COMP 3613 Assignment 1

Half Moon Studios

November 2019

Group Members:

Selah Lewis 816012230

Dominic Smart 816007605

Shereece A. A. Victor 816001671

Jerrel Williams 816009134

Contents

1 Theory - Software Process **3**
- 1.1 Software Performance Metrics (SPM) . . . 3
- 1.2 The Goal-Question-Metric Paradigm . . . 3
- 1.3 The Five Stages in the Process change process . . . 5
- 1.4 Process Improvement Models (PIMs) . . . 6

2 Software Engineering Tools and Requirements **8**
- 2.1 Software Configuration Management (SCM) . . . 8
- 2.2 Assessing SaaS for Continuous Integration . . . 8
- 2.3 Requirements Tracing . . . 9

3 Theory - Software Design and Construction **10**
- 3.1 Defensive Coding Practices . . . 10
- 3.2 Secure Coding . . . 11
- 3.3 Robust/Fault Tolerant Programs . . . 11

4 References **13**

1 Theory - Software Process

1.1 Software Performance Metrics (SPM)

A software metric is a measure of software characteristics which are quantifiable or countable [6]. Software metrics are important for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses [6].

1. Apdex is a software metric measure which aims to quantify and measure the user satisfaction of a software application. The Apdex method converts various software metrics into a single number on a scale of 0 to 1, 0 being the lowest and 1, the highest. The formula can be seen below.

$$Apdex_t = \frac{SatisfiedCount + \frac{ToleratingCount}{2}}{TotalSamples}$$

Figure 1: Mathematical formula for Apdex

2. Response Time measures the server response of every single transaction or query. Response time starts when a user sends a request and ends at the time that the application states that the request has completed [13]. Average response time is the average of the response time measurements. The goal for software response time is 0.1 seconds with 1 second being the maximum upper limit, and most users will close the application if response time is greater than or equal to 6 seconds. Response time can be measured using applications such as JMeter and Load Runner.

3. The Error Rate is a mathematical calculation which displays the percentage of problem requests against all requests [13]. Most errors, in the case of a web application, will be identified through the HTTP status codes returned by the server. However, some errors are not captured through that method and these errors must be accounted for separately. Errors also tend to raise in frequency as server load increases.

1.2 The Goal-Question-Metric Paradigm

The Goal-Question-Metric Paradigm is an approach used to measure software development in a hierarchical top down way.
At the highest level, the conceptual level, goals are defined. These include the business processes (eg., designing and testing software), products(eg., deliverables and documents), and/or resources(eg.,hardware and software used to produce output) which will be measured to assess performance according to the

objectives of the organization.[3]
Goals are then used to form questions which make up the operational level of the hierarchy. These questions are used to determine how the goals can be achieved and which areas should be targeted.
At the lowest level there are metrics, which are chosen to provide answers to one or many the questions to ensure that the metrics are collected from all viewpoints necessary. Metrics form the quantitative level and can be either subjective (eg., level of user satisfaction) or objective(eg., number of products sold). [2]

