is comprehensively evaluated through several aspects, such as running time, efficiency, memory usage and so on. For the ANU Fifty50, I still use the same three properties as what I do in discussion of FutureYou to describe its performance which are functionality, database and allocation of domain function [7].

- Functionality

Functionality usually refers to what a software does when it is running, such as the implementation of functions in an application. Similar to FutureYou, ANU Fifty50 is also a Web-application.

First, we do tests on ANU Fifty50 for individual functions. In ANU Fifty50, three users are involved in which are mentee, and mentor and staff as the software administrator. A common function they all have is login. ANU Fifty50 developers designed three functions which are registration, login and reset passwords in the project. However, these three functions are only applicable to mentor and mentee. Mentor and mentee can log into the home page to view the latest posts. Moreover, they can also query their personal information and have a view of their mentors and mentees. If they forget the password, they can reset the password by receiving verified email in their authored email box.

For staff, according to the design description, they should log in on another login web page, and then jump to a management and publishing interface which is more functional after logging in. However, the login function of the staff is not specifically designed, instead, developers directly provide an authorized web page for management and operation. There is no doubt that this is a lack of functionality. If non-administrators get this authorized web page to manipulate and steal users' information in the entire application, the consequences will be unimaginable.

- Database

In the design of ANU Fifty50, it applies the division of classes, developers put different classes into different folders, and make separate calls of function. For example, during the login process, if the login is mentee, the application will access the folder named "mentee" and call the function of the relative Python file and the data of the database to implement the login function. Again, this happens in "mentor" when the mentor logs in the website.

In this application, the administrator can directly manipulate the database to publish the latest post using a function called "publish ()". They can also delete it on website with a function called "auto_delete ()". In this project, developers use the Django framework and Python to modify the information in the database. We can find out through the following class diagram that most of the operations on the database come from web applications, not in the Django framework and Python.

