TASK

Quality Management

Visit our website

Introduction

WELCOME TO THE QUALITY MANAGEMENT TASK!

Developing high-quality software requires careful planning and attention to detail. This task covers the key areas you need to consider, including quality planning, product introduction and planning, measurements and metrics, standards, and component analysis. In exploring each of these topics, you'll gain a clearer idea of what it means to deliver software that meets industry standards and exceeds customer expectations.

INTRODUCTION TO QUALITY MANAGEMENT

Ever since the 1960s, when the first large-scale software systems were developed, problems with software quality were staple of the software profession. These problems continued to plague software engineering throughout the 20th century. The software that was delivered during this time was slow, unreliable, difficult to maintain and hard to reuse. This undesirable situation ultimately led to the development of formal techniques for software quality management. These software quality management techniques were developed from the methods used in the manufacturing industry and have led to significant improvements in the level of software quality.

Quality is something we all know, but it can be tricky to define. If you were looking at a pair of shoes, how would you know they were of high quality? Usually, we have a set of predefined standards, like genuine leather outer, solid stitching and a robust sole. We could also rely on some predefined measure of quality, like the reputation of the brand. In software engineering, when determining the quality of a product, we have a Quality Management (QM) team that will sign off on the quality of the end product. This team's primary tasks are to create a framework of processes and standards that will lead to a high-quality product, to apply these processes, and regularly check that they are being implemented.

As you might imagine, the QM team is most effective when they are completely separate from the development team. This helps them to keep an unbiased view of the product, thereby helping to uphold high standards for the quality of the product.

The QM team is also in charge of release testing. This means that they are in charge of the maintenance of system and requirements tests, as well as the test records that come from them. They then assess these records to see that the tests were carried out properly.

The QM team begins with **quality planning**: deciding which software qualities are the most important to assess and how they plan to assess them. This is the backbone of how the QM team will decide if the product is of high quality or not. This also helps the Software Engineers to decide what needs to be included in the development process that will achieve the outlined software qualities.

In his book on software management, **Humphrey** (1989) outlined the structure of a quality plan as follows:

- **Product introduction:** the product, its intended market, and the quality expectations for the product are defined.

- **Product plans:** the release dates and responsibilities for the product, as well as plans for distribution and product servicing, are defined.

- **Process descriptions:** the development and service processes and standards that should be used for product development and management are described.

- **Quality goals:** the quality goals and plans for the product are defined. This includes the identification and justification of critical product quality attributes.

- **Risks and risk management:** the key risks that might affect product quality and the actions to be taken to address these risks are defined.

SOFTWARE QUALITY

If one person were looking for smart shoes and another looking for hiking boots, their ideas on quality would be completely different — smart shoes would be terrible for hiking! In the same way, the notion of a high-quality product can mean different things to different stakeholders depending on what they are looking for the product to achieve. That is why it is important for the QM team to explore all requirements from all stakeholders, decide which should be included, and decide how they will assess if those requirements have been achieved.

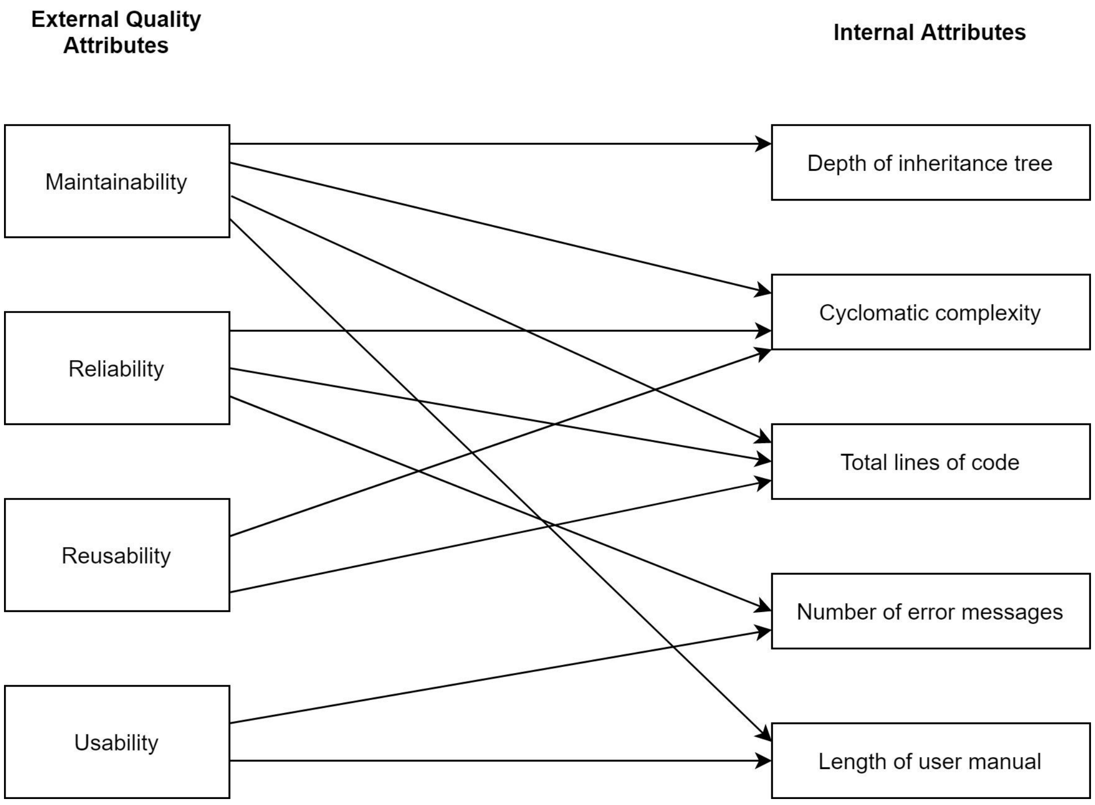
However, this is easier said than done. As we will see later in *Measurement and Metrics*, it is very difficult to decide a product's quality based on subjective information. Rather, we can use qualitative tests to determine if the system is functioning based on its requirements. This can be done through questions like, "Is the performance good enough under normal conditions?" or "Were all highlighted software standards followed?" Have a look at some important software quality attributes that need to be considered (Narang, 2015):