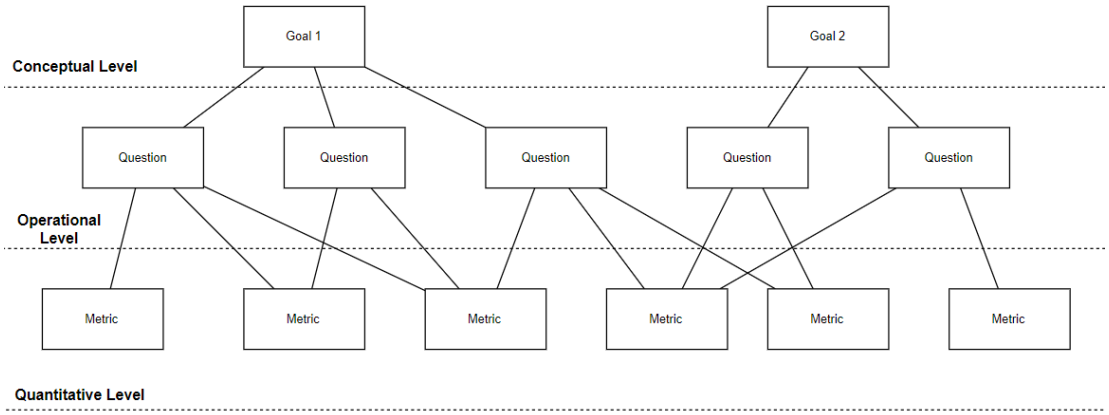

Figure 2: The hierarchical structure of the Goal Question Metrics Paradigm [1]

1.3 The Five Stages in the Process change process

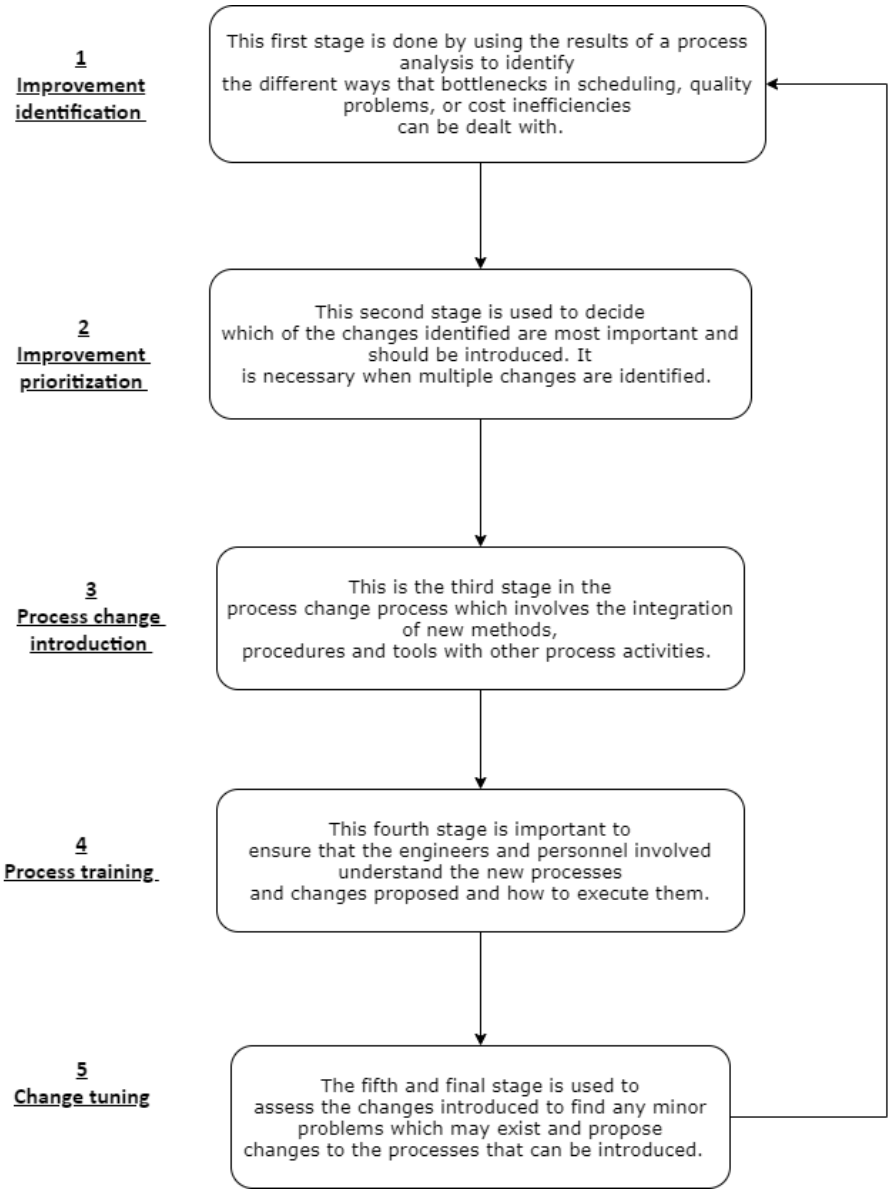

Figure 3: The Five Stages in the Process change process [10]

1.4 Process Improvement Models (PIMs)

Software Process Improvement Models aim to evaluate the software product, project, quality and drawbacks in order to streamline the software process.

1. The Capability Maturity Model involves a layered approach including 5 levels in order to refine the software development process. The first level is the initial level where processes are not sufficiently documented or replicable. Next is the repeatable level where organization has been developed and successes are now repeatable. After this is the defined level where a standard software processed has been developed. The managed level is next which is where the organization can now control and monitor its software processes. Finally is the optimizing level where software processes are constantly being improved through monitoring and feedback. An advantage of CMM is that higher CMM levels directly correlate to lower software defects. A disadvantage is that the level system of CMM is very rigid and may not tell a complete story.

2. Capability Maturity Model Integration is a process and behavioural model that helps optimize process improvement and decrease risks in development. CMMI also follows a level system which is the same as CMM but due to various improvements, it is more applicable in safety critical environments. An advantage of CMMI is that it fosters a mindset of quality maintenance within the programming team. A disadvantage is that while CMMI describes what process should be implemented by the organisation, it does not specify how. [14]

3. Continuous Quality Improvement refers to identifying issues and solving them in a continuous cycle. It is a theory-based management system which focuses on processes and outcomes. An advantage of CQI is its flexibility, allowing the organization to quickly meet changing requirements. A disadvantage is the model's difficulty of implementation as it requires an organization wide commitment to its practices.

4. The Plan-Do-Check-Act model is a four-stage approach to continuously improving processes. It involves testing possible solutions, assessing results and implementing the successful solutions. The Plan phase involves identifying and analysing the problem. Do refers to testing the possible solution. Check is where the result is studied for effectiveness. Finally, Act is where the solution is implemented if the Check phase is passed. The advantage of PDCA is that it instils a mindset of continuous improvement which can improve efficiency and productivity over time. However, the disadvantage is that the PDCA cycle can be much slower than a straightforward solution implementation.[15]

5. The ISO9000 model was defined as a set of standards for quality management and assurance to help document the quality systems necessary

for an efficient organization. It is based on 7 quality management principles which can be applied to promote organizational improvement. These are, customer focus, leadership, engagement of people, process approach, improvement, evidence-based decision making and relationship management. Some advantages of ISO9000 are increased marketability, reduced operational expenses and improved internal communication. The disadvantage is that attaining ISO9000 certification can be quite costly due to the documentation and overhead requirements.

2 Software Engineering Tools and Requirements

2.1 Software Configuration Management (SCM)

Software Configuration Management is a systems engineering process aimed at establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design, and operational information.[9] Some activities include:

- Promotion management (creating versions for developers)
- Release management (creating versions for clients)
- Branch management (managing concurrent developments)

However, these activities will employ the same software engineering tools and techniques used to create the product as illustrated below:

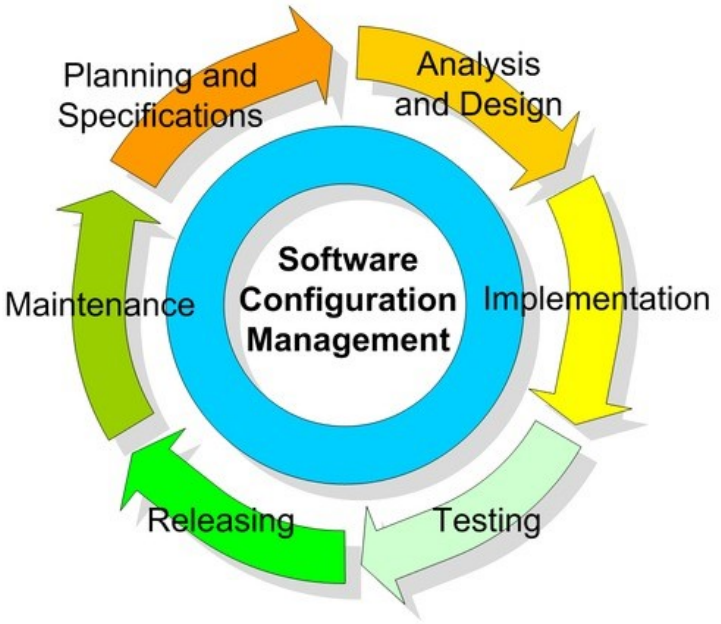

Figure 4: Showing the software configuration management cycle[8]

2.2 Assessing SaaS for Continuous Integration

Continuous integration refers to the automated building and testing of an application on every new commit. CircleCI is a Software as a Service model that delivers Continuous Integration to clients. Conversely, Jenkins is a self-contained

Java based program used to achieve CI. One advantage of using CircleCI is that they will oversee the setup, security and maintenance of continuous integration instances. Hence, software firms can reap the benefit of quicker releases of higher quality/stable products without it being another administrative concern. [5]

2.3 Requirements Tracing

Requirements Tracing refers to the ability to describe and follow the life of a requirement in both forward and backward directions. This entails tracing it from origins, through development, to deployment and through refinements. This is often achieved with a Requirements Traceability Matrix (RTM). It can be used to verify that all requirements have a corresponding design element, component, module or artefact. This is referred to as a forward trace. When the RTM is used to verify and document the source of the requirements, this is known as a backward trace. The backward trace is useful when customers have queries as to why certain features were included since it provides a comprehensive audit trail to make justified responses. Forward traceability is useful for checking the progress of the project by ensuring that each requirement is applied and tested thoroughly.[7]

3 Theory - Software Design and Construction

3.1 Defensive Coding Practices

Defensive Coding Practices:

Defensive coding is a form of defensive design intended to ensure the continuing function of a piece of software under unforeseen circumstances. These are most often used in environments where high security, availability, or safety are required.[4] Common defensive coding practices are [12] -

1. Validation
   The basic premise behind the concept of input validation is "Do not trust without verifying." Assume always you're going to receive something you don't expect. Two methods of validation are whitelisting and blacklisting.

2. Code Access Security
   Code Access Security prevents the code from untrustworthy sources comprising run-time permissions to function privileged operations. It also prevents the code from the trusted sources against intentionally or unintentionally comprising the security.

3. Container Versus Component
   Declarative security or container security offers a container managed strategy to ensure software flexibility, portability, and low deployment cost. Here, the security rules are configured outside the source code as a part of the deployment descriptor. The application container, or constraints, oversee applying dynamic security constraints to the code whereas programmatic or component security embeds the security rules in code or component itself. It involves a granular approach to implement security. When using this security, the programmatic enforcement is embedded in the application along with programmatic validation on policies to find, what a valid user is allowed to access.

4. Cryptographic Agility
   Cryptographic Agility is the capability of the code to switch to standard or recommended algorithms from insecure algorithms.

5. Attack Surface Evaluation and Reduction
   The attack surface is defined as the various points where a hacker can get into the software and can achieve data access. The attack surface evaluation is the process of understanding the areas of risks in the software, to make security specialists and developers aware about the regions of software that are vulnerable to attacks and determine methods to reduce this.

6. Memory Management
   Secure Memory Management can safeguard the revelation of data when it is processed. To implement proper memory management, it is important to follow some memory management concept.

3.2 Secure Coding

Secure Coding:

Secure coding is the practice of developing computer software in a way that guards against the accidental introduction of security vulnerabilities. Defects, bugs and logic flaws are consistently the primary cause of commonly exploited software vulnerabilities.[11]

1. Validate input. Validate input from all untrusted data sources.
2. Heed compiler warnings. Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
3. Architect and design for security policies. Create software architecture and design your software to implement and enforce security policies.
4. Keep it simple. Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation and use.
5. Default deny. Base access decisions on permission rather than exclusion.
6. Adhere to the principle of least privilege. Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task.
7. Sanitize data sent to other systems. Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components
8. Practice defense in depth. Manage risk with multiple defensive options, so that if one layer of defense turns out to be unreliable then another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and limit the consequences of a successful exploit..
9. Use effective quality assurance techniques. Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program.
10. Adopt a secure coding standard. Develop a secure coding standard for your target development language and platform.

3.3 Robust/Fault Tolerant Programs

Robust/Fault Tolerant Programs

Robustness describes an application's response to its input, while fault-tolerance describes an application's response to its environment.

An app is robust when it can work consistently with inconsistent data. For example: a maps application is robust when it can parse addresses in various formats with various misspellings and return a useful location. A music player is robust when it can continue decoding an MP3 after encountering a malformed frame. An app is fault-tolerant when it can work consistently in an inconsistent environment. A database application is fault-tolerant when it can access an alternate shard when the primary is unavailable. A web application is fault-tolerant when it can continue handling requests from cache even when an API host is unreachable.

4 References

[1] V. R. B.-G. Caldiera and H. D. Rombach, "Goal question metric paradigm", *Encyclopedia of software engineering*, vol. 1, pp. 528–532, 1994.

[2] P. Berander and P. Jönsson, "A goal question metric based approach for efficient measurement framework definition", in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ACM, 2006, pp. 316–325.

[3] F. Yahya, R. J. Walters, and G. B. Wills, "Using goal-question-metric (gqm) approach to assess security in cloud storage", in *International Workshop on Enterprise Security*, Springer, 2015, pp. 223–240.

[4] E. Adams. (Oct. 2017). Defensive coding best practices: A ceo's perspective - dzone security, [Online]. Available: `https://dzone.com/articles/defensive-coding-best-practices-a-ceos-perspective`.

[5] L. Lazinskiy. (May 2017). What is continuous integration?, CircleCI, [Online]. Available: `https://circleci.com/blog/what-is-continuous-integration/`.

[6] Stackify. (Sep. 2017). What are software metrics and how can you track them?, Stackify, [Online]. Available: `https://stackify.com/track-software-metrics/`.

[7] A. M. Aztarain. (Jun. 2018). Requirements traceability – the what, why, and how, netmind, [Online]. Available: `https://netmind.net/requirements-traceability/`.

[8] GCReady. (Aug. 2018). Configuration management, GCReady, [Online]. Available: `https://www.gcreddy.com/2014/07/configuration-management.html`.

[9] S. T. Help. (Nov. 2019). 10 best software configuration management tools (scm tools in 2019), Software Testing Help, [Online]. Available: `https://www.softwaretestinghelp.com/top-5-software-configuration-management-tools/`.

[10] M. Burns. (). Chapter 26 process improvement, [Online]. Available: `https://slideplayer.com/slide/12802795/`.

[11] (). Confluence, [Online]. Available: `https://wiki.sei.cmu.edu/confluence/display/seccode/Top%2010%20Secure%20Coding%20Practices`.

[12] (). Defensive application coding practices secrets, [Online]. Available: `https://www.hack2secure.com/blogs/defensive-application-coding-practices-secrets`.

[13] Guru99. (). What is response time testing?, Guru99, [Online]. Available: `https://www.guru99.com/response-time-testing.html`.

[14] M. J. Margaret Rouse. (). Capability maturity model (cmm), [Online]. Available: `https://searchsoftwarequality.techtarget.com/definition/Capability-Maturity-Model`.

[15] M. T. C. Team. (). Plan-do-check-act (pdca), Mind Tools Content Team, [Online]. Available: https://www.mindtools.com/pages/article/newPPM_89.htm.