UNIT-V

METRICS FOR PROCESS

Software measurement:

It can be categorized in two ways:

- Direct measures of the software process – cost and effort applied
- Indirect measures of the product – include functionality, quality, complexity, efficiency, reliability, maintainability.

Size oriented metrics:

LOC Metrics

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

Following are the points regarding LOC measures:

1. In size-oriented metrics, LOC is considered to be the normalization value.
2. It is an older method that was developed when FORTRAN and COBOL programming were very popular.
3. Productivity is defined as KLOC / EFFORT, where effort is measured in person-months.
4. Size-oriented metrics depend on the programming language used.
5. As productivity depends on KLOC, so assembly language code will have more productivity.
6. LOC measure requires a level of detail which may not be practically achievable.
7. The more expressive is the programming language, the lower is the productivity.
8. LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
9. It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.
10. These metrics are not universally accepted.

Based on the LOC/KLOC count of software, many other metrics can be computed:

a. Errors/KLOC.
b. $/ KLOC.
c. Defects/KLOC.
d. Pages of documentation/KLOC.
e. Errors/PM.
f. Productivity = KLOC/PM (effort is measured in person-months).
g. $/ Page of documentation.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
.	.	.	.	.	.		
.	.	.	.	.	.		
.	.	.	.	.	.		

Function – oriented metrics:

Function-Oriented Metrics are also known as **Function Point Model**. This model generally focuses on the functionality of the software application being delivered.

These methods are actually independent of the programming language that is being used in software applications and based on calculating the Function Point (FP). A function point is a unit of measurement that measures the business functionality provided by the business product.

Calculating Function Point :

$$\text{Function Point (FP)} = \text{Count total} * [0.65 + (0.01 * \text{Sum}(F_i))]$$

Reconciling LOC and FP metrics:

Reconciling LOC and FP Metrics

Programming Language	LOC per Function point			
	Avg.	Median	Low	High
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP 69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400
Cool:Gen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel47	46	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/Cool:Gen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	—

15

Object – oriented metrics:

Lines of code and functional point metrics can be used for estimating object-oriented software projects.

object-oriented projects, different sets of metrics have been proposed. These are listed below.

- **Number of scenario scripts:** Scenario scripts are a sequence of steps, which depict the interaction between the user and the application. A number of scenarios is directly related to application size and number of test cases that are developed to test the software, once it is developed. Note that scenario scripts are analogous to use-cases.
- **Number of key classes:** Key classes are independent components, which are defined in object -oriented analysis. As key classes form the core of the problem domain, they indicate the effort required to develop software and the amount of 'reuse' feature to be applied during the development process.
- **Number of support classes:** Classes, which are required to implement the system but are indirectly related to the problem domain, are known as support classes. For example, user interface classes and computation class are support classes. It is possible to develop a support class for each key class. Like key classes, support classes indicate the effort required to develop software and the amount of 'reuse' feature to be applied during the development process.

- **Average number of support classes per key class:** Key classes are defined early in the software project while support classes are defined throughout the project. The estimation process is simplified if the average number of support classes per key class is already known.
- **Number of subsystems:** A collection of classes that supports a function visible to the user is known as a subsystem. Identifying subsystems makes it easier to prepare a reasonable schedule in which work on subsystems is divided among project members.

Web engineering project metrics:

- Number of static web pages
- Number of static web pages
- Number of internal page links
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

Metrics for software quality:

Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.

In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. Set of activities in SAQ are continuously applied throughout the software process. Software Quality is measured based on some software quality

1.Code Quality – Code quality metrics measure the quality of code used for the software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development.

2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not is checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.

5. Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.

6. Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include time requires to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.

7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software's which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyber attacks.

8. Security – Security metrics measure how much secure the software is? In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

RISK MANAGEMENT

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Reactive vs proactive risk strategies:

Reactive risk management:

One fundamental point about reactive risk management is that the disaster or threat must occur before management responds. Proactive risk management is all about taking preventative measures before the event to decrease its severity, and that's a good thing to do.

