Quality plan for To Do List

**File Name:** C:\Users\OEM\Documents\Open Polytechnic\BIT707 - Software Engineering\A1\BIT707\_A1\_5001428\_Quality\_Plan\_Template.docx

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description** | **Date** | **Author** |
| 1-0 | Initial Version | 23/03/2022 | Jason Norton |

# Overview

[Write a brief introductory sentence about what this plan covers]

# 1. Quality Management issues for this project

**Issue:**

If the architecture is not carefully considered or the initial code is not well written then adding extra features later on could prove difficult. This could severely limit the useful lifespan of the application

**Explanation:**

If data and methods are not strategically designed and encapsulated, then the code could quickly become messy and difficult for a developer to navigate or extend. This could increase complexity when it comes to adding further functionality later on, which in turn could lead to more errors. Likewise, if the architecture does not carefully implement abstraction principles then adding new functionality in later iterations could require significant refactoring which again could lead to more errors.

**Management plan:**

* Ensure each method/function only does one thing
* Use Model View Controller (MVC) design pattern to achieve abstraction between layers
* Consider adding a second layer of abstraction between the business logic and the model to make it even easier to change aspects of the model without having to refactor the core business logic as well.
* Ensure variable and function names are clear and meaningful
* Add comments to ambiguous or complex code to increase readability

**Issue:**

Dealing with dates on the calendar will be a complex task and may be prone to errors

**Explanation:**

The complexity of dealing with dates comes from the fact that days of the week fall on different dates each month, and different months have different amounts of days e.g. 28, 30, 31 and 29 for February on a leap-year. Also, dates change based on time, and time can be expressed in different ways (UTC, GMT, etc.) which may differ from the user’s local time zone. This could lead to an incorrect date for *‘now’* being displayed to the user depending on how time is measured in the database.

**Management plan:**

* Use existing calendar libraries/tools where possible instead of reinventing the wheel
* If it is not possible to use an existing library, the scheduling aspect of the application should be tested thoroughly.
* Store all timestamps in UTC but display them as *‘localtime’.*
* Where possible the scheduling system should be peer reviewed by another developer.

**Issue:**

The short timeframe allocated for design, implementation and deployment may lead to errors being overlooked or requirements not being met.

**Explanation:**

Building the initial prototype as a minimum viable product in a ‘quick to go live’ timeframe could lead to shortcuts being taken to get it deployed on time. Often when time is a factor aspects such as testing and evaluation may not be completed in as thorough manner as they should be. This could result in a sub-standard application that does not meet all of the requirements.

**Management plan:**

* Use a test-driven development approach to minimise the possibility of bugs
* Maintain a requirements traceability matrix
* Each piece of functionality should be checked against the requirements traceability matrix before being built to ensure it can be traced back to a specific requirement

# 2. Planned Quality Management Activities

1. **Planning Stage (Assignment 1)**
   1. Confirm that the client expects the finished product to be a simple java-based tool that will enable users to easily manage their tasks. If this is not the case, the project mission should be revised before repeating this activity.
   2. Confirm that the client would like the finished product to be an easy to use tool that allows company staff to visualise, schedule and prioritise their tasks, and that is highly extensible so that future iterations can be utilised by clients and integrated with other projects. If this would not meet the client’s expectation, then the project vision should be revised before repeating this activity.
   3. Check that the client agrees with the specified parameters for the minimal viable product, as outlined in the project charter. If this would not meet the client’s expectation, then the project scope should be revised before repeating this activity.
   4. Review planning documents and agreed project specifications to verify that the proposed To-Do list application is achievable within time, budget and resource constraints.
   5. Review all planning artifacts to verify that the proposed To-Do list application will meet all of the agreed requirements and achieve the mission.
2. **Requirements Engineering Stage (Assignment 1)**
   1. Check all user stories to ensure each one can be traced back to a specific requirement for the To-Do list application.
   2. Evaluate each specified requirement against the project scope as outlined in the project charter document. Where the requirement falls outside of the scope this should be discussed with the client.
   3. Have the client review prototypes and provide feedback to confirm whether or not the proposed To-Do list application will provide all of the functions they need it to.
   4. Review each requirement for the proposed To-Do list application and ensure it is clear how each one will assist the project mission and vision to be achieved.
3. **Design Stage (Assignment 2)**
   1. Verify that the design will work effectively by identifying the happy path in regard to core functions such as creating, editing, scheduling and viewing tasks then evaluating the happy path against the proposed design.
   2. Review styles and themes for consistency throughout the To-Do List application.
   3. Where possible, have an independent third party review the design of the proposed To-Do list application and provide feedback regarding design quality.
   4. Utilise prototypes to simulate functionality and confirm that all requirements have been met.
   5. Review proposed architecture to ensure the To-Do List design is technically sound.
4. **Construction Stage (Assignment 3)**
   1. Proof-read code to check for common errors and readability.
   2. Ensure the principles of Test-Driven development are followed so that the minimum viable product is delivered.
   3. Create a test suite with requirement traceability to ensure that each test relates to a single requirement, and that there is at least one test for every requirement.
   4. Perform integration testing to ensure cohesion throughout the To-Do List application.
   5. Complete functional and acceptance testing to establish whether or not the To-Do List implementation meets all specified requirements.
5. **Transition Stage (Assignment 3)**
   1. Check that user stories are working as expected in the To-Do list implementation
   2. Engage in system testing to confirm that the to-Do List application works well as a whole.
   3. Complete performance testing to ensure the To-Do list application operates well under stress.
   4. Re-run the test suite in the production environment to ensure the To-Do List application is functioning correctly and there are no conflicts.
6. **Post-Project Review (Assignment 3)**
   1. Review lessons learned from the To-Do List application project.
   2. Obtain feedback from the client to ascertain whether any further improvements can be made to the To-Do List application.
   3. Evaluate actual time, cost and resource usage against estimated values.
   4. Evaluate the To-Do List application against the success criteria specified in the project charter to ascertain whether the project was a success.

