Unit - V

Software Maintenance :

Software maintenance is a post development phase of software development life cycle. Its main purpose is to modify and update software application after delivery to correct faults and to improves performance.
Software maintenance is a vast activity which includes optimization, error correction, deletion of discarded features and enhancements of existing features.

There are a number of reasons, why modifications are required, some of them are :

- Market conditions
- client requirements
- Host modifications
- Organization changes

Need for maintenance :

① Bug fixing - Bug fixing comes at priority to run the software seamlessly. This process contains search out for errors in code and correct them. This must be done without hurting rest of the functionalities of existing software.

② Capability Enhancement - This comprises improvement in features and functions to make solution compatible with varying market environment. It enhances software platforms, work pattern, hardware upgrade, compilers and all other aspects that affect system workflow.

③ Removal of outdated functions - The unwanted functionalities are useless. Moreover, by occupying space in solution, they hurt efficiency of the solution.

④ Performance Improvement - To improve system performance, developer detects issues through testing and resolve them. It prevents the solution from vulnerabilities.

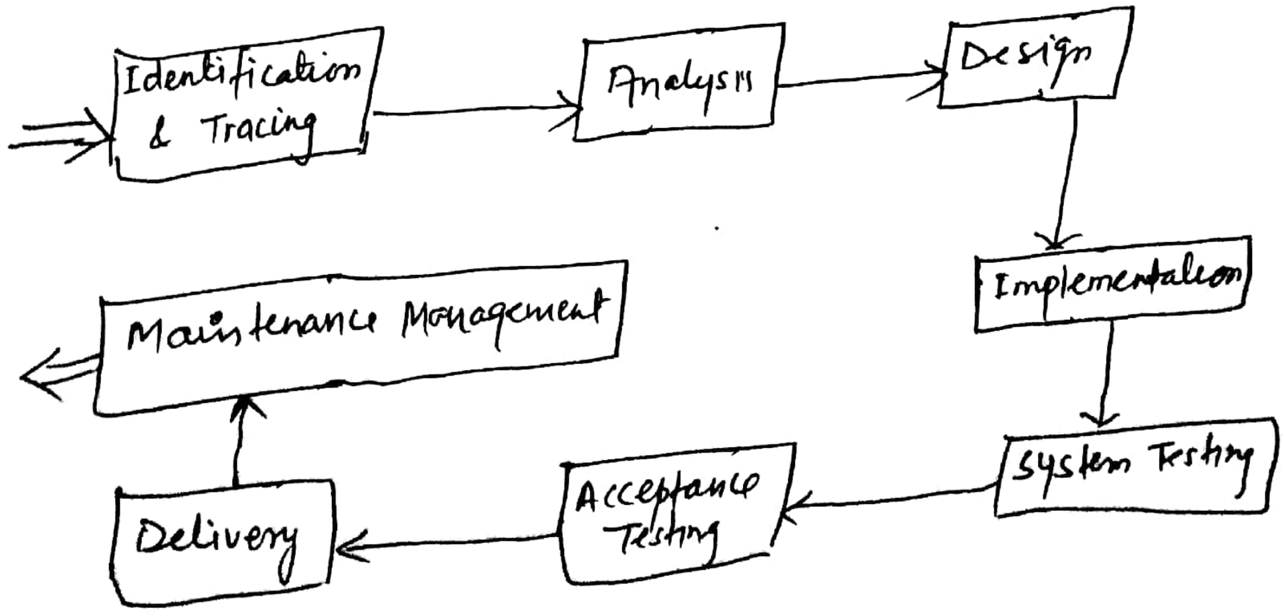
Categories of Maintenance

① Corrective Maintenance - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

② Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

③ Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improves its reliability and performance.

④ Preventive Maintenance - This includes modifications and updations to prevent future problems of the systems. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.



Maintenance Activities

① Identification and Tracing - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.

② Analysis - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe alternative solution is looked for. A set of required modifications is then materialized into requirement specifications.

③ Design - New modules, which need to be replaced or modified are designed against requirement specifications set in previous stage.

④ Implementation - The new modules are coded with the help of structured design created in design step. Every programmer is expected to do unit testing in parallel.

⑤ System Testing - Integration testing is done among newly created modules. Finally, the system is tested as a whole, following regressive testing procedure.

Acceptance Testing - After testing the system internally, it is tested for acceptance with the help of users.

Delivery - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system.

Cost of Maintenance :

On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

→ Real-world factors affecting maintenance cost - :

The standard age of any software is considered upto 10 to 15 years.
Most maintenance engineers are newbie and use trial and error method to ~~rectif~~ rectify problem.
Changes are left undocumented which may cause more conflicts in future.
Often changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
As technology advances, it becomes costly to maintain old software.