At the same time, however, organizations should develop reactive risk management plans that can be deployed ***after*** the event. Otherwise management is making decisions about how to respond as the event happens, which can be a costly and stressful ordeal.

Proactive Risk Management

As the name suggests, proactive risk management means that you identify risks before they happen and figure out ways to avoid or alleviate the risk. It seeks to reduce the hazard's risk potential or, even better, prevent the threat altogether.

A good example here is vulnerability testing and remediation. Any organization of appreciable size is likely to have vulnerabilities in its software, which attackers could find an exploit. So regular testing (or, even better, continuous testing) can help to repair those vulnerabilities and eliminate that particular threat.

Software Risks

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control

something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. Known risks: Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)

2. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover)

3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks : generic risks and product-specific risks.

Generic risks are a potential threat to every software project. Product-specific risks can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the project at hand.

One method for identifying risks is to create a risk item checklist. The checklist can beused for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

• **Product size**—risks associated with the overall size of the software to be built or modified.

• **Business impact**—risks associated with constraints imposed by management or the marketplace.

• **Customer characteristics—**risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.

• **Process definition—**risks associated with the degree to which the software process has been defined and is followed by the development organization.

• **Development environment—**risks associated with the availability and quality of the tools to be used to build the product.

• **Technology to be built—**risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.

Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

Assessing Overall Project Risk

The following questions have derived from risk data obtained by surveying experienced software project managers in different part of the world. The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end-users enthusiastically committed to the project and the system/product to be built?
3. Are requirements fully understood by the software engineering team and their customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end-users have realistic expectations?
6. Is project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk Components and Drivers

software risk components—performance, cost, support, and schedule. In the context of this discussion, the risk components are defined in the following manner:

• **Performance risk**—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.

• **Cost risk**—the degree of uncertainty that the project budget will be maintained.

• **Support risk**—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.

• **Schedule risk**—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic.

Risk Projection

Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities:

(1) establish a scale that reflects the perceived likelihood of a risk,

(2) delineate the consequences of the risk,

(3) estimate the impact of the risk on the project and the product, and

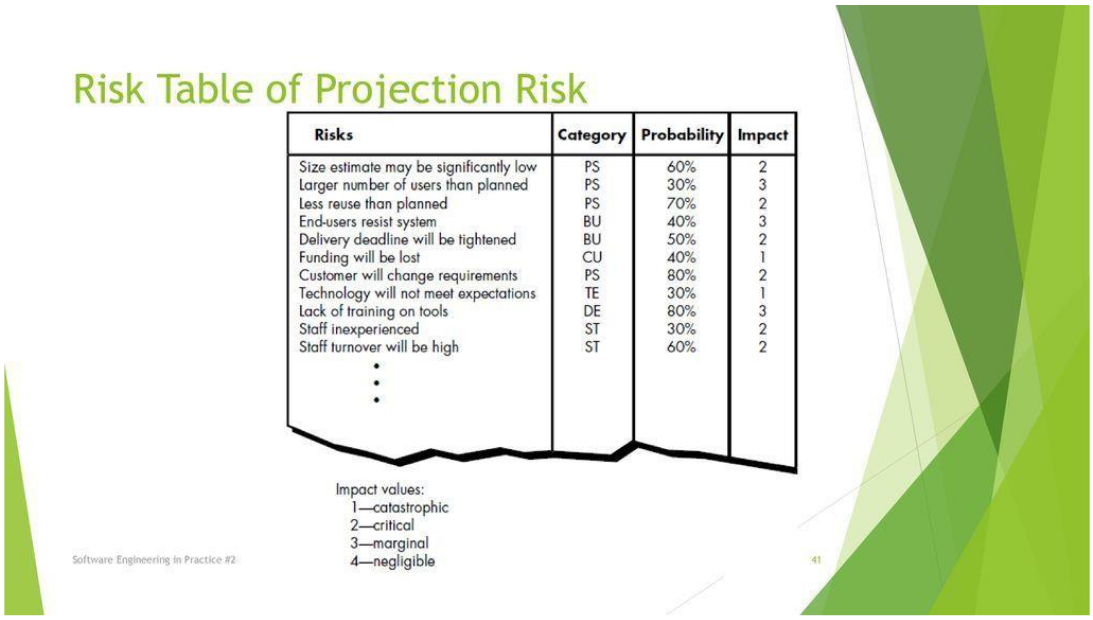
(4)note the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table

A risk table provides a project manager with a simple technique for risk projection .A project team begins by listing all risks (no matter how remote) in the first column of the table. This can be accomplished with the help of the risk item checklists Each risk is categorized in the second column (e.g., PS implies a project size risk, BU implies a business risk).