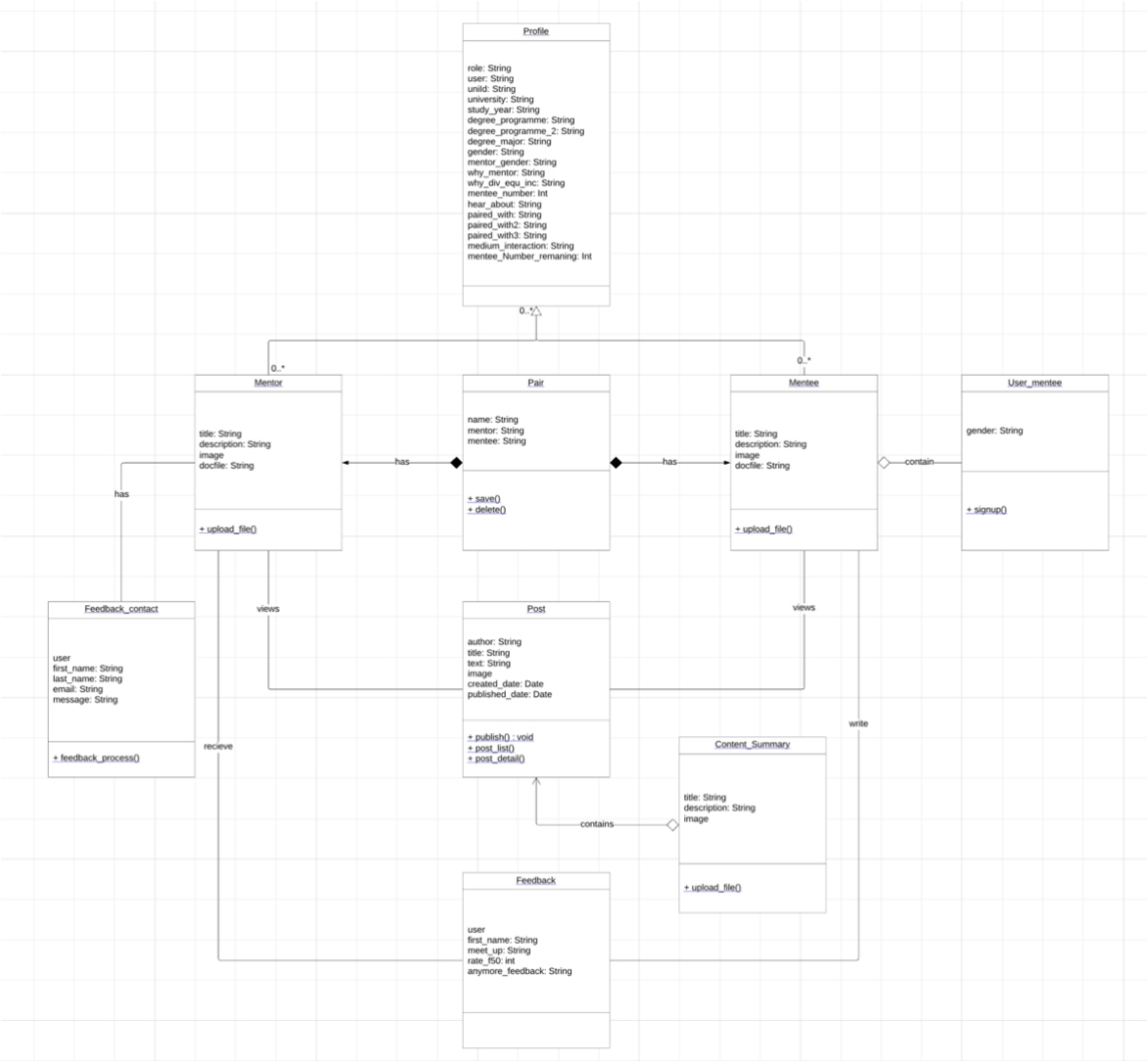

Figure 9. Class diagram for ANU Fifty50

- Allocation of domain function

  Now, I will extract a series of coherent operations from the application for analysis. For example, process of a mentee looks for and matches with a mentor. First of all, for mentee, its main function is to query the mentor and send a request to his favourite mentor to complete the pairing, and then having tutorial. The "search ()" function is involved in this process. When the application uses "check ()" automatically and the results are displayed on website, mentee selects the mentor that he likes and has available positions, and then uses "send_request ()". Meanwhile, the mentor receives the notification, he could view all the mentees who sent him requests, and he would choose whether to accept or reject the request based on the mentees' information. If the mentor accepts the request of a mentee, the system will

send the pair information to the administrator for review. Once the administrator's audit has passed, the id of the mentor and mentee will be recorded to a table called "pair" in the database. Also, a function that automatically modifies the attribute of "available_positions" is called. In the "mentor" table, the attribute of the available positions is decremented by one, and the "paired_with" attribute of the mentee is changed to the id of his mentor.

This matching process involves nine functions. It seems that this application is superior and rigorous in its implementation. Also, it follows the single responsibility principle (SRP): "a module should be responsible to one, and only one, actor" which is mentioned in book "clean architecture" [13].