→ Software-end factors affecting Maintenance cost -

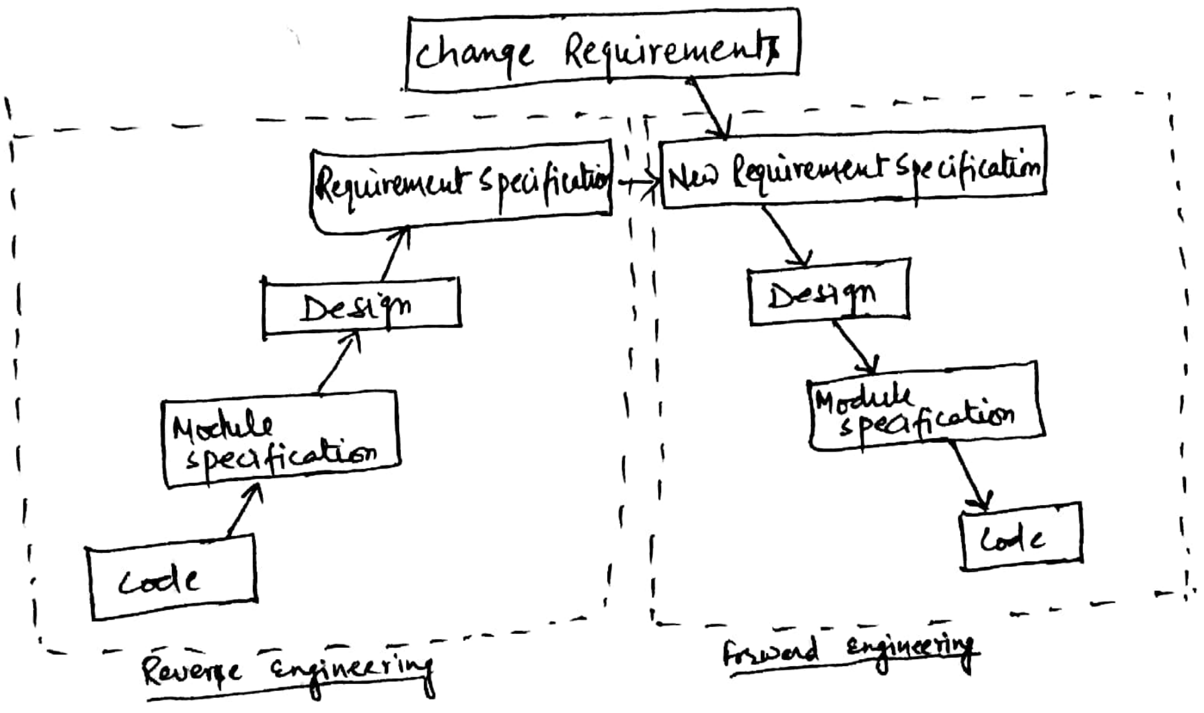
structure of software

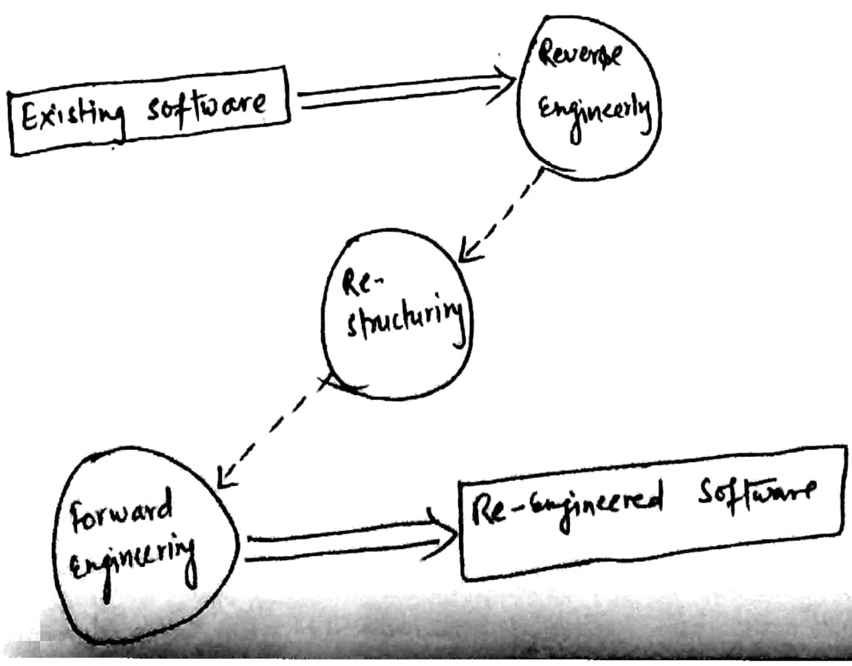
Programming language

Dependence on external environment

staff reliability and availability.