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability

Having a look at the list above, you might notice some are in direct competition with one another. For example, robustness is not really conducive to adaptability. That means that not all of the above can be achieved in equal measure within the same product. Therefore, it is up to the QM team to decide on the most important attributes to be tested that would lead to a high-quality product. This is the purpose of the **quality plan**.

MEASUREMENTS AND METRICS

Another difficulty the QM team may face is that the above requirements are difficult to measure because they are based on a developer's or user's personal experience with the product. For example, we can't put a numeric value on adaptability or security (think **functional vs. non-functional requirements**). Rather, the QM team can use more quantitative measures that loosely relate to these more qualitative requirements. For example, to look at reusability, the QM team could look at the number of error messages the program receives as well as how complex the code is, such as the levels of inheritance used. Have a look at some more examples below:



How External Quality Attributes Could be Measured with Quantitative Internal Attributes (Sommerville, 2016, p. 718)

The diagram above shows the software qualities on the left and the related software metrics on the right. A **software metric** is a characteristic of the system that can be measured quantitatively. There are two types of metrics, which are discussed below:

Process metrics: these are related to improving development and maintenance in the software process, such as the number of bugs found per line of code. This is where project managers can see if processes need to be changed. Measuring these metrics is based on the diagram above. You first identify the internal quantitative components that relate to this metric. Next, you combine all of these measurements to make one value. Finally, you assign this value to that particular metric. This value will then be compared to the company's established standards, which are discussed in the next section.

Product metrics

Product metrics highlight the attributes of the software, like the amount of lines of code in a program. Project managers leverage this data to assess the resources needed in case of alterations or additions to the system. These indicators are evaluated by pinpointing areas with inadequate quality, such as a module with notably high complexity. These will be the areas of the system that are most at risk of creating a low-quality product due to lack of understandability, maintainability, or adaptability, for example.

Product metrics also have their own subcategories:

- **Dynamic metrics:** refer to a set of measurement techniques that are executed during program testing or after the system has been released to users. These metrics determine attributes such as the time it takes to run the system, the number of system crashes, and the number of bug reports submitted by users. Unlike static metrics, dynamic metrics evaluate the behaviour of the program when executed to identify issues related to performance, functionality, and user experience.

- **Static metrics:** these refer to metrics that are measured during the development phase, such as average file length or the number of classes used. These metrics provide insights into the quality, complexity, and maintainability of the software codebase. They are called "static" because they analyse the code's structure, design, and complexity, rather than evaluating its behaviour when executed. By leveraging static metrics, software developers can pinpoint sections of code that require enhancements, including code that is challenging to maintain or vulnerable to errors.

There are various metrics that can be monitored to assess the performance and success of a software system. Here are some other examples:

1. **Active Users**: The number of active users is a crucial metric that reflects the popularity and engagement of the software product. This metric can be measured on a daily, weekly, or monthly basis, and it helps determine the system's adoption rate and user satisfaction level.
2. **User Engagement**: User engagement metrics measure the frequency and duration of user interaction with the software product, such as the number of sessions per user, the time spent on the system, or the number of clicks or views.
3. **Conversion Rate:** is a metric that calculates the proportion of users who successfully accomplish a specific goal or action, like purchasing a product

or completing a form. This metric is a useful indicator of how effective the software product is in achieving its intended purpose.
4. **Website Traffic**: Website traffic metrics measure the volume and quality of visitors to the software product's website. Metrics like page views, unique visitors, bounce rate, or time on site can help identify areas for improvement in user experience or content.
5. **Performance**: refers to how well the software product functions and stays available to users. Factors like response time, uptime, and error rate can help determine how reliable and scalable the system is.

