Comparison and Analysis of the Four Independent Software Engineering Projects

Master of Science in Computer Science

City University of Seattle

CS 504 - Software Engineering Summer 2021

Group Members:

Ip Chan [chanip@cityuniversity.edu](mailto:chanip@cityuniversity.edu)

Hong Van [hongvan@cityuniversity.edu](mailto:hongvan@cityuniversity.edu)

Anil Ertuk [erturkanil@cityuniversity.edu](mailto:erturkanil@cityuniversity.edu)

Shubhashree Bhattacharya [bhattacharyashubhashree@cityuniversity.edu](mailto:bhattacharyashubhashree@cityuniversity.edu)

Instructor: Shane Frigon, School of Technology and Computing

Affiliation (CS504\_02\_IN: Software Engineering- SEATTLE - Summer 2021-2022, Degree Program, School, and University)

**Abstract**

Software engineering concentrates on discovering and designing a practical solution to a problem with a system. It involves more of the design elements, implementation, testing, and maintenance of the software. When building software, the improvement and change process is a necessary step to keep on upgrading the system and software configuration management, which is a system engineering process that tracks and monitors changes to a software systems configuration metadata, will support engineering teams to build fast and stable systems. In this paper, the authors will present their four different independent projects (Configuration Management System (CMS), Requirements and Architecture, Test Plan and Cases, and Software Metrics) regarding the design of a system and components based on the given subject, implementation of the designed system, and testing of the system. The authors will then provide a detailed report showing a comparison of the four total independent projects by analyzing each project’s requirements, architecture, test plan, and test cases, as well as software metrics and documentation.

**Keywords:** Software Engineering, Software Configuration Management, Test Plan, Test Cases, Software Architecture, and Software Metrics

**1.INTRODUCTION**

Software engineering is about the development and maintenance of all the software that people use every day, from productivity tools to any web browser. It is also about the application of engineering concepts to software development with the main goals of creation, improvement, and maintenance of software. There are many types of software that a software engineer can develop, such as operating systems, computer games, middleware, business applications, and network control systems.

When designing any software product, it is very important to analyze and understand clearly the needs of the customer, so any software engineer can design, test, and develop their software products to meet those needs. Moreover, during the software development process, project planning, system designing, implementation, testing of software, and maintenance are important steps that need to pay much attention to.

The purpose of this paper is to compare the differences and similarities of the four individual projects with the following topics: collaborative online judge system, IoT-based weather application, a web service to manage user’s password, and a vaccine scheduling system. Individual project one was about the introduction to the configuration management system (CMS). Individual project two focused on the project requirements engineering process and map the resulting requirements into a systems architecture or high-level design. Individual project three created a test plan including test cases for the system they designed for all four projects. Finally, individual four was about software metrics.

**2. IP01: Configuration Management System**

The first individual project was about the introduction to the configuration management system (CMS) and demonstrated how the related processes are handled in the CMS environment. CMS is a system engineering process that tracks and monitors changes to a software system configuration metadata. It uses version control software, such as Git, SVN, Hg (Mercurial), and ClearCase, to manage the state of software code as well as to track changes to the software over time. As the most popular version control system or online service, Git and GitHub were used as a CMS environment to perform this project. In this project, the authors were required to create a test case and perform the following situations in the GitHub environment including code check out, code check in, code snapshot, code development, and code changes. Below are the screenshots of each author’s GitHub repository in order to complete this first project.