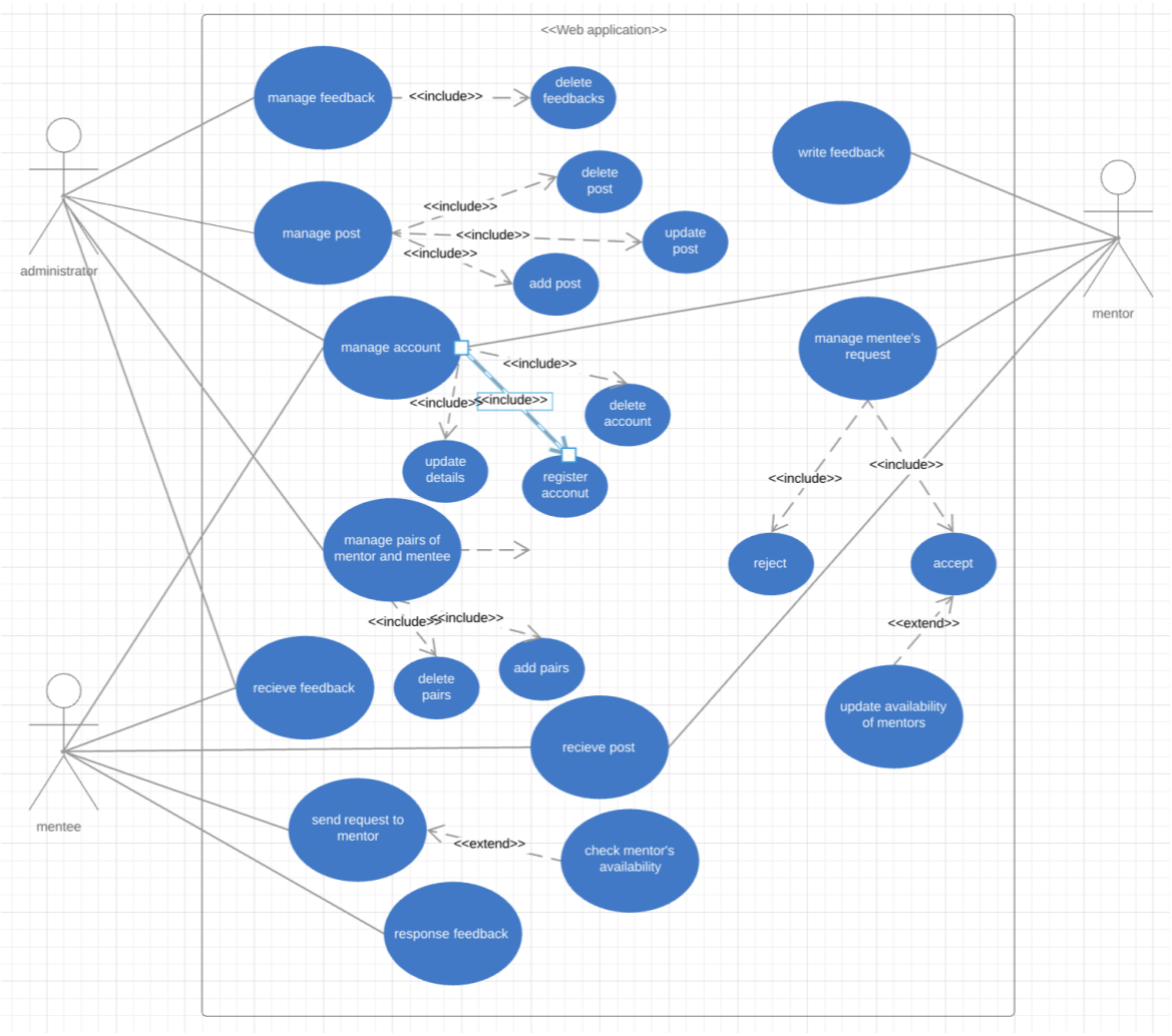

Figure 10. Use case diagram for ANU Fifty50

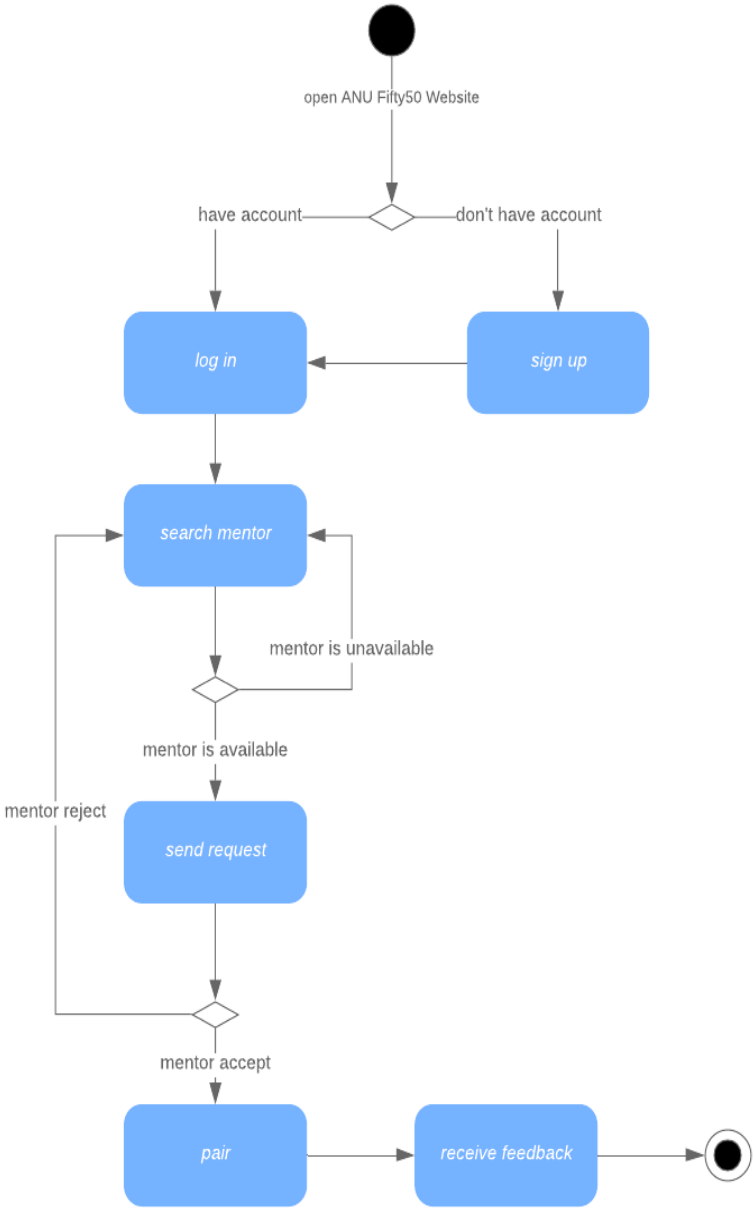

Figure 11. Activity diagram for ANU Fifty50

Scalability

Scalability means that when the number of users increases, an application's architecture can be scaled up to satisfy the requirements. There are two types of approaches to scale up which are in hardware and software.

On the hardware side, in order to improve processing efficiency, developers can use more powerful servers, or use multiple servers to solve it when the workload is overloaded. However, the cost of these developments is relatively high, for the ANU Fifty50, this application is relatively simple and does not require a very powerful machine to support it. Moreover, this leads to an increase in cost.

On the software side, the efficiency of dealing with problems can be improved by using multitasking. For example, assume that a machine can process up to 100 pair information in sequence. Then, we can process every four matching information concurrently, such that the processing speed is four times faster than before. There is an attribute called "available_position" to control the problem of over-matching which guarantees the mutual exclusion among each pair, using the idea of "first in first out". There are methods to implement this: separate computers or CPUs on the same computer or separate threads on the same CPU [9]. This can improve efficiency by processing data at the same time. Such an operation not only saves costs but also increases the efficiency of the machine.

In this regard, the developers of ANU Fifty50 have a basic design that will scale well for the future

Modifiability

For each TechLauncher project, stakeholders prefer the project to have a long lifetime. Therefore, in order to achieve software advancement and avoid being obsolete, modifiability is indispensable. In this section, we will imagine two hypothetical scenarios to explore whether ANU Fifty50 is a modifiable TechLauncher project.

