Module 15: Quality Management, Process Improvement

Objectives:

1. Understand what quality management is and why it is important in software engineering.
2. Understand the importance of standards in the quality management process.
3. Know the difference between quality assurance and quality control.
4. Understand the rationale behind using process improvement as a way to improve product quality.
5. Understand the cyclic nature of the process improvement process.
6. Be introduced to the Goal-Question-Metric approach to process improvement.
7. Be introduced to the SEI's original Capability Maturity Model, as well as the newer, integrated, CMMI Model (both staged and continuous versions).

Assigned Reading:

Sommerville, chapter 24

Web Chapter 26 (Process Improvement, available on the companion website: http://iansommerville.com/software-engineering-book/files/2014/07/Ch_26_ProcessImprovement.pdf).

Assignment:

Complete the Group Project Postmortem Assignment, located in the Module 15 section.

Quiz:

There is no quiz for Module 15.

Lecture:

Quality Management

Software quality has been a recurrent theme throughout this course. We want to be able to produce software that is high quality. Precisely what "high-quality" means can vary, as we have seen, and the definition can be tailored to suit specific applications or systems. Users want the software they use to work correctly; to be reliable; to protect sensitive data; to be easily usable; to perform well; to not corrupt other installed software; etc. But quantifying these characteristics is not so easy. For example, if an application crashes unexpectedly every time it's used (as Microsoft Visio 2010 does on my Windows 7 system), it will be for all practical purposes useless, regardless of the measures of its other quality characteristics. But if an application only crashes once every 500 times it is used, does that mean the application is not high-quality?

Just as quality measures can be applied to a finished software product, they can also be applied to the process used to build that product. Over a century ago, it was found that adopting certain methods and practices could lead to more efficient manufacturing of goods. The assembly line, for example, allows many types of products to be mass assembled, drastically increasing the number of units that can be produced per unit of time. Using components that have been designed and produced according to consistent standards promotes greater repairability of products, as well as greater interoperability between related products.

Management processes can also be measured and optimized, although it is somewhat harder to identify and quantitate specific management characteristics and processes. We've discussed a few broad categories of team structures, for example, where decisions might be made democratically, or made by a single individual through an established chain of command. One of the major problems with assuring quality in management relates to the fact that not all individuals will perform at their highest level for the same set of management processes; in other words, there is no "one-management-process-fits-all" solution. Management quality can also extend to aspects that are not directly related to product development. One good example here is the idea adopted by several software companies to allow developers to work in a relatively non-structured manner. A software team might work on a project for a couple of hours, then play games in an on-site game room for a couple of hours, and finally return to work for a few more hours on the project. But how can an approach like that be readily quantitated and standardized?

A common misconception is that the quality of a software product directly corresponds to the quality of the development process. For goods that are simply manufactured, there may be a close correspondence between the quality of the process and the quality of the product. But as we have noted before, software is not simply manufactured. Software is designed, and in most cases the same software is never created twice, so software cannot be mass produced like cars, electronic equipment, or any other type of durable or consumable good.

Most large software development organizations have dedicated groups of people who deal with ensuring quality. In general, quality management has historically been separated into two broad categories:

1. **Quality Assurance**, which covers processes and standards that, in theory, should lead to the creation of high-quality products.
2. **Quality Control**, which verifies whether or not the finished product meets acceptable quality standards.

With respect to software engineering, quality control is usually the responsibility of those who test the software. Software testing can be performed in-house, through the use of third-party, independent testing groups, or both. Quality assurance covers the entire management and engineering processes like an umbrella, to ensure that not only is high-quality software built, but that the processes used for building the software and for managing those who build the software, are of high quality as well.

Software Quality

We covered one possible set of characteristics that may be used to gauge software quality, known as McCall's Quality Model, back in Module 1. For convenience, those characteristics are:

1. **Correctness** – without a doubt the software does precisely what it is supposed to do, and everything it is supposed to do
2. **Reliability** – the probability that, over time, the software will perform correctly
3. **Efficiency** – does it perform reasonably with respect to speed, resource consumption, etc.?
4. **Integrity** – how stable is it? How easily can it be penetrated by a malicious hacker?
5. **Usability** – how easy is it for the average end user to use?
6. **Maintainability** – how easy is it to upgrade and fix?
7. **Testability** – how easy is it to test?
8. **Flexibility** – how well can it handle unforeseen circumstances? If functionality must be changed, how easily can it be done?
9. **Portability** – will it operate in different environments (e.g., different operating systems)?