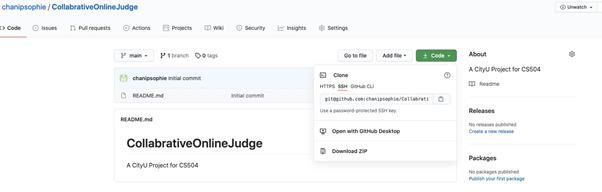


Figure 1.1: GitHub Repository - Author# 1

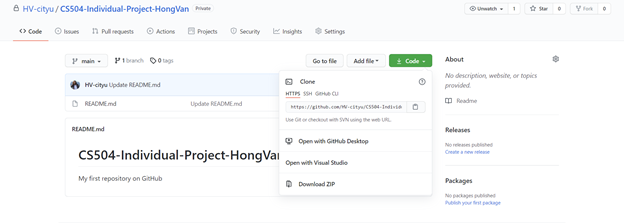


Figure 1.2: GitHub Repository - Author# 2

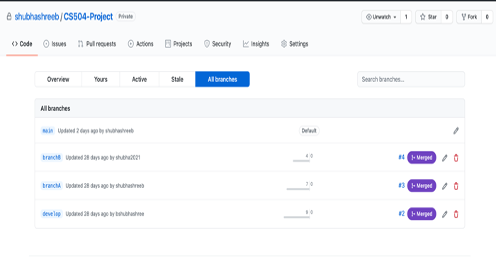


Figure 1.3: GitHub Repository - Author# 3

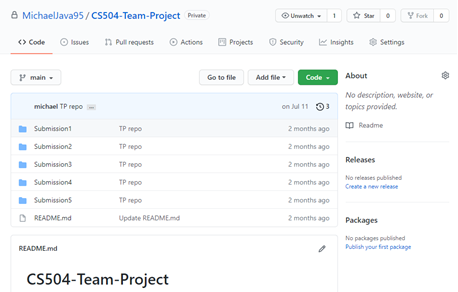
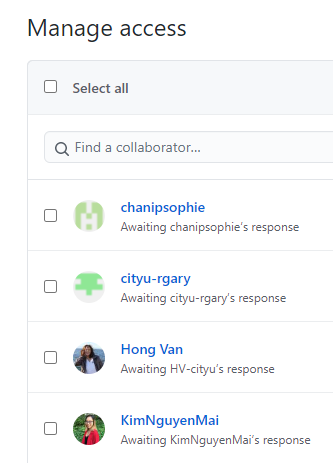


Figure 1.4: GitHub Repository - Author# 4

**Comparison**

Since the authors were allowed to create different test cases to demonstrate each situation, each individual project had different code design and the output of each situation was different from each other as well. For the check out of the source code, all of the authors were able to clone the repo to their local machines successfully by running the “git clone” command. To do this, authors #2, #3, and #4 did git clone using HTTPS URL while author #1 was using SSH URL - see figure 1.1, 1.2, 1.3, and 1.4. Moreover, author #4 also created a branch name and did a git clone using the “git clone –b branchName” command. For the code changes, each author made some changes to the code locally and saved it in the file that they just cloned into their local machine. For source code check in, all the authors were also successfully pushing the new code changes to the CMS environment by using the following commands git add . , git commit -m “message”, and git push origin master. Commit means saving the state of the current codebase, and push is to send the changes committed to the GitHub environment so that other developers can see the changes. For code snapshot, only three authors have done this part, one created a scenario to verify a user be able to create a release from a given branch as well as to be able to update and edit release while another one included a commit message in the code and called it code snapshots. In the last part, code development by 2 users, one is working on a new version, and one is fixing bugs in a previous release, author #1 has demonstrated the situation by creating a new branch, called “bug\_fixing”. The purpose is to demonstrate that multiple developers can work together on different tasks without conflicting with the current release branch version. Author #4, has shown the multiple users - see the screenshot below - to prove that multiple users can work on a single project and this is generally done through branches. All the authors had proved the process successfully.



*Figure 1.5: Multiple users access*

Overall, there were some variations in the design and structure between each of the author’s projects. It includes the differences in code changes and code development situations that are handled in the CMS environment since each author created different cases to handle on the GitHub environment. However, all three projects, when testing in CMS, print the output as the authors expected without any error.

**3. IP02: Requirement and Architecture**

In individual project two, the authors were focusing on the project requirements engineering process and mapping the resulting requirements into a systems architecture or high-level design. The requirements included functional requirements and non-functional requirements. Therefore, in this section, the authors will compare and analyze the requirements and high-level designs of these four projects in detail.