Software Re-engineering :





CASE Tools

Diagram Tools - flow chart maker

Process Modeling Tool - EPF composer

Project Management Tool - BaseCamp, Trac Project

Documentation Tool - Doxygen, DrExplain

Analysis Tool - CaseComplete, Accompa

Design Tool -

SCM Tool - - AccuREV, Fossil

Prog. Tools - Cscope, Eclipse

Prototyping Tool - Mockup builder

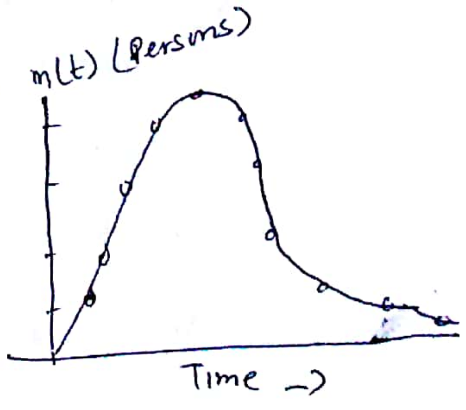
QA Tool - Appswatch, JMeter

Maintenance Tool - BugZilla

The Putnam Resource Allocation Model

The manpower curve rise, peak and then exponentially trail off as a function of time.

$$\boxed{m_0 = \frac{K}{t_d\sqrt{e}}}$$



Question: A software project is planned to cost 95 PY in a period of 1 year 9 months. calculate the peak manning and average rate of software team build up?

Solution: Software Project cost $K = 95$ PY
Peak development time $(t_d) = 1.75$ Years

① Peak manning $(m_0) = K / t_d\sqrt{e} = \frac{95}{1.75 \times 1.648} = 32.94$

② Average rate of software build up $= \frac{m_0}{t_d} = \frac{33}{1.75} = 18.8$ person/year

During the reverse engineering, the old code is analyzed (abstracted) to extract the module specifications. The module specifications are then analyzed to produce the design. The design is analyzed (abstracted) to produce the original requirements specification. The change request are then applied to this requirements specification to arrive at the new requirement specification.

At this point a forward engineering is carried out to produce the new code. At the design, module specification, and coding stages, a substantial reuse is made from the reverse engineered product.

When to re-engineer?

→ When the system changes are mostly confined to part of the system then re-engineer that part.

→ When hardware or software supports becomes obsolete.

→ When tools to support re-structuring are available.

Advantages of Re-engineering

① Reduced risk – There is a high risk in new software development. There may be development, staffing and specification problems.

② Reduced cost – the cost to reengineering is often significantly less than the cost of developing new software.

Re-Engineering Process -

Decide: what to re engineered? Is it whole software or part of it?

Perform: Reverse engineering, in order to obtain specifications of existing software

Restructure Program: If required. For example function-oriented program to object oriented.

~~Restructured data:~~

Re-structure data: as required.

Apply forward engineering: concepts in order to get re-engineered software.

Reverse Engineering -

It is a process of analyzing software with a view to understanding its design and specification

Re-engineering -

It is re organizing and modifying existing system to make them more maintainable.

Forward Engineering -

It is the traditional process of moving from high level abstractions and logical designs to the physical implementation of a system.

$$\begin{tikzcd}\text{concept} \arrow[r] & \boxed{\begin{matrix}\text{Engineering}\\ \text{a Product}\end{matrix}} \arrow[r] & \text{Product}\end{tikzcd}$$

<u>forward engineering</u>

$$\begin{tikzcd}\text{Product} \arrow[r] & \boxed{\begin{matrix}\text{Reverse}\\ \text{engineering}\end{matrix}} \arrow[r] & \text{concept}\end{tikzcd}$$

<u>Reverse Engineering</u>

Reverse engineering can be mainly viewed as the process of analyzing a software to identify its components and their interrelationships, to create representations of it in another form or a higher level of abstraction

<u>Need of Reverse engineering</u>

-> Recovery of lost information
-> Providing proper system documentation
-> Assisting with maintenance
-> facility of software reuse
-> Discovering unexpected flaws and faults.

Software Configuration Management (SCM)

Software configuration management, also called change management, is a set of activities designed to manage change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed and auditing and reporting on the changes made.