All risks that lie above the cutoff line must be managed. The column labeled RMMM contains a pointer into a Risk Mitigation, Monitoring and Management Plan or alternatively, a collection of risk information sheets developed for all risks that lie above the cutoff.

Risk probability can be determined by making individual estimates and then developing a single consensus value. Although that approach is workable, more sophisticated techniques for determining risk probability have been developed.



Assessing Risk Impact

Three factors affect the consequences that are likely if a risk does occur: its nature, its scope, and its timing. The nature of the risk indicates the problems that are likely if it occurs.

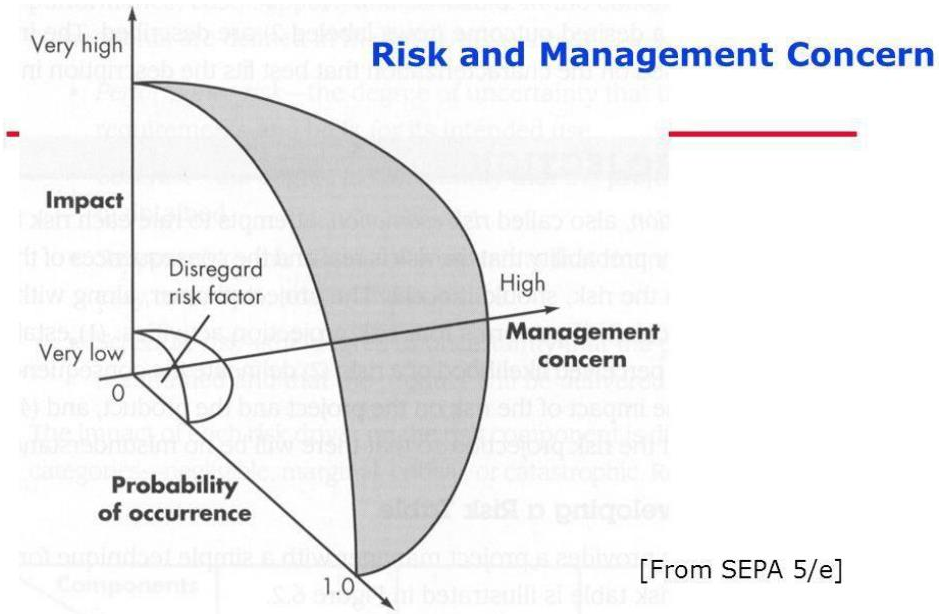
Returning once more to the risk analysis approach proposed by the U.S. Air Force , the following steps are recommended to determine the overall consequences of a risk:
1. Determine the average probability of occurrence value for each risk component.
2. Determine the impact for each component based on the criteria .
3. Complete the risk table and analyze the results as described in the preceding sections. The overall risk exposure, RE, is determined using the following relationship:

$$RE = P \times C$$

Risk identification. Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Risk probability. 80% (likely).
Risk impact. 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00, the overall cost (impact) to develop the components would be 18 x 100 x 14 = $25,200.

Risk exposure. RE = 0.80 x 25,200 ~ $20,200.



RISK REFINEMENT

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage.

One way to do this is to represent the risk in condition-transition-consequence (CTC) format . That is, the risk is stated in the following form: Given that <condition> then there is concern that (possibly) <consequence>.

Using the CTC format for the reuse risk noted in Section 6.4.2, we can write: Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined subconditions remains the same (i.e., 30 percent of software components must be customer engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

RISK MIGRATION, MONITORING, AND MANAGEMENT

Risk Migration:
It is an activity used to avoid problems (Risk Avoidance).
Steps for mitigating the risks as follows.