The collaborative online judge is a full-stack web app supporting collaborative online code editing, compiling, execution, and result judgment. The functional requirements of collaborative online judge systems are detailed design and implementation plans for those core features. The first core feature for this website is CRUD the problems that allow an admin to create, read, update, and delete (CRUD) coding problems into the system database. Another core feature is Submit code, execute code, and show results. After users have finished coding, one of them can hit the submit button. For the non-functional requirements, there are three major non-functional requirements for most web-based applications: security, availability, and performance.

The IoT-based weather application has been designed to facilitate the reporting of weather parameters over the Internet to a centralized place preferably in the cloud. In this, the IoT system will be embedded with temperature, humidity, and rain sensors that can monitor the weather conditions and provide live reports of weather statistics. Information will be stored in a centralized database. It focuses on the requirement gathering, system requirement specification, design, and implementation of the app.

The main core feature of the password management web service is the password manager feature. This feature will allow a user to create/add, store, update, and delete their accounts and passwords into the system database. To do this, the users need to list all the names of the applications, usernames, and passwords to input this information into the system.

The main feature of the vaccine scheduler system is to allow people to reserve a time slot remotely, in order to decrease unvaccinated contact. This system doesn’t have to be just for the coronavirus vaccine. Other vaccines can also be scheduled via the vaccine scheduler. The functional requirements of the vaccine scheduler system are showing the closest vaccination spots, and the available date and time slots. The non-functional requirements of the vaccine scheduler system are to be secure and accessible via the internet. A web application is going to give the system multi-platform accessibility. The same web applications can be accessed from different operating systems and from different types of devices such as phones, tablets, laptops, and desktops. The system also needs to be robust against cyberattacks to protect user data.

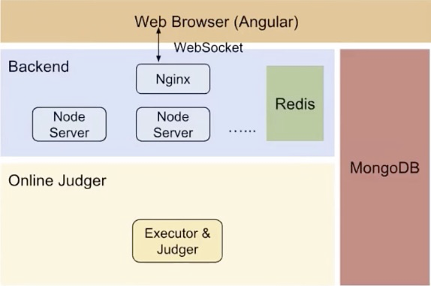
**Comparison**

All four projects are web-based applications. This means that they have sub-components that can talk to each other via network communications. Also, they allow clients to make HTTP(S) requests and provide corresponding responses. In terms of some non-functional requirements, once the application gets more and more popular, they should make sure that their application services have high security, high availability, and high performance. How to achieve these three requirements are design and implementation details, however, the principles are the same. However, there are some differences among these projects shown as below:

* **Businesses areas**

What is interesting is that these four projects focus on different areas of business. Collaborative Online Judge focuses on the ever-increasing need for coding practices before technical interviews for today’s software engineering candidates. On the other hand, IOT-Based Weather Application takes advantage of today’s rapid growth of the Internet of Things and Cyber-Physical Systems, which also requires big-data processing. In addition, Password Management Services puts attention to internet security.

Finally, the vaccine scheduler system helps people to reserve a time slot remotely, in order to decrease unvaccinated contact. This system is not only just for the coronavirus vaccine but also for other vaccines. Simply put, all four projects are reflecting today’s business trends, although they are dealing with completely different areas and pain points.