10. **Reusability** – can parts of the software be reused to build other software?

11. **Interoperability** – will it interact well with other software?

These characteristics are all good choices, and ideally we would like to achieve high scores for all of them. The reality, though, is that it usually isn't possible to optimize all of the characteristics. Optimizing one characteristic often comes at the cost of another, so in the end decisions have to be made regarding which of the characteristics are most important, and which tradeoffs are most acceptable. Some efficiency, for example, might have to be sacrificed in order to achieve the minimum acceptable level of integrity. Part of the job of the quality assurance people is to define what quality means for the organization in terms of whatever quality characteristics are used. Each characteristic will usually consist of several metrics (ratio of two measures), and minimum values for those metrics will be set. How well the finished product meets those minimum requirements will determine its quality rating.

Standards

We've already talked about the importance of standards for software development, so it should come as no surprise that standards are equally important to quality assurance. I think Sommerville sums up the importance of standards very nicely in the three points he gives in Chapter 24:

1. Standards capture the wisdom that is of value to the organization. The experience gained from past projects, along with knowledge gained from other sources, will help to reduce future mistakes, and will help the organization hone its development process. Standards, in effect, are simply the documentation of this experience.

2. Standards provide a framework for defining what quality means, by helping to remove subjectivity from the metrics used to determine quality. For example, in terms of reliability, we know there are probably going to be occasional failures, but what frequency of failures is allowed to still be able to say software is high-quality or not? This value can be quantitated arbitrarily, justified according to some rationale, and set as a standard. Future experience can then be used to optimize this value until it becomes reasonably attainable.

3. Standards assist continuity. One thing we've discussed before is that any given project may be worked on by more than one team, especially if the project is very large or if the final product has a long life span requiring significant maintenance. Standards will help to ensure all the developers who work on a project are on the same page, so to speak, and they make it easier for the transition from one team taking over from another.

Standards apply to processes as well as to products. You probably already have a feel for what might be used for product standards, but process standards are a little more difficult to define. In fact, there is an international standard, ISO 9000, that is essentially a framework for developing process standards for just about any type of business. Although ISO 9000 can be used by software development organizations, there is another standard, ISO 9001, that is tailored to organizations that design, develop, and maintain products, as opposed to simply manufacturing products. The most recent version of ISO 9001 was written in 2008 and is referred to as ISO 9001:2008. There are also other ISO standards that are specific for software engineering, grouped under the JTC-1/SC 7 heading. "JTC" stands for "Joint Technical Committee", and JTC-1 deals specifically with the realm of information technology. Subcommittee (SC) 7 of JTC-1 handles all standards related to software and systems engineering. ISO certification has become an important advertising and marketing tool. Organizations that have been successfully certified as being ISO-compliant proudly display this fact, in order to convince both existing and potential customers to have confidence in the organization and, by extension, its products.

The ISO 9001 standard does not prescribe specific processes an organization must use to ensure quality. Instead, the standard defines a set of core process areas. Implementation of certain process areas is mandatory, while others need only be addressed, but not implemented. It is up to the organization to decide what specific processes will be used to address each process area, define how data about these processes will be collected and maintained, and ultimately used to assess the effectiveness of the processes. Whatever the organization comes up with must be reviewed and approved by auditors, who will periodically visit the site to verify the processes are being followed as specified. A failed audit could mean revocation of the organization's ISO certification, which is considered a major setback for most organizations.

Process Improvement

Software processes, like most things, can be improved over time. An organization starts off with the best process they can come up with initially, and optimizes it as new information is obtained from each project, from the expertise of employees and other people, and from the results of research. Theoretically, as the process is optimized, software quality should increase. The two main schools of thought regarding process optimization are the **process maturity approach** and the **agile approach**. Process maturity tends to involve significant overhead as processes are documented, data about the processes are collected and analyzed, and standards are created. None of these activities are directly related to the development of a software product. The agile approach, as we have seen, tries to achieve greater quality by reducing overhead that is considered non-essential, in order to rapidly deliver functional software increments and adapt to changes in requirements. For the remainder of this module, I will focus on the process maturity approach.