1. Finding out the risk.
2. Removing causes that are the reason for risk creation.
3. Controlling the corresponding documents from time to time.
4. Conducting timely reviews to speed up the work.

To mitigate this risk, project management must develop a strategy for reducing turnover. The possible steps to be taken are:

- Meet the current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner.
- Assign a backup staff member for every critical technologist.

Risk Monitoring :
It is an activity used for project tracking.
It has the following primary objectives as follows.

1. To check if predicted risks occur or not.
2. To ensure proper application of risk aversion steps defined for risk.
3. To collect data for future risk analysis.
4. To allocate what problems are caused by which risks throughout the project.

As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

- General attitude of team members based on project pressures.

- Interpersonal relationships among team members.
- Potential problems with compensation and benefits.
- The availability of jobs within the company and outside it.

Risk Management and planning :

- **Maintain a worldwide perspective:** view software risks within the context of a system and therefore the business drawback planned to solve.
- **Take an advanced view:** ink regarding the risk which can occur in the longer term and make future plans for managing the future events.
- **Encourage open communication:** Encourage all the stakeholders and users for suggesting risks at any time.
- **Integrate:** A thought of risk should be integrated into the software process.
- **Emphasize never-ending process:** Modify the known risk than a lot of info is understood and add new risks as higher insight is achieved.
- **Develop a shared product vision:** If all the stakeholders share a similar vision of the software then it's easier for better risk identification.
- **Encourage teamwork:** whereas conducting risk management activities pool the skills and knowledge of all stakeholders.

Drawbacks of RMMM:

- It incurs additional project costs.
- It takes additional time.
- For larger projects, implementing an RMMM may itself turn out to be another tedious project.
- RMMM does not guarantee a risk-free project, infact, risks may also come up after the project is delivered.

THE RMMM PLAN

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.

In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis. After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

QUALITY MANAGEMENT

QUALITY CONCEPTS:

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.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

User satisfaction= compliant product + good quality + delivery within budget and schedule

Quality Control

It can be compared to having a senior manager walk into a production department and pick a random car for an examination and test drive. Testing activities, in this case, refer to the process of checking every joint, every mechanism separately, as well as the whole product, whether manually or automatically, conducting crash tests, performance tests, and actual or simulated test drives.

Quality Assurance is a broad term, explained on "*the continuous and consistent improvement and maintenance of process that enables the QC job*". As follows from the definition, QA focuses more on organizational aspects of quality management, monitoring the consistency of the production process.

Cost of Quality :
It is the most established, effective measure of quantifying and calculating the business value of testing. There are four categories to measure cost of quality: Prevention costs, Detection costs, Internal failure costs, and External failure costs.
These are explained as follows below.

1. **Prevention costs** include cost of training developers on writing secure and easily maintainable code
2. **Detection costs** include the cost of creating test cases, setting up testing environments, revisiting testing requirements.
3. **Internal failure costs** include costs incurred in fixing defects just before delivery.
4. **External failure costs** include product support costs incurred by delivering poor quality software.

Major parts of total cost are detecting defects and internal failure cost. But, these costs less than external failure costs. That's why testing provides good business value.

Software Quality Assurance

Software Quality: Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

Quality Assurance focuses on how the engineering and management activity will be done?

As anyone is interested in the quality of the final product, it should be assured that we are building the right product.

It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

Software Quality Assurance

Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

SQA Activities

Software quality assurance is composed of a variety of functions associated with two different constituencies? the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

Following activities are performed by an independent SQA group:

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews

the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g. ISO-9001), and other parts of the software project plan.

3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.
5. **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project method, process description, applicable standards, or technical work products.
6. **Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

Software Reviews

Software Review

It is systematic inspection of a software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC).

Software review is an essential part of Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality and other vital features and components of the software. It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.

Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, test plans and test cases.

Objectives of Software Review:

The objective of software review is:

1. To improve the productivity of the development team.
2. To make the testing process time and cost effective.
3. To make the final software with fewer defects.
4. To eliminate the inadequacies.