*Figure 2.1: High-level architecture design*

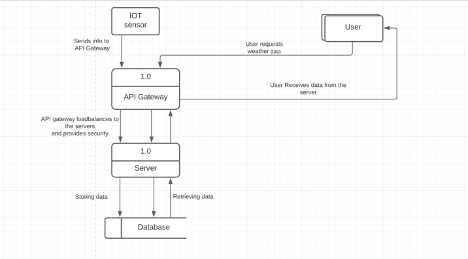
* **Software Architecture**

This one is a little interesting. Four projects take different approaches to design high-level architectures. Microservice-based approach. The Collaborative Online Judge uses a modern Service Oriented Architecture (SOA) where are multiple sub-services in this application, such as the Agular-based frontend, Node.js-based backend, WebSocket-based communication channel between frontend and backend, Caching layer, Background processes such as the judges, databases and so forth. All these sub-services will communicate in several different ways. WebSocket, as mentioned above, is one of the mechanisms. Others include RPC, HTTP, etc. This type of architecture has the advantage of being easy to scale out, high availability, and so forth. However, if the application is not a large-scale distributed service, this approach may seem a little overkill. The complexity of this approach is also higher compared to other approaches.

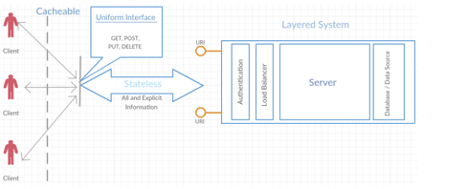
The IoT-based weather app takes another different architecture style. It uses a more traditional architecture, i.e., the Three-tier architecture. There is a presentation tier (the user interface), an application server tier (the API Gateway and the server), a data tier (the database). There is another tier called the IoT sensor, however, it is still the traditional three-tier architecture. This architecture is well-established and easy to implement. All of the business logic-related code can be written inside each tier. The way to scale is also well-known. The downside of this approach is that if the service is a large-scale distributed service, it is more difficult to scale compared to the microservice-based approach. Another downside is that the code becomes more coupled with each other's teams, as a result, it is easier to introduce bugs and errors, and the cost of communications and extensions is also higher.

Password Management is somewhat between the two styles. It, however, is more like a microservice-based architecture, since it has multiple components which will communicate with each other through well-defined protocols such as HTTP or RPC.

The vaccine scheduler system uses MVC architecture which is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. In this system, MVC allows the creation of a dynamic website where the user can see dynamically generated pages and can submit data. The model part will be necessary to have persistent data. Separating Model, View, and Controller makes the web apps more maintainable. Each of these components is built to handle specific development aspects of an application. This architecture also allows the system to perform CRUD (Create, Read, Update, and Delete) operations. The user needs to be able to see the current data of unreserved time slots.



*Figure 2.2: Data Flow Diagram*

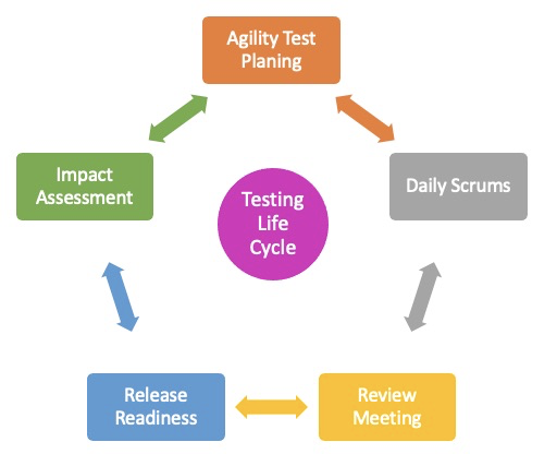
****

*Figure 2.3: REST architecture style constraints. Source: Yadav (2020).*

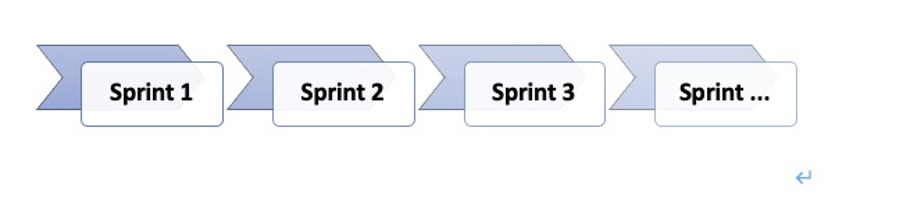
**4. IP03: Test Plan and Cases**

In this project, each author was expected to create a test plan including test cases for the system they designed for the previous project. They had to develop a test plan and the plan had to include an overview of the approach to testing including the project test philosophy and develop example test cases for all the required tests. The test cases had to be consistent with the test plan approach and requirements.