In the first scenario, we make a hypothesis that the software requires a functional upgrade. The current version only supports a mentee's search for available mentors and does not provide information about the home college of a mentor when a mentee is performing a search. For example, a mentee from College of Business and Economics can search the result which contains a mentor from College of Engineering and Computer Science. Therefore, this function needs to be upgraded. For the first step, when mentee enters the mentor search page, we can create several choice boxes and name them as CS, Business, Law, and so on. The functions

corresponding to these boxes will be implemented in Python. For example, CS needs to add a function to query mentors who teach computer science in the database. In the process of adding this function, we only need to add a query function to the autoForm.py document in the “pair” document and use the "description" attribute in "mentor" table to implement such a classification.

In the second hypothetical scenario, we make a hypothesis of adding new features. Suppose that, when mentee gets a feedback from mentor, and mentee wants a face to face meeting. Then, he needs to book a study room whether it is in the Chiefly library or the Hancock library. All we have to do is show mentee the available rooms after he chooses the time period and date. In the process of adding this function, the first step is to create a temporary table in the database called "study_room". In the second step, we need to create a search algorithm to retrieve all the buildings in the ANU which have study rooms and query the room vacancy and store it in the temporary "study_room" table. In the process of adding this function, all parts are relatively independent and have no side-effect on other components.

In summary, by making assumptions about the addition and modification of ANU Fifty50, we can see that ANU Fifty50 has relatively strong modifiability.