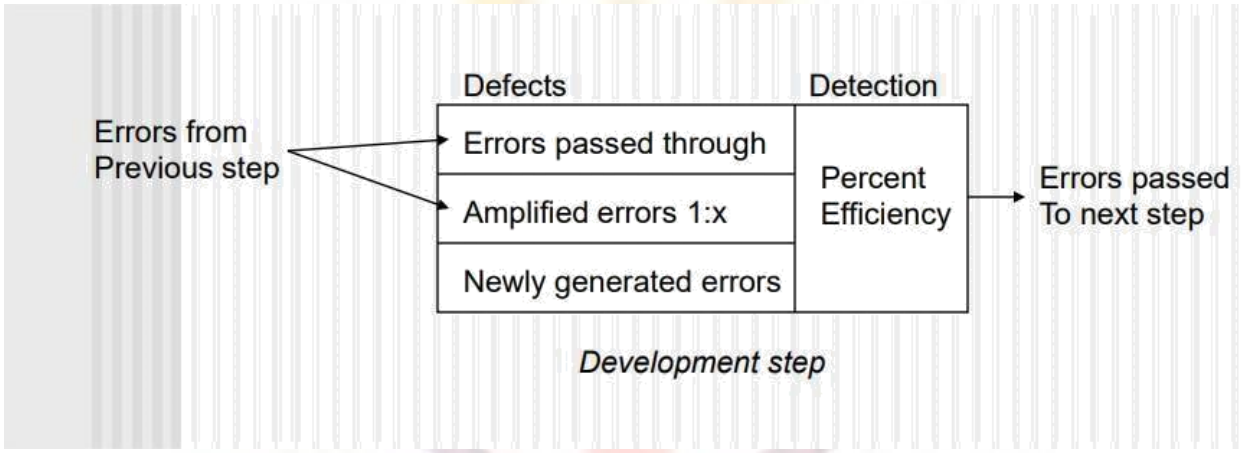
Cost impact of software defects

The primary objective of formal technical reviews is to find errors during the process so that they do not become defects after release of the software.