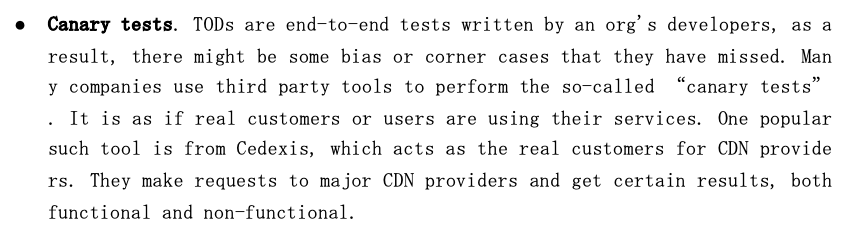
The author #1 developed a test strategy and a plan to test the collaborative-online-judge-system that she had designed earlier. The project methodology and test plan were comprehensive and very professionally written where the author described how unit testing would be done on the web browser, backend, and the online judger and how the units would be integrated and the methodology for the integration test, end-to-end, canary and regression test. the author #1’s implementation of the agile testing process is done as per industry standards and can be observed below.



*Figure 4.3 Agile testing life cycle*

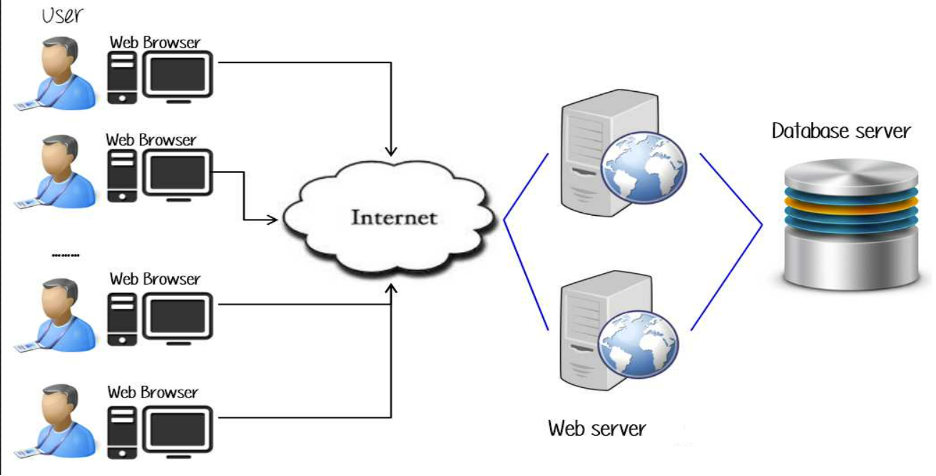
**

*Figure 4.4 Testing life cycle and Sprints*

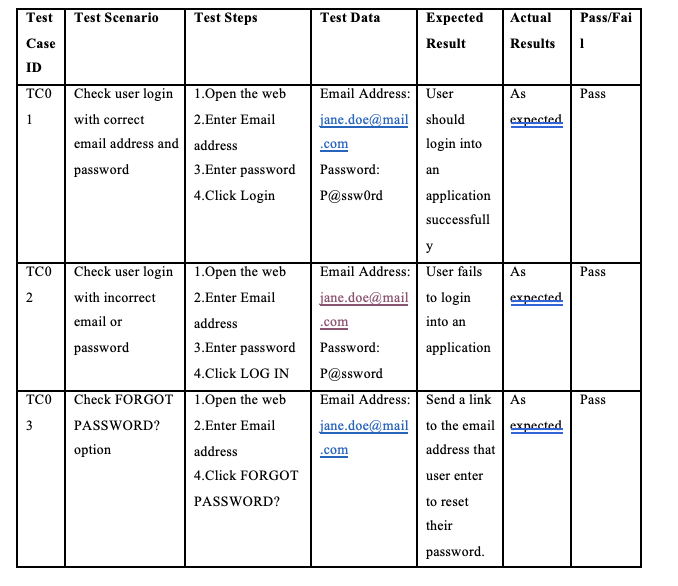
**

*Figure 4.5: Canary test*

In author #2’s project, the author at first defined the test scope, objectives, and entry-exit criteria. She created a test plan where she described the test methodology and test environment. Then, the author #2 designed the test cases for the system that was designed for the previous assignment, which was a web service to manage the user's password. The test cases were written to demonstrate the end-to-end tests and functional tests to test the user experience. The test case report was very well written in a tabular format as per the industry norm, with well explained test case scenarios, steps, expected, and actual results. Below is a good example of how to write good test cases.

