UNIT 5:
Project Closure and Software Quality Assurance

Goals of Software Quality Assurance :

- Quality assurance consists of a set of reporting and auditing functions.
- These functions are useful for assessing and controlling effectiveness and completeness of quality control activities.
- It ensures management of data which is important for product quality.
- It also ensures that software which is developed, does it meet and compiles with standard quality assurance.
- It ensures that end result or product meets and satisfies user and business requirements.
- It simply finds or identify defects or bugs, and reduces effect of these defects.

Measures of Software Quality Assurance :

There are various measures of software quality. These are given below:

1. **Reliability –**
   It includes aspects such as availability, accuracy, and recoverability of system to continue functioning under specific use over a given period of time. For example, recoverability of system from shut-down failure is a reliability measure.
2. **Performance –**
   It means to measure throughput of system using system response time, recovery time, and start up time. It is a type of testing done to measure performance of system under a heavy workload in terms of responsiveness and stability.
3. **Functionality –**
   It represents that system is satisfying main functional requirements. It simply refers to required and specified capabilities of a system.
4. **Supportability –**
   There are a number of other requirements or attributes that software system must satisfy. These include- testability, adaptability, maintainability, scalability, and so on. These requirements generally enhance capability to support software.
5. **Usability –**

It is capability or degree to which a software system is easy to understand and used by its specified users or customers to achieve specified goals with effectiveness, efficiency, and satisfaction. It includes aesthetics, consistency, documentation, and responsiveness.

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR): Some of these are:

- Useful to uncover error in logic, function and implementation for any representation of the software.
- The purpose of FTR is to verify that the software meets specified requirements.
- To ensure that software is represented according to predefined standards.
- It helps to review the uniformity in software that is development in a uniform manner.
- To makes the project more manageable.

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely. FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise. Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successful only if it is properly planned, controlled and attended.

The review meeting: Each review meeting should be held considering the following constraints- *Involvement of people*:

1. Between 3, 4 and 5 people should be involve in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1. Accept the product without any modification.
2. Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3. Accept the product provisional (minor errors are encountered and should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Software Reliability

Software Reliability means **Operational reliability**. It is described as the ability of a system or component to perform its required functions under static conditions for a

specific period.

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

MAINTAIBILITY

Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced. Software maintainability is important because it is approximately 75% of the cost related to a project!

Transportability

Transportability usually refers to the ability to migrate from one service provider to another service provider—versus from one product vendor to another, which is a matter of interoperability. While interoperability is an important operational and technical requirement, it tends to be a point-in-time decision. You choose a vendor and go forward with that

product, allowing for occasional patches and upgrades that may or may not be applied.
Transportability of service is different from interoperability of products because the IoT system is a constantly changing the array of services that may even vary from day to day due to the myriad of externalities that vary: weather, staffing, holidays, environmental conditions, the cost of power, the cost of coffee in the cafeteria, traffic that day, and so on. As much as service providers strive incredibly hard to make services such as device management, networks, clouds, and DCs stable and predictable, they are prone to small fluctuations all the time and wholesale outages are worryingly common.

Interoperability

Interoperability is the property that allows for the unrestricted sharing of resources between different systems. This can refer to the ability to share data between different components or machines, both via software and hardware, or it can be defined as the exchange of information and resources between different computers through local area networks (LANs) or wide area networks (WANs). Broadly speaking, interoperability is the ability of two or more components or systems to exchange information and to use the information that has been exchanged.
There are two main types of interoperability:

1. Syntactic Interoperability: Where two or more systems are able to communicate and exchange data. It allows different software components to cooperate, even if the interface and the programming language are different.
2. Semantic Interoperability: Where the data exchanged between two or more systems is understandable to each system. The information exchanged should be meaningful, since semantic interoperability requires useful results defined by the users of the systems involved in the exchange.

Efficiency

In general terms, **efficiency** is defined *as the ratio of the amount of energy used to the amount of work done to create an output*. But when it comes to **software development**, “efficiency” is the *amount of software developed or requirement divided by the number of resources used like time, effort, among other examples*. In other words, efficiency

usually means that waste is avoided. Waste in software development involves defects, waits, overproduction, unused creativity, or extra revisions, to mention a few.

When we talk about software development efficiency, each part of the process of software development must be taken into account to reflect the efficiency of the team. Efficiency within software development drives faster product life cycles, shorter time to market, and eventually a higher end result. Software team leaders can employ many metrics to measure efficiency throughout the process. However, the definitive measure of a software product's performance is **whether it meets the needs of the end-users and improves the company's bottom line**.