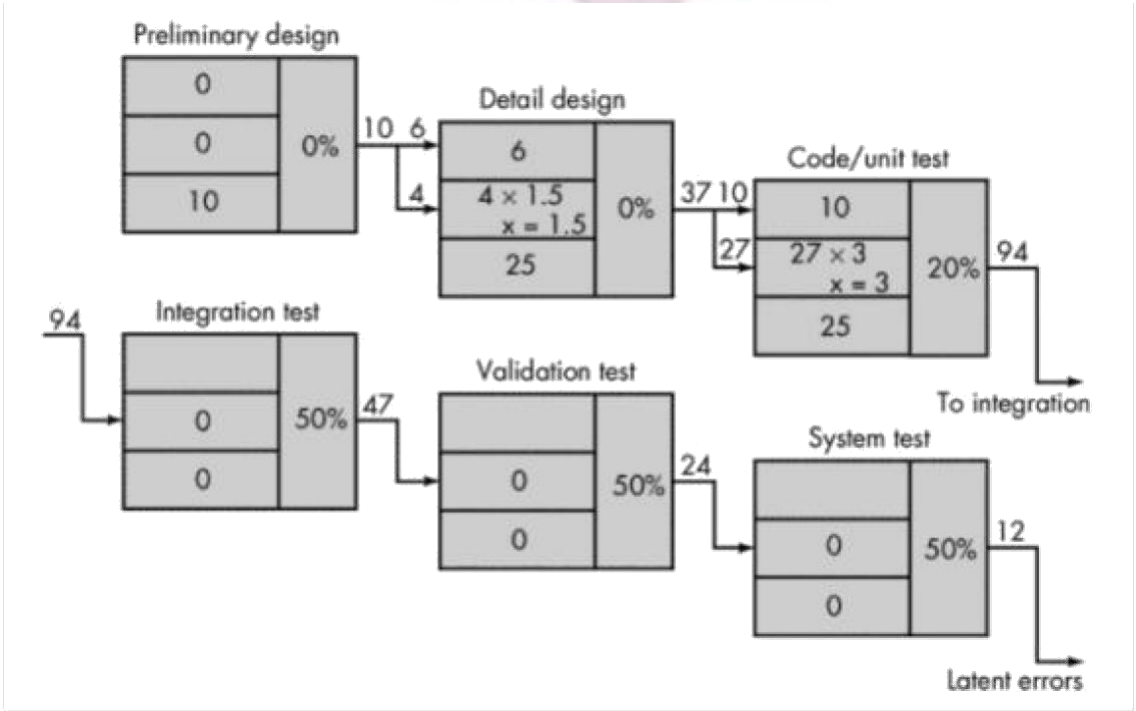
Defect amplification and removal

A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of a software engineering process.

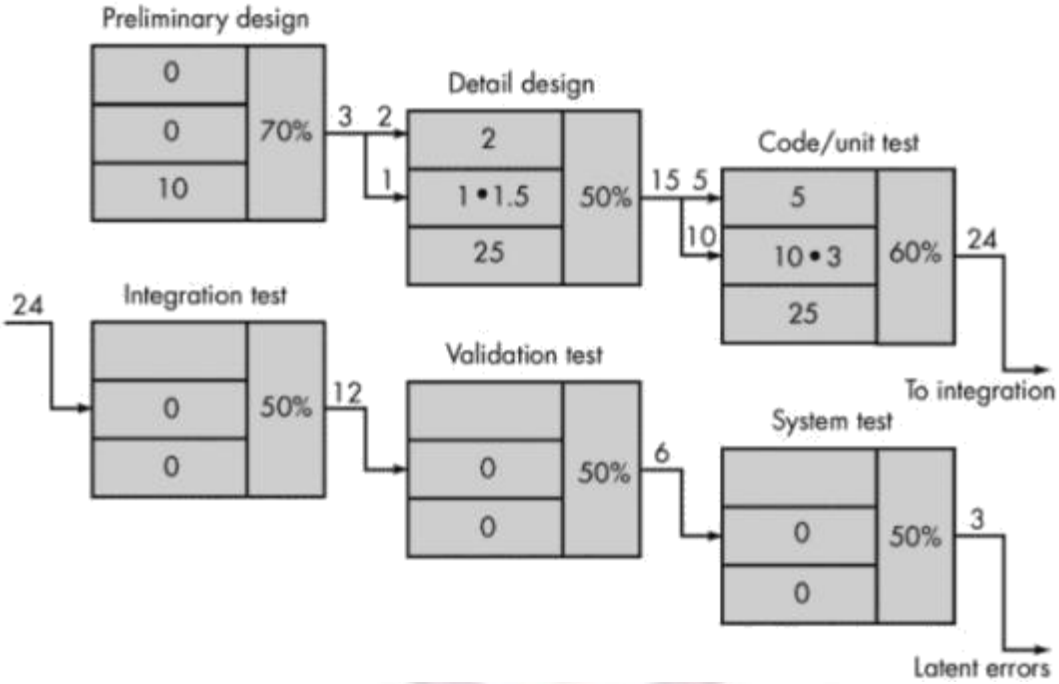
Defect amplification model:



Defect amplification – no reviews:



Defect amplification – reviews conducted:



Formal Technical Reviews

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 are 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.

Review reporting and record keeping :-

1. During the FTR, the reviewer actively records all issues that have been raised.
2. At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3. Finally, a formal technical review summary report is prepared.

It answers three questions :-

1. What was reviewed ?
2. Who reviewed it ?
3. What were the findings and conclusions ?

Review guidelines :-

Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

1. Review the product, not the manufacture (producer).
2. Take written notes (record purpose)
3. Limit the number of participants and insists upon advance preparation.
4. Develop a checklist for each product that is likely to be reviewed.
5. Allocate resources and time schedule for FTRs in order to maintain time schedule.
6. Conduct meaningful training for all reviewers in order to make reviews effective.
7. Reviews earlier reviews which serve as the base for the current review being conducted.
8. Set an agenda and maintain it.
9. Separate the problem areas, but do not attempt to solve every problem notes.
10. Limit debate and rebuttal.