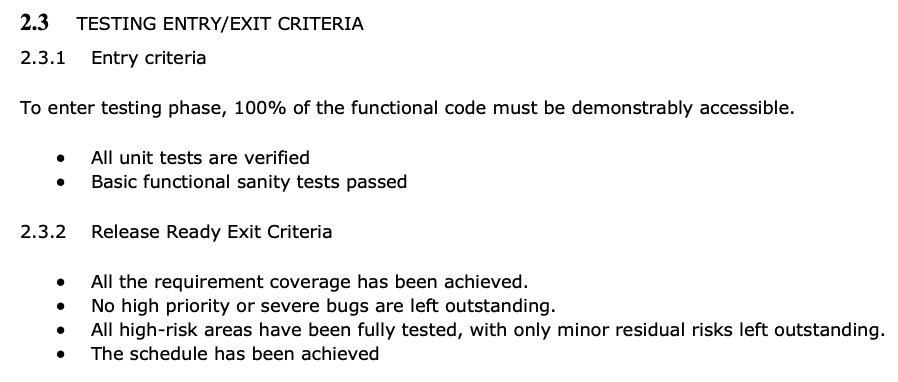


*Figure 4.1: Test Environment*



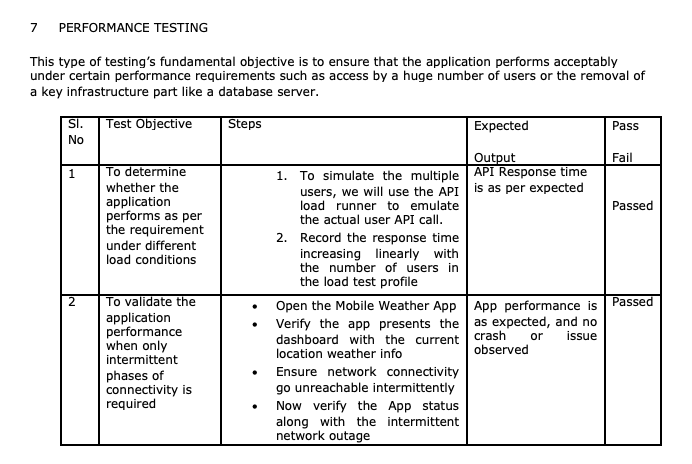
*Figure 4.2 Test case format*

The author #3 developed a test strategy, scope, objectives, and entry-exit criteria and a test plan to test the IoT based weather application which she had earlier designed.



*Figure 4.6: Test Entry/Exit Criteria*

Then, the author #3 developed the test cases based on the test plan for unit testing, integration, performance, functional (e2e), performance, security, and compatibility testing. Below is an example of the implementation of performance testing.



*Figure 4.7 How to implement Performance testing.*

In the author #4 project on Vaccine Scheduler, the author wanted to test how the software handled correct and incorrect inputs and that the components of the web application were able to withstand attacks such as SQL injection and cross-site scripting. Therefore, the author #3 created the test plan where he included the test objective, test methodology, and the test environment. After the test planning was completed, the author #3 developed test cases for unit testing, functional testing, security testing, and regression testing to ensure that his application worked as per expectations.