Reusability

Reusable software case is everywhere in our daily life. Developers would like to have a basic developed template for reuse to get more profits. If this works well, a developer changes the template to meet users’ requirements to form a new application that is designed to a particular user. For example, consider the most common sales system. In theory, we can use a supermarket cash register system for another supermarket. In the process of reusing this software, all we need to do is modify the database, such as the product name, price barcode and other attributes.

For ANU Fifty50, developers divide functions and data structures by class rather than in the same large directory, which is both powerful and detrimental to project reusability.

The benefit is that the ANU Fifty50 can be used in similar TechLauncher projects such as software that needs a “pair” function. Also, it can also be used to train interns

in some companies, when an intern enters a company, as a mentee, they also need a mentor. So, we don't need to make many changes on the functionality. We just need to make some adjustments to the properties of some tables in the database. There are only minor changes on functionality in the ANU Fifty50 complex code base.

However, for some other applications, ANU Fifty50 is not very reusable. For instance, the student selective course system also involves three users: administrators, students, and teachers. It also needs “pair” function. However, ANU Fifty50's files are too fragmented and complex to reuse. When we are doing code reuse, we will find that there are many small Python files in each folder, and each file contains some functions. If you want to reuse ANU Fifty50, we need to add, modify and delete functions in each file. This undoubtedly increased the workload, because the code logic structure of ANU Fifty50 is confusing.

In general, the reusability of the ANU Fifty50 project has limitations and can only be reused for projects with similar requirements. When there are some changes in the project requirements, ANU Fifty50 is difficult to reuse because its logic of code is confusing, as well as the software architecture is really complicated.

Extensibility

Extensibility looks similar to scalability, but it has different definitions. Scalability is aimed at optimizing processes when the workload increases. The difference is that extensibility refers to the complexity of maintenance such as functional upgrade. Maintenance includes the addition and modification of functions in application. Therefore, extensibility and the strength of the structure are closely related.

For the ANU Fifty50 project, it is an application that is relatively difficult to extend.

First of all, in terms of adding functionality, because its developers divide different classes into different folders, maintenance staff can find corresponding Python or HTML files to add functions through these folders.

However, there are some complications with regard to the modification and deletion of functions. Because in ANU Fifty50, the function is implemented in both Python and HTML. When a function is modified and deleted, the maintenance staff needs to

find the module corresponding to the function and modify/delete it. For example, in the “pair” function between mentor and mentee, if we want to remove the administrator review step. First, we need to remove the admin review function and make related changes to the web page. At the same time, we also need to modify the function of mentor to accept the request. Because we want to make the "accept" function can directly modify in the "pair" table in the database. From this, we can see that, in the function modification and deletion, it is difficult to directly find the content that needs to be modified and deleted because of the complexity of the structure.

Therefore, ANU Fifty50 is not an extensible project because of its complex architecture and confusion design. Moreover, architecture cannot be easily understood by maintenance staff.

Portability

In this era of technological advancement, the tools people use to interact with computing systems are not only computers, but also a series of electronic products such as mobile phones. Different electronic products have different operating systems, such as Android, Windows, and so on. In order to implement the application on different operation systems, the concept of portability has appeared in the design concept.

ANU Fifty50 is a Web-application based on the Django framework. Its user interface is presented in the form of websites. It is not difficult to find that an electronic product, which can be browsed online, has an application called browser. The browser solves the portability problem of Web-applications such as ANU Fifty50. Users can not only use ANU Fifty50 websites on their computers for viewing and matching, but also can do this on mobile devices such as phones and PADs. However, there is no guarantee that the web pages will load well and be usable on mobile devices.

The design concept of the ANU Fifty50, which is Web-application, makes this TechLauncher project portable but not necessarily usable.

Reliability

Briefly, in a computer network, reliability is presented in the form of notification, for example, a notification is sent to user to check whether or not the information was successfully received [15]. Reliability also refers to the extent to which a project is affected in the event of a failure in the project. The lower the impact is, the higher the reliability of the project is. In terms of errors or faults, it can be divided into hardware errors and software errors, errors can also be divided into expected errors and unexpected errors.