Given that efficiency in software development is not directly traceable or measurable, we need to determine factors that will help us measure the performance of your software team.

SQA PLAN

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly. Software Quality Assurance is a process which works parallel to development of software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

Software Quality Assurance has:

1. A quality management approach
2. Formal technical reviews
3. Multi testing strategy
4. Effective software engineering technology
5. Measurement and reporting mechanism

Major Software Quality Assurance Activities:

1. **SQA Management Plan:**
   Make a plan for how you will carry out the sqa through out the project. Think about which set of software engineering activities are the best for project. check level of sqa team skills.

2. **Set The Check Points:**
   SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

3. **Multi testing Strategy:**
   Do not depend on a single testing approach. When you have a lot of testing approaches available use them.

4. **Measure Change Impact:**
   The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.

5. **Manage Good Relations:**
   In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of sqa team with programmers team will impact directly and badly on project. Don't play politics.

Benefits of Software Quality Assurance (SQA):

1. SQA produces high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.
4. SQA is beneficial in the condition of no maintenance for a long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

Disadvantage of SQA:
There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

Factors to measure software development efficiency

Determining which metrics will ensure that a company's software team is being efficient as possible is an important stage. It should be noted that the efficiency rate is not the same for different developers and teams. However, these are some of the factors that allow efficiency to be measured:

- Meeting times
- Lead time
- Code Churn
- MTTR and MTBF
- Impact

Metrics

1. **Meeting times**

   What if the team frequently lengthens its meeting times? Even if a problem discussion is happening, it is recommended that every meeting has a fixed time to share project status and solve any possible blockers. Team leaders or Scrum Masters should analyze the fixed meeting time and the actual meeting time at the end of the sprint sessions, in order to figure out if, the team is being efficient for you or not.
2. **Lead time**
   Lead time is the time period from product conception to its final delivery to the customer. This time depends on the type of project and the number of engineers required (which affects the cost of the project). By tracking your team's lead times, you can predict more accurately the time to market for similar future projects.
3. **Code Churn**
   Code Churn is defined *as the percentage of time developers spend editing, adding, or deleting their own code*.

   Churn allows software managers and other project stakeholders to control the software development process, especially its quality. It can also help you identify problems with individual developers. For example, a high churn rate may indicate that the team is experiencing difficulty in solving a particular task, that there's a need to do some rework, among others.
4. **MTTR and MTBF**
   MTTR stands for **Mean Time To Repair** and MTBF for **Mean Time Between Failure**s.

   The development and delivery of a well-performing software product involve knowing how to solve problems such as bugs and carry out efficient modifications. MTTR and MTBF help to manage loose ends also after the delivery of the software/requirement.

   MTTR is defined as the average time required to fix a bug or repair a problem. MTBF records failures caused by design constraints. To know how to calculate them, we have:

   MTTR = (Total maintenance time)/(Total number of repairs)
   MTBF = (Total uptime)/(Total number of failures)

   These metrics are valuable considering that taking too long to repair can affect the performance of the business.
5. **Impact**
   Impact helps to measure how changes made to the code affect the overall project. It is also a measure of how those changes affect the developers who carry them out. This is

an important metric, as code modifications can prolong the product’s time to market, thus making it more costly. In addition, developers could be spending too much time making certain changes, which could negatively affect their productivity and increase delays. As a result, large changes such as those involving more code or files, and changes that are more difficult to perform, have higher impact scores.

What Is Project Closure?

The last step of a project is called project closure. Project managers use this term to indicate that a customer, stakeholder, or client has approved project deliverables for use. There must be a plan in place for continuing maintenance of the project or product.

To ensure a successful conclusion to any endeavor, a thorough assessment of the project will be conducted by its management. During this phase, the management team will participate by providing observations and comments that will be compiled in a document titled "lessons learned." This serves as a blueprint for future endeavors.

Closing a project involves more than simply clearing off all the paperwork, completing any vendor contracts, and freeing the team to work on new projects. A project's end-ofproject review ensures that all of the project's goals have been completed and that any remaining concerns, such as risk, have been resolved.