Statistical software quality assurance

- Collect and categorize information (i.e., causes) about software defects that occur

- Attempt to trace each defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer)
- Using the Pareto principle (80% of defects can be traced to 20% of all causes), isolate the 20%

A generic example

- A sample of possible causes for defects:
- Incomplete or erroneous specifications
- Misinterpretation of customer communication
- Intentional deviation from specifications
- Violation of programming standards
- Errors in data representation
- Inconsistent component interface
- Errors in design logicϒ Incomplete or erroneous testing
- Inaccurate or incomplete documentation
- Errors in programming language translation of design
- Ambiguous or inconsistent human/computer interface

Six sigma

- Popularized by Motorola in the 1980s Is the most widely used strategy for statistical quality assurance
- Uses data and statistical analysis to measure and improve a company's operational performance Identifies and eliminates defects in manufacturing and servicerelated processes
- The "Six Sigma" refers to six standard deviations (3.4 defects per a million occurrences)

Three core steps

- Define customer requirements, deliverables, and project goals via well-defined
- methods of customer communication
- Measure the existing process and its output to determine current quality performance (collect defect metrics)
- Analyze defect metrics and determine the vital few causes (the 20%)
- Two additional steps are added for existing processes (and can be done inϒ parallel)
- Improve the process by eliminating the root causes of defects
- Control the process to ensure that future work does not reintroduce the causes of defects

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.

Measures of reliability and availability

Two meaningful metrics used in this evaluation are **Reliability** and **Availability**. Often mistakenly used interchangeably, both terms have different meanings, serve different purposes, and can incur different cost to maintain desired standards of service levels.

Availability refers to the percentage of time that the infrastructure, system, or solution remains operational under normal circumstances in order to serve its intended purpose. For cloud infrastructure solutions, availability relates to the time that the data center is accessible or delivers the intend IT service as a proportion of the duration for which the service is purchased. The mathematical formula for Availability is :

Percentage of availability = (total elapsed time – sum of downtime)/total elapsed time

Reliability refers to the probability that the system will meet certain performance standards in yielding correct output for a desired time duration.

Reliability can be used to understand how well the service will be available in context of different real-world conditions. For instance, a cloud solution may be available with an SLA commitment of 99.999 percent, but vulnerabilities to sophisticated cyber-attacks may cause IT outages beyond the control of the vendor. As a result, the service may be compromised for several days, thereby reducing the effective availability of the IT service.

MTBF = (total elapsed time – sum of downtime)/number of failures

Where MTBF means mean time between failures

Software safety:

As systems and products become more and more dependent on software components it is no longer realistic to develop a system safety program that does not include the software elements.

Does software fail? We tend to believe that well written, well tested, safety critical software never fails. Experience proves otherwise with software making headlines when it actually does fail, sometimes critically. Software does not fail the same way hardware does, and the various failure behaviors we are accustomed to from the world of hardware are often not applicable to

software. However, software does fail, and when it does, it can be just as catastrophic as hardware failures.

Safety-critical software

Safety-critical software is a creature very different from both non-critical software and safety-critical hardware. The difference lies in the massive testing program that such software undergoes.

The ISO 9000 QUALITY STANDARDS

A quality assurance system may be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications. These systems cover a wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process. ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.

The ISO 9000 standards have been adopted by many countries including all members of the European Community, Canada, Mexico, the United States, Australia, New Zealand, and the Pacific Rim. Countries in Latin and South America have also shown interest in the standards.

The ISO Approach to Quality Assurance Systems

The ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes. For a quality system to be ISO compliant, these processes must address the areas identified in the standard and must be documented and practiced as described.

The ISO 9001 Standard

ISO 9001 is the quality assurance standard that applies to software engineering. The standard contains 20 requirements that must be present for an effective quality assurance system. Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guidelines (ISO 9000-3) have been developed to help interpret the standard for use in the software process.

NRCM
your roots to success...