In the analysis of the ANU Fifty50 project, we can make assumptions based on hypothetical error scenarios (as mentioned above in our treatment of the FutureYou project. In the first scenario, assuming that the hardware has failed. When a hardware failure occurs, it is not difficult to imagine, for example, the server has failed. In this term, we refer the server as a computer. When a computer breaks down, the simplest and straightforward solution is to change another computer to be a server. Therefore, for the small website application ANU Fifty50, hardware failure is not a difficult problem to solve. Moreover, when the server stops working, users who visit the website will also be notified by the browser, just like, the server cannot be found.

For a second hypothetical scenario, we assume that there is an error which occurs in the software. If this problem is expected, for example, the user has forgotten the login password. In this expected problem, the application will display a notification like user password not matched and ask the user whether they need to reset the password. The user can choose to reset the password through the verified mailbox. And there is another example. When mentee chooses mentor, we assume his favourite mentor has no extra vacancies. The system will notice that the mentor cannot be selected because he is full.

For a third scenario, we assume an unexpected error. For example, the workload suddenly increases, in other words, there are a large number of users who are using ANU Fifty50 at the same time and make changes to personal information and pairing. The processing scheme here is as we mentioned in the scalability property. The solution is that performing multi-process processing. Hardware can be solved by enhancing server performance or adding more servers.

In general, as a web application, ANU Fifty50 has strong reliability, and reliability can be enhanced with the enhancement of scalability.

Summary

The following table lists the positive and negative aspects of architecture of ANU Fifty50.

Table 2:

criterion	***Good***	***Bad***
Performance	Many streamlined and powerful functions	Some functions cannot be implemented Database and structure is too complicated
Scalability	Easy to scale up in both hardware and software	None
Modifiability	Can add new functions to the current project	Not easy to do modification s due to complicated structure and bad logic in some code design
Reusability	Can be reused for some similar projects	Functions are too scattered to reuse to other projects
Extensibility	None	Complex architecture and confusing design
Portability	As a web application, ANU Fifty50 is portable on most operation systems	Usability issues for mobile devices
Reliability	Can solve most expected and unexpected errors.	Not design for multitasking, but can be modify into multitasking

Conclusion

In a conclusion, ANU Fifty50 is a project, which is uncompleted, because some features which are defined in the web pages cannot be implemented on code level. In terms of attributes, we can find that the architecture of ANU Fifty50 is not quite clean. It is easy to modify, scale, and reuse as well as portable. However, there are still some problems in extensibility. And there some problems in its poor design logic and complex architecture which may cause issues in maintenance.

Comparison

A table summarized the result of comparisons will be given first. Then, it comes to the discussion in detail.

Table 3:

criterion	***FutureYou***	***ANU Fifty50***
Performance		√
Scalability	√	√
Modifiability	√	
Reusability	√	
Extensibility	√	√
Portability	√	√
Reliability		√

- **Performance**: Both of FutureYou and ANU Fifty50 are uncompleted project because there are some functions cannot implement in their applications. In terms of performance, I think ANU Fifty50 is much better which has powerful and various functions. Meanwhile, functions in FutureYou is quite simple, but the design ideas of FutureYou is pretty good.

- **Scalability**: In terms of this property, I think there is no difference between these two projects. Both of them have some preparations for scaling up through software, moreover, they are also able to be scaled up through the hardware.

- **Modifiability**: FutureYou is better in modifications. First, it is a smaller project than the ANU Fifty50, in other words, it is not as complicated as ANU Fifty50. Second, its development team design it in a clean logic and architecture which make it easy to modify.

- **Reusability**: As I mentioned in modifiability part, FutureYou is a smaller project than the ANU Fifty50. Also, because of its clean logic and architecture, it saves developers' time and energy to reuse it for other projects.

- **Extensibility**: In terms of extensibility, I cannot say which one is better. FutureYou is a simple application which has more space for expansions. Meanwhile, ANU Fifty50 has various features, and it can also be upgraded and modified for the future requirements.

- **Portability**: Both of FutureYou and ANU Fifty50 use the Django framework and same design pattern. As a Web-application, they can be implemented on most operation systems, however, there are also usability issues on some mobile devices.

- **Reliability**: Compared with FutureYou, it seems that ANU Fifty50 make more preparations and solutions for the expected and unexpected errors. Therefore, ANU Fifty50 is relatively reliable.