The Role of Closure Analysis

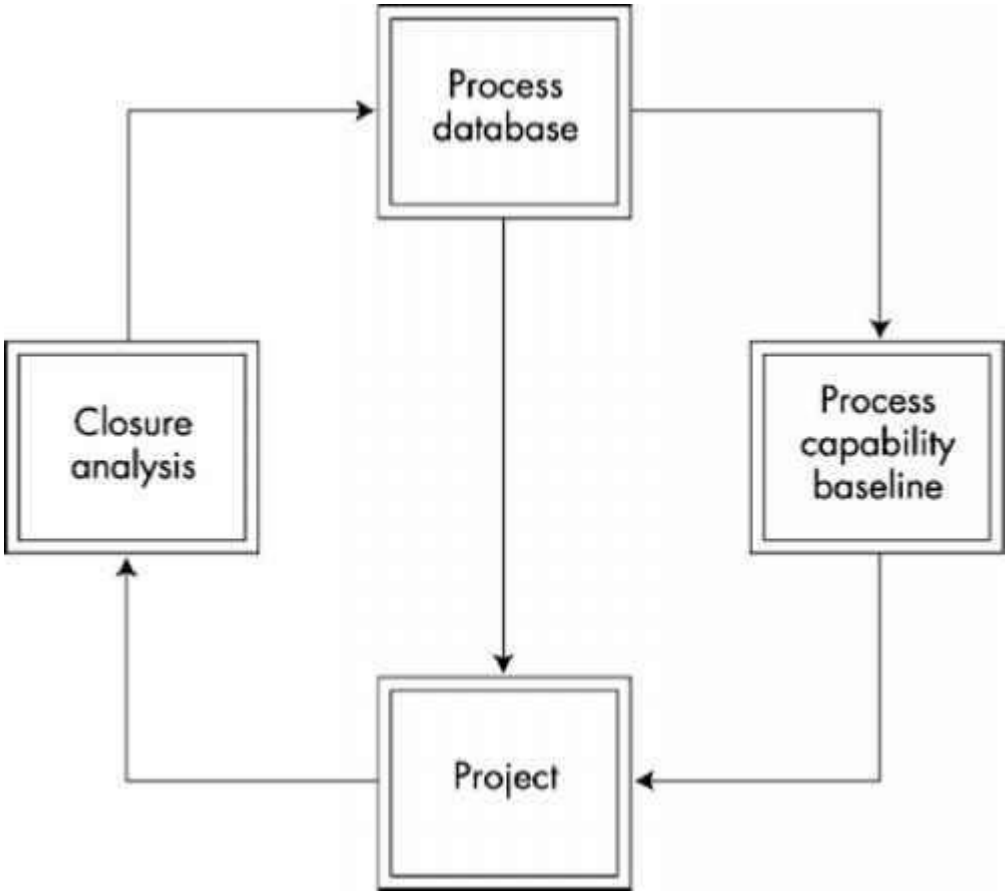
Last Updated on Sun, 15 May 2022 | Managing Software

The objective of a postmortem or closure analysis is "to determine what went right, what went wrong, what worked, what did not, and how it could be made better the next time."2 Relevant information must be collected from the project, primarily for use by future projects. That is, the purpose of having an identified completion analysis activity, rather than simply saying, "The project is done," is not to help this project but rather to improve the organization by leveraging the lessons learned. This type of learning can be supported effectively by analysis of data from completed projects. This analysis is also needed to understand the performance of the process on this project, which in turn is needed to determine the process capability.

As noted earlier, the data obtained during the closure analysis are used to populate the process database (PDB). The data from the PDB can be used directly by subsequent projects for planning purposes. This information is also used in computing the process capability, which is used by projects in planning and for analyzing trends. Figure 12.1 illustrates the role of closure analysis.

Figure 12.1. The role of closure analysis

Figure 12.1. The role of closure analysis



Earlier chapters discuss the types of data generally collected in a project and describe the collection methods. The amount of raw data collected in a project can be quite large. For example, a project involving five people and lasting for 25 weeks will have 125 entries for weekly effort, data for about 250 defects (assuming about 0.05 defects injected per person-hour), data on many change requests, various outputs, and so on. Clearly, these data will be of limited use unless they are analyzed and presented within a proper framework and at a suitable level of abstraction. Closure analysis aims to accomplish this goal.

After data analysis and extraction of all lessons learned from the analyses, the results should be packaged so that they can be used by others (packaging is the last step in the quality improvement paradigm6). Furthermore, to leverage this information, project processes must be constructed so that their execution requires the effective use of data. It can be argued, however, that even if others do not learn from the packaged information, the project personnel will have consolidated their experience and will carry the lessons learned from the analysis into future projects.2 In other words, a closure analysis is useful even if others do not directly gain from it.