Here's a **thought-provoking article** on software metrics and how best to track them. Bookmark it for easy reference when you begin thinking about product metrics more critically as a professional developer.

SOFTWARE STANDARDS

Once the quality plan has been created and the team has decided what they are going to measure to ensure a high-quality product, they then look at software standards to decide when they will know quantifiably when their product is of high quality. **Software standards** describe the standards that should be applied to the development process to ensure the product's quality. These are based on trial and error from previous projects that the company has completed. From there, they create a framework to benchmark the quality of a new product. Because they are standardised, it ensures continuity for others who join or take over the project. These software standards can be applied to both products and processes.

As the name suggests, product standards relate to the actual product being developed, including the requirements document, the software documentation and the coding language itself. On the other hand, process standards relate to the process the development team follows, such as the design phase, development phase, and the validation process. It will be up to the QM team to decide what standards to use. They need to ensure that they will lead to greater quality of the product and are able to adapt based on different development software.

Software standards can be locally developed but should still be based on international standards, such as the **ISO 9001 Standards Framework**.

CODE REVIEWS

Once the product has been developed and is now in the testing phase, code reviews are a way for the QM team to check the quality of the product. During this process, they will look at the code itself, the process and software documentation, and any records made to find potential errors and omissions. This assesses the verification and validity of the product.

Throughout the software development lifecycle, code reviews are usually carried out during different stages such as coding, integration, and testing. Various approaches like peer reviews (sometimes called 'walkthroughs') can be used to conduct these reviews.

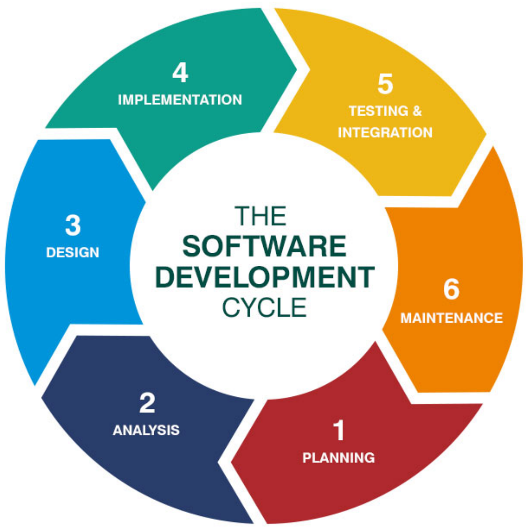


Image source: **https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/**

Code reviews in software quality management are advantageous because they allow developers to detect and resolve defects early in the development cycle. They can reduce the overall time and effort required in the build. Moreover, they improve the overall consistency, readability, and maintainability of the code quality.

It is worth mentioning at this point that agile development does not work well with a formally structured quality management system. This is because there is very little formal documentation in agile programming — rather, reviewing takes the form of paired programming or informal inspections. It is always worthwhile to have more than one person look at a piece of code during the development process.

TESTING

In addition to code review, testing is another crucial aspect of software quality management. The purpose of testing is to assess a software product and ensure that it satisfies its intended objectives and meets the prescribed criteria for functionality and performance.

Once the product has been reviewed; the team collaborates to look through the code line by line to find issues. During this time, the team could use a checklist, formal or informal, of common errors, anomalies or general common problems that could be found. As with the review process, agile development does not usually do formal inspections; rather, the programmer will have someone else look over the code on an informal basis.

SOFTWARE COMPONENT ANALYSIS

Part of the inspection process is software component analysis. This comprises each component of the software being analysed separately based on the agreed-upon metrics. The values of these metrics are then compared to different components as well as historical standards based on the company's previous projects. The aim is to find deviant values, which could indicate low quality at some part of the system. The team then goes to review these anomalies to see if they are, in fact, low quality. The data from this process is then kept on record as historical data for future projects.

Compulsory Task

Answer the following:

- Briefly describe possible quality standards that might be used for these real-world products:
    - Mobile game app
    - Banking application
    - Database for a medical organisation

- Explain why a high-quality software process should lead to high-quality software products. Discuss possible problems with this system of quality management.

- Explain why program inspections are an effective technique for discovering errors in a program. What types of errors are unlikely to be discovered through inspections?

- Why is it difficult to validate the relationships between internal product attributes and external attributes?

Completed the task(s)?

Ask an expert code reviewer to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES

Humphrey, W. 1989. *Managing the Software Process*. Reading, MA: Addison-Wesley.

Narang, R. (2015). *Software Engineering — Principles and Practices*. New Delhi: McGraw-Hill Education.

Sommerville, I. (2016). *Software Engineering* (10th ed., pp. 700-729). Essex: Pearson Education Limited.