The Process Improvement Process

There are three primary steps involved in the maturation of a process, which are performed cyclically:

1. Measure the process using characteristics that can be quantitated. Typically the measures are used to derive useful metrics that describe the effectiveness of the process. There are three main types of process measures and metrics:
    1. Time required to complete a particular process task (e.g., calendar time, CPU time)
    2. Resources required to complete a particular process task (e.g., person-months, # of computers/task)
    3. Number of occurrences of a particular event (e.g., faults/LOC, faults found/testing procedure)
2. Analyze the data from the measures and derived metrics.
3. Make changes to the process that are predicted to result in improvements over the current results.

In practice, completing these steps successfully isn't always as easy as it sounds. Sometimes it is difficult to determine what measures and metrics will prove useful in improving a process, or even to figure out how to improve a particular process in the first place. A very direct approach to accomplishing this is known as the Goal-Question-Metric (GQM) paradigm, which Sommerville talks about in Chapter 26. Basically, you start off with a goal—reduce the time required for testing, for example. You then as a series of questions related to each goal. These questions should be designed to clarify the goal, and to provide a foundation for the third step, which is determining what should be measured, and which metrics should be used to assess whether a change made to the process is, in fact, improving the process.

The Capability Maturity Model

Back in the 1980's the Software Engineering Institute (SEI), a federally funded research organization, undertook a study to determine the capabilities of the American software industry, with the goal of improving the software engineering capabilities of the United States. They released a five-level model called the Capability Maturity Model (CMM). The purpose of the model was to provide a framework and guidelines for software engineering companies to mature their business processes. After a time, an additional level was added to account for companies that basically had no defined process at all. The six-level CMM is outlined in Table 15.1. At the lowest level, Level 1, there is essentially no defined process at all. At Level 2, there is a defined process, but it is ad hoc—unorganized. It is easy for a company to achieve Level 2. At Level 3, the process is organized, and for the first time a company can have some assurance of successfully completing projects, whereas at Level 2 success is more or less hit or miss. It begins to get more difficult to rise up to the next level, Level 4. Achieving Level 4 requires that the process be standardized, and all activities must be capable of being monitored. At Level 5, quantitative data are collected about process activities, and these data are used when making future decisions about the process or individual process activities. The final level, Level 6, is very difficult to achieve. At Level 6, all process improvement is driven by feedback from the data collected on all process activities.

Level	Name	Brief Description
1	Not Performed	One or more goals are not performed at all.
2	Performed	A process exists, and all goals are performed, but there is no clear organization; the process is ad hoc.
3	Managed	Organizational policies are in place. Success in past projects can be repeated in future projects.
4	Defined	Focus is on organizational standardization. All individual process activities, not just the overall process itself, can be monitored.
5	Quantitatively Managed	Quantitative measurements and data are collected from goals and are used in making decisions for future process activities.
6	Optimizing	Feedback from data gathered from all goals is used to drive process improvement.

Table 15.1. Summary of the various levels of the Capability Maturity Model.

Over time, other capability models were developed by different organizations and individuals. Some models were complete in their own right, while others simply extended the existing CMM model. The various models all had their strengths and weaknesses, and eventually a new model was developed that attempted to integrate the best features of these different models. The result came to be known as CMMI, or Integrated Capability Maturity Model. Two versions of CMMI exist: 1) the **Staged CMMI Mode**l, and 2) the **Continuous CMMI Model**.

The Staged CMMI Model is completely compatible with the original CMM model, and allows companies to broadly classify themselves as one of five discrete levels, or stages. It differs from the six-level CMM model in that levels 1 & 2 from that model are combined into a single level. The backwards compatibility with the older CMM model is the greatest advantage of the Staged CMMI Model. It suffers from the fact that it can be overly prescriptive with respect to what goals must be met for each level. It also behaves similarly to the waterfall model of software development in that all the goals at the current stage must be met before moving on to the next stage.

The Continuous CMMI Model does away with the five discrete levels of classification, instead focusing on classifying individual process activities on a level from 0 to 5, that correspond roughly with levels 1 to 6 of the six-level CMM model. Rather than being overly prescriptive in terms of which processes must be used, the continuous model allows companies to tailor the selection of processes to best fit its own requirements. The broad categorization of the staged model can mask an organization's true capability maturity, since an organization may be advanced in many areas, yet held back to an overall low level due to poor maturity in one or two process activities. The continuous model, therefore, provides a much more accurate, finer grained, picture of an organization's overall process capability maturity level.