Closure Analysis Report

This section briefly discusses the major elements in an Infosys project closure analysis report; later, we present the closure report of the ACIC project. The contents of this analysis report form a superset of the data that are put in the PDB. The PDB contains only those metrics data that are needed often by projects and whose use is required by the current processes. The analysis report, however, may capture other data that might shed light on process performance or help to better explain the process.

General and Process-Related Information

The closure report first gives general information about the project, the overall productivity achieved and quality delivered, the process used and process deviations, the estimated and actual start and end dates, the tools used, and so on. This section might also include a brief description of the project's experience with tools (detailed "experience reports" are put into the BOK system). The information about tools can be used by other projects to decide whether use of the tool is warranted. It can also be examined to identify tools that have good advantages and to propagate their use throughout the rest of the organization.

Risk Management

The risk management section gives the risks initially anticipated for the project along with the risk mitigation steps planned. In addition, this section lists the top risks as viewed in the post-project analysis (they are the real risks for the project). This information can be used by later projects and can be used to update risk management guidelines. Notes may also be provided on the effectiveness of the mitigation steps employed.

Size

As discussed in Chapter 6, many projects use the bottom-up method for estimation. In this method, the size of the software is estimated in terms of the number of simple, medium, or complex modules. Hence, this size is captured along with the criteria used for classification (different projects may use different criteria). Data on both the estimated size and the actual size are included.

For normalization purposes, the productivity of a project is measured in terms of function points (FP) per person-month. Although FP can be counted by studying the functionality of the system, at closure time it is computed from the measured size in lines of code (LOC). If multiple languages are used, we simply add the sizes (in FP) of the modules in different languages. Strictly speaking, function points (unlike lines of code) are not an additive measure. Because we are measuring only the size of the complete system in FP, however,

this approach is equivalent to converting all LOC counts into an LOC count of some "universal" language and then converting that size into FP. Furthermore, because of the inherent limitations of software metrics and their use, some inaccuracies are acceptable, provided that the methods are used consistently. The size in FP is also captured in the closure analysis report.

Effort

The closure analysis report also contains the total estimated effort and the actual effort in person-hours. The total estimated effort is obtained from the project management plan. The total actual effort is the sum of the total effort reported in all WARs submitted by the project members, including the project leader. If the deviation between the actual and the estimated values is large, reasons for this variation are recorded.

For each of the major steps in the process, the total actual effort and estimated effort for the stage are captured, too. This information can be useful in planning, and it is a key input in forming the PCB. For each stage, where possible, the effort is separated into the effort for the task, for the review, and for the rework. The WAR codes described earlier in the book permit this separation. The distribution of effort in the various phases can then be computed and recorded. The separation of effort between task, review, and rework aids in identifying the scope of productivity improvement.

The cost of quality for the project is also computed. It measures the cost of all activities that directly contributed to achieving quality. The cost of quality can be defined in many ways; here it is defined as the percentage of the total effort spent in review, testing, rework to remove defects, and project-specific training.

Defects

The defects section of the closure analysis report contains a summary of the defects found during the project. The defects can be analyzed with respect to severity (percentage of defects that were major, minor, or cosmetic), stage detected (percentage of total detected defects detected by which activity), stage injected (which activity introduced what percentage of total defects), and so on. Injection rate and defect distribution are also determined.

The defect removal efficiency of a defect removal task is defined as the percentage of total defects that existed at the time of execution of the task that are detected by the execution of the task. This metric is useful for determining which quality activities need improvement. The closure report gives the defect removal efficiency of the major quality control tasks, as well as the overall defect removal efficiency of the process. Other analyses of defect data may also be included. Sometimes, a separate analysis of the review data may be performed. The estimated versus actual defect levels are also analyzed.

Causal Analysis

When the project is finished, the performance of the overall process on this project is known. If the performance is outside the range given in the capability baseline, there is a good chance that the variability has an assignable cause. Causal analysis involves looking at large variations and then identifying their causes, generally through discussion and brainstorming.

Process Assets

In addition to the metrics data, other project artifacts are potentially useful for future projects. Chapter 2 discusses the use of process assets. These process assets are collected at project closure. The potential entries to the BOK are also identified during closure, although they are submitted later.