**Comparison**

Even though all the projects were different from each other, all the authors were able to successfully design a well-rounded test strategy and test scope. They all designed test plans and were able to derive test cases to test their product using their choice of the software model.

The author #1’s test strategy was comprehensive, the test plan was extensive and included all the details, but had the test cases been individually chalked out, it would have been easier to understand. The author #2’s test plan was concise, the test cases were very beautifully written and easy to understand and as per the industry norm but had a few more diverse types of tests been included, it would have been better. The author #3’s test plan was well written and included the test cases for all the various kinds of testing as required but having diagrams for some of the test scenarios would make it easier for others to understand better. While the author #4’s test plan approach and requirements were very well-defined, including more specific written test cases to include performance testing and integration testing would be more helpful to identify bugs.

Overall**,** from this project (IP03), all the authors understood how to design a test strategy, objectives and design test cases based on their test plan to test the system successfully. The authors concluded that a test plan serves as a roadmap to the testing process, that has all the necessary details related to the process. It is a systematic approach, so it provides better functional coverage. It would enable the team to accurately estimate the required effort and cost to execute the testing process. The students understood that test cases bring together the whole testing process. If the tests are designed to test the functionality, performance, load, integration, and others, and cases are designed to test all the scenarios, they are helpful to measure whether the expectations are fulfilled or not. When the test cases are well written and executed correctly, the chances of missing the bugs in the software reduces significantly.

**5. IP04: Software Metrics**

The fourth project was about metrics for the validation and acceptance testing of the software system designed in the previous projects. In order to understand why these metrics are necessary, one needs to understand the purpose of software testing.

The purpose of software testing is not to make the product bugless. The purpose of software testing is to ensure that the software product is above a predetermined quantifiable level of quality. This is because the software team agrees to spend a certain amount of resources on testing to ensure the number of tests, coverage, rate of bugs and failed tests. Agreeing on objective, quantifiable, and systematic metrics allows the development team to determine whether the software product is above the predetermined level of quality which makes it ready for deployment. This is why metrics are necessary for acceptance testing of a software product.

Software metrics are objective, quantitative assessments, and standard measurements of software attributes.

**Comparison**

The author#1 divided the metrics for functional and nonfunctional requirements. For example, Problem-list, problem details for functional requirements; Security, availability, and performance for non-functional requirements. There are two types of metrics in the functional requirements, Counter and Gauge. The counter is the total count of some value for a certain metric, whereas the Gauge is the arbitrary value for a metric. In addition to the functional metrics which are more generic and can be applied to different services. Non-functional metrics can be further divided into three broad categories: Security, Availability, and Performance; such metrics should be used to collect the Service uptime and downtime; and DNS lookup time, cache hit rate (e.g. Redis), and so forth.

The author #2 analyzed metrics for usability, security, and validation testing of the system she designed, which is a web service to manage passwords of users. She emphasized that what matters to the user is the user experience and functionalities.

The author #3’s project worried more about testing the code itself. Coming up with good tests that actually discover bugs was important enough to become a metric. Being efficient with test resources in the form of writing and reviewing many tests per unit time was also a metric. Coverage of code lines and analysis of the rate of bugs and failed tests were also metrics. She also came up with objective, quantitative goals for software. These goals are required for the deployment of the software. Some of these goals include more than 95% test coverage, more than 95% test pass rate, failed tests to be noncritical, achieving functional coverage, and the robustness of handling inputs.

The author #4’s metrics focused more on functional and nonfunctional requirements (security, safety, performance, and usability) and analysis of code lines such as coverage and bug rate. Security tests included external attacks such as SQL injection and cross-site scripting. Safety tests focused on test cases for internal exceptions such as NullPointerException. Performance tests analyzed run time of methods. Functional metrics focused on the completion of user stories.