SCM can be considered as having three major components:

i) software configuration item identification
ii) change control
iii) status accounting & auditing

1) Configuration Identification: When a change is done, it should be clear to what changes has been applied. This requires baselines to be established. A baseline change is the changing of the established baseline, which is controlled by SCM. Some of the common baselines are functional or requirement baseline, design baseline and system baseline. After baseline changes the state of the software is defined by the most recent baseline and the changes that were made.

ii) Change Control: Change is initiated by change request (CR). The CR for change generally consists of 3 parts.

The first part describes the change, reason for change, the SCIs that are affected, the priority of the change etc.
The second part describes the decision taken by the change control board (CCB) on this CR, the action the change Manager(CM) feels need to be done to implement this change and any other comments.
The third part is filled by implementer, which later implements the change.

III) <u>status accounting and auditing</u> - The current status which could be active, completed, or not scheduled is specified. A summary is prepared to track all the changes.

<u>Benefits</u>

① with the help of SCM we can easily control all changes which are done in development process.

② It gives the surety to check that changes are done on required area.

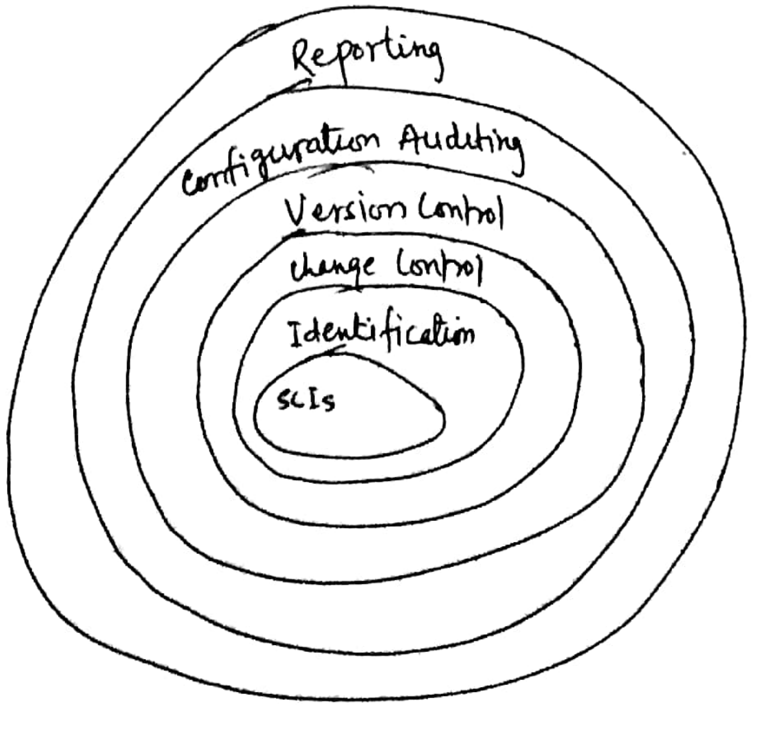
③ It is helpful to generate the new software with old components.

SCM process objective

1. Identify all items that define the software configuration.
2. Manage changes to one or more configuration items.
3. Facilitate construction of different versions of a software application.
4. Ensure that software quality is maintained as configuration evolves.

SCM tasks

1. Identification (tracking multiple versions to enable efficient changes)
2. version control (control changes before & after release to customer)
3. change control (Authority to approve and prioritize changes)
4. configuration auditing (ensure changes made properly)
5. Reporting (let others know about changes made)



Change Control & Process :

Need for change is recognized

↓

Change request from user

↓

Developer Evaluates

↓

Change report is generated

↓

Change control authority decides

↙ ↘

Request is queued for action | Change request is denied

↓ | ↓

Assign individuals for configuration objects | User is informed

↓

"Check-out" configuration objects

↓

Make the change

↓

Review (audit) the change

↓

"Check-in" the configuration items that have been changed

↓

Establish a baseline for testing

↓

Perform quality assurance and testing activities

↓

"Promote" changes for inclusion in next release

↓

Rebuild appropriate version of software

↓

Review (audit) the change to all configuration items

↓

Include changes in new version

↓

Distribute the new version

Software Risk Management

Risk is a problem that could cause some loss or threaten the success of the project, but which has not happened yet.
These potential problem might have an adverse impact on cost, schedule or technical success of the project, quality of software products.
Risk management is the process of identifying, addressing and eliminating these problems before they can damage the project.

Typical Software Risks:

① Dependencies - Many risks arise due to dependencies of project on outside agencies or factors.

② Requirement Issues - Many projects face uncertainty and