Conclusion

In this report, the main analytical ideas are from Roy Fielding's paper "Architectural Styles and the Design of Network-based Software Architectures." I used some of the architectural properties, in the Chapter 2 Section 3 of the paper, which is "Architectural Properties of Key Interest," to analyse the architecture of FutureYou and ANU Fifty50. Meanwhile, some of ideas are from articles on academic websites and academic reports. In this report, I combine these ideas together for analysis.

In terms of analysis, most of the time, I use hypothetical scenarios to analyse and judge the attributes of architecture. It must be said that this method of analysis has side-effect, because there might exist other situations out of my hypothetical scenarios. In other words, there are some situations that I did not consider. For example, in some cases, the result of the better one of these two project might be different.

The results of the analysis show that ANU Fifty50 is a larger and more ambitious project than FutureYou. In fact, due to the complex structure and architecture of the ANU Fifty50, it is not better than FutureYou. The logic of FutureYou is clear and its architecture is clean, which saves time and cost for maintenance and development.

However, there is no doubt that ANU Fifty50 has diverse and powerful features. Meanwhile, FutureYou's features are not as powerful and diverse as ANU Fifty50, that is why FutureYou needs to be improved.

For future work, I will make more hypothetical scenarios to facilitate a comprehensive analysis on each attribute. Moreover, I will also do more researches so that the two projects' architecture can be analysed more comprehensively.

Reference

1. Aladdin, M. (2018). *Software Architecture - The Difference Between Architecture and Design*. [online] codeburst. Available at: https://codeburst.io/software-architecture-the-difference-between-architecture-and-design-7936abdd5830 [Accessed 31 May 2019].
2. Bräutigam, R. (2018). *Is "Clean Architecture" by Bob Martin a rule of thumb for all architectures or is it just one of the options?*. [online] Software Engineering Stack Exchange. Available at: https://softwareengineering.stackexchange.com/questions/371966/is-clean-architecture-by-bob-martin-a-rule-of-thumb-for-all-architectures-or-i [Accessed 31 May 2019].
3. Chung, T. (2017). *How to Organize CLEAN ARCHITECTURE to Modular Patterns in 10 Minutes*. [online] Hacker Noon. Available at: https://hackernoon.com/applying-clean-architecture-on-web-application-with-modular-pattern-7b11f1b89011 [Accessed 31 May 2019].
4. Deutsch, D. (2018). *A quick introduction to clean architecture*. [online] freeCodeCamp.org News. Available at: https://medium.freecodecamp.org/a-quick-introduction-to-clean-architecture-990c014448d2 [Accessed 31 May 2019].
5. MDN Web Docs. (2019). *Django introduction*. [online] Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction [Accessed 31 May 2019].
6. Eeles, P. (2006). *What is a software architecture?* [online] Ibm.com. Available at: https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html [Accessed 31 May 2019].
7. Fielding, R. (2000). *Architectural styles and the design of network-based software architectures*. Irvine: University of California, Irvine ©2000, p.162.
8. Hansen, S. (2017). *Advantages and Disadvantages of Django*. [online] Hacker Noon. Available at: https://hackernoon.com/advantages-and-disadvantages-of-django-499b1e20a2c5 [Accessed 31 May 2019].

9. Jenkov, J. (2014). [online] Tutorials.jenkov.com. Available at: http://tutorials.jenkov.com/software-architecture/scalable-architectures.html [Accessed 31 May 2019].
10. Karam, L. and Mohsin, A. (2016). *Why is a good software architecture so important? | Apiumhub*. [online] Apiumhub. Available at: https://apiumhub.com/tech-blog-barcelona/importance-good-software-architecture/ [Accessed 31 May 2019].
11. Kazman, R., Bass, L., Abowd, G. and Webb, M. (n.d.). SAAM: a method for analyzing the properties of software architectures. *Proceedings of 16th International Conference on Software Engineering*.
12. Macneil, M. (2017). *Better Software Design with Clean Architecture*. [online] Fullstackmark.com. Available at: https://fullstackmark.com/post/11/better-software-design-with-clean-architecture [Accessed 31 May 2019].
13. Martin, R. (2018). *Clean Architecture*. Boston Columbus Indianapolis New York San Francisco Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo: Prentice Hall.
14. Read_me ANU Fifty50. (2018). 1st ed. Australia: ANU Fifty50 development team, p.2.
15. En.wikipedia.org. (2019). *Reliability (computer networking)*. [online] Available at: https://en.wikipedia.org/wiki/Reliability_(computer_networking) [Accessed 31 May 2019].
16. En.wikipedia.org. (2019). *Reusability*. [online] Available at: https://en.wikipedia.org/wiki/Reusability [Accessed 31 May 2019].
17. Schults, C. (2017). *A Detailed Introduction to Clean Architecture - NDepend*. [online] NDepend. Available at: https://blog.ndepend.com/introduction-clean-architecture/ [Accessed 31 May 2019].
18. Schults, C. (2019). *A Detailed Introduction to Clean Architecture - NDepend*. [online] NDepend. Available at: https://blog.ndepend.com/introduction-clean-architecture/ [Accessed 31 May 2019].
19. Sei.cmu.edu. (2017). *Software Architecture*. [online] Available at: https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328 [Accessed 31 May 2019].

20. En.wikipedia.org. (2019). *Software architecture*. [online] Available at: https://en.wikipedia.org/wiki/Software_architecture [Accessed 31 May 2019].
21. Codementor.io. (2016). *Software Architecture 101: What Makes it Good? | Codementor*. [online] Available at: https://www.codementor.io/learn-development/what-makes-good-software-architecture-101 [Accessed 31 May 2019].
22. En.wikipedia.org. (2019). *Software portability*. [online] Available at: https://en.wikipedia.org/wiki/Software_portability [Accessed 31 May 2019].
23. Google Docs. (2018). *Techlauncher_Poster.pdf*. [online] Available at: https://drive.google.com/file/d/0Bw7MUyh12MoYVDNnWVN6cXgzTkk/view [Accessed 31 May 2019].

Appendix

Australian National University

INDEPENDENT STUDY CONTRACT PROJECTS

Note: Enrolment is subject to approval by the course convenor

SECTION A (Students and Supervisors)

UniID: ___u6022913 _____

SURNAME: ___Lin_____ FIRST NAMES: ___Zhanhui_____

PROJECT SUPERVISOR (*may be external*): _____Henry Gardner_____

FORMAL SUPERVISOR (*if different, must be an RSSCS academic*): _____

COURSE CODE, TITLE AND UNITS: _____COMP 4560___12 unit_____

COMMENCING SEMESTER ☐ S1 ☐ S2 YEAR: _____ _ Two-semester project (12u courses only): X

PROJECT TITLE:

Software Architectures for Techlauncher

LEARNING OBJECTIVES:

Students will learn about software architectures, frameworks and tools used in group projects. Students will learn how to critically evaluate architectures and to plan for change. Students will learn how to write a formal report.

PROJECT DESCRIPTION:

This project will examine a collection of reports and software repositories from Techlauncher group projects that have already been completed. With reference to modern ideas of good software architecture, students will ask questions such as: "What common features do these code-bases have?", "How do they plan for future change?", "How could they be improved"? Students will draw up architecture diagrams and will suggest ways in which these architectures might be refactored and improved. If the time permits, students will build information modules to help future Techlauncher teams working on similar projects to get quickly up to speed.

Research School of Computer Science *Form updated Jun 2018*

Australian National University

ASSESSMENT (as per the project course's rules web page, with any differences noted below).

Assessed project components:	% of mark	Due date	Evaluated by:
Report: style: _____90_____ (e.g. research report, software description...,)	(min 45, def 60)		
Artefact: kind: _____0_____ (e.g. software, user interface, robot...,)	(max 45, def 30)		(supervisor)
Presentation :	(10)		(course convenor)

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfil the above defined contract:

…….. ……30/07/2018……
Signature Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following examiner, and have obtained their consent to review the report (via signature below or attached email)

……..…… ……..…….. ……27/07/2018……
Signature Date

Examiner:
Name: …… Signature ……
(Nominated examiners may be subject to change on request by the supervisor or course convenor)

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course convenor approval)

…….. ……..
Signature Date

Research School of Computer Science *Form updated Jun 2018*