One similarity is the need to have a metric for usability in order to keep a track of how usable the system is. Both the author #2 and the author #4 had a metric to gauge the usability of the product by the end-user. This metric is important to shorten the gap between the software team and the user. The intent of developers and the experience of users can be different. This is why there are good design patterns and standards that the developers should follow. The functionality should be clear and intuitive to the user.

Another similarity was the need to test the security of the application. For example, some of the projects included tests for security to ensure that the product is robust against external attacks. One out of four projects focused more on the safety of her product which is related to ensuring that the product is robust against internal errors. This can be caused by the types of applications. A web application is going to need to worry more about security than a GUI calculator application. Also, different fields are going to have their own set of problems they need to worry about.

Another similarity is that all projects have metrics that are used to track defects and structure the process of their resolution in order to reduce the occurrence of accidents and defects.

One difference was that the author #2 was the only one who had metrics for software maintainability. This metric relies more on an agile process to put the product in the hands of the user so the team can gather metric data such as the speed at which bugs are fixed.

Another difference between all the authors’ projects is that they consider different requirements. For example, the IoT-based Weather app concerns effectiveness, coverage, and defect distribution. For Web Service to Manage User’s Password system, its metrics should be based on usability, security, and maintainability.

**6. CONCLUSION**

After completing these 4 independent projects over the last 8 weeks, the authors understood that having a process and documentation is very important at every stage to ensure that they can monitor and guide the software project towards a successful end.

GitHub is a great platform for managing configuration data. Moving configuration data into a GitHub repository enables better version controlling and the repository to act as a source of truth. Version control also solves another configuration problem: unexpected breaking changes. Managing unexpected changes using code review and version control helps to minimize downtime.

A software requirements specification is the basis for the entire project. It lays the framework that every team involved in development or in our case, each student needed to follow. It should be written to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page. Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about our product’s lifecycle — for instance when to retire a feature. Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS.

Design documents guide us regarding the design pattern to flow and flowcharts such as DFD, UML, and ER diagrams help us understand the flow of data.

The test plan keeps track of possible tests that will be run on the system after coding. The most important test cases come from the requirements of the system. When the system is in the design stage, the initial tests can be refined a little. During the detailed design or coding phase, exact test cases start to materialize. After coding, the test points are all identified, and the entire test plan is exercised on the software.

**7. References**

*Ross, D. T., Goodenough, J. B., and Irvine, C. A. "Software Engineering: Process, Principles, and Goals," in Computer, vol. 8, no. 5, pp. 17-27, May 1975, doi: 10.1109/C-M.1975.218952.*

*Davis, A. M. "Fifteen principles of software engineering," in IEEE Software, vol. 11, no. 6, pp. 94-96, Nov. 1994, doi: 10.1109/52.329409.*

*Ebert, C., Murthy, B. K., and Jha, N. N. "Managing Risks in Global Software Engineering: Principles and Practices," 2008 IEEE International Conference on Global Software Engineering, 2008, pp. 131-140, doi: 10.1109/ICGSE.2008.12.*

*Mohod A.(2015). Why test cases are so important* [*https://blog.e-zest.com/why-test-cases-are-so-important/*](https://blog.e-zest.com/why-test-cases-are-so-important/)

*ReQtest. (2016). The Pros & Cons of Using a Test Plan.* [*https://reqtest.com/testing-blog/using-test-plan/*](https://reqtest.com/testing-blog/using-test-plan/)

Buchanan, I. (n.d). Configuration Management. [*https://www.atlassian.com/continuous-delivery/principles/configuration-management*](https://www.atlassian.com/continuous-delivery/principles/configuration-management)

*Kruger N (2018). How to write software requirement specification documentation.* [*https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document*](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document)

*Cigniti Technologies (n.d). Why do you need software testing metrics?* [*https://www.cigniti.com/blog/why-do-you-need-software-testing-metrics/*](https://www.cigniti.com/blog/why-do-you-need-software-testing-metrics/)

Notes:

The author# 1: Ip Chan