# 3. Discussion: Quality Management in different projects

The task of identifying and clearly defining all of the requirements necessary to create a successful product can be difficult. Misra and Fernandez-Sanz (2011), explain that “Gathering requirements from customers is always a challenge for software engineers. The quality of the product is highly affected by the absence of clear requirements.” (p. 326). This outlines the idea that often customers do not know exactly what they want, or what they think they want may not be what they actually need. Furthermore, clients may have implicit expectations that they fail to verbalise to the development team, for example if they assume it is obvious that the product should be able function in a particular way.

Misra and Fernandez-Sanz (2011) elaborate further on the complexities of this issue by outlining how changing business needs and technological advancements can lead to project requirements evolving both during and after the development process. This problem can lead to project failure if not factored into the development process from the outset, for example a long-term project may already be obsolete by the time it is developed if changing business needs are not addressed during the development process.

There are many examples of failed projects due to requirements being overlooked, misunderstood or not allowed to evolve to meet a company’s changing needs. The risk is further exacerbated when following a rigid development process, such as the waterfall approach, as any requirements identified late in the process cannot be easily catered for.

Misra and Fernandez-Sanz (2011) argue that these risks can be mitigated by following an iterative process and having developers work closely alongside the customer so that requirements can be dealt with incrementally. By delivering a minimal viable product after each iteration, the customer can ‘try out’ the system much eariler in the process and thus gain a better understanding of how it meets their needs, and where it falls short. This agile approach allows for a quick feedback loop between the customer and the development team which enables them to better identify requirements, quickly clarify ambiguities, and decreases the timeframe of delivering benefits to the customer, thus increasing the chances of project success.

Another common quality management issue that software projects have to contend with is a “lack of common understanding of goals and requirements amongst development team members” Misra and Fernandez-Sanz (2011, p. 326).

When working in any team, information must be communicated very clearly to prevent misinterpretation, and this certainly holds true in a software development context. Barak (2005) suggests that when deciding how to interpret written information, the underlying question that requires answering is ‘what meaning should be attached to the text?’. It is argued that the same holds true for any interpersonal communication, no matter the mode of delivery.

Throughout the software development lifecycle information must be passed from the customer to those responsible for planning, and from there to those responsible for design, implementation, testing, and evaluation. In a small team each person may be involved in several or even all of these phases, so it is often easier to ensure that everyone involved understands the project mission, goals and requirements. On the other hand, as a team increases in size it becomes easier for things to get ‘lost in translation’. The game known as ‘Chinese Whispers’ comes to mind here.

This issue is particularly pertinent in software development where it is becoming increasingly common for some members of the team to work remotely. Factors such as different time-zones, languages, and the technologies used to facilitate information sharing can all create barriers to effective communication and lead to inconsistent understanding between team members.

Without a common understanding between all project stakeholders some team members may attach a different meaning to specified goals and requirements for the project, increasing the risk that the product delivered may not be what the customer needed.

Creating user stories and establishing use cases provides an effective solution to this problem by giving context to project requirements and goals, thus providing a means of generating a normalised interpretation across the team.

Another way to help build a common understanding is through frequent team meetings, such as a ‘daily stand up’ meeting where the key tasks for the day are discussed to ensure everyone is on the same page.

A third common quality management issue arises when different teams, departments, or companies must be coordinated to achieve the project’s goals, especially if some aspects of the project are outsourced (Misra and Fernandez-Sanz, 2011).

Different departments or organisations are likely to follow different standard operating procedures and use different technologies or programming languages. This can lead to a breakdown in cohesion and result in integration issues later in the project. Likewise, where different workgroups or organisations are required to work together on a shared task, it can become unclear who is responsible for achieving that task. If a task is not completed satisfactorially this could lead to multi-directional finger pointing but nobody willing to wear the cost of resolving the issue.

One way to mitigate the risk in this regard is to maintain traceability documentation naming a specific person who is responsible for each task being completed. Establishing clear lines of communication is also important to ensure each team knows who they should be discussing issues with, and frequent meetings are important to ensure they remain on the same page.

Engaging in early integration testing can further diminish the risk of integration issues having a significant impact on the project later on.

Ambiguous requirements or documentation can also be interpreted differently by different teams or workgroups, thereby decreasing the chances of successfully meeting the project success criteria.

As previously discussed, establishing context around requirements and other development artifacts is essential to enabling a shared understanding between individuals in a team, and the same holds true when multiple workgroups or organisations are involved.

**References**

Barak, A. (2005). *Purposive Interpretation in Law.* Princeton University Press. https://doi .org/10.1515/9781400841264

Misra, S., & Fernandez-Sanz, L. (2011). Quality Issues in Global Software Development. In ICSEA, *Proceedings of the Sixth International Confrence on Software Engineering Advances* (pp. 325-330). IARIA. https://www.researchgate.net/publication/233 777217\_Quality\_Issues\_in\_Global\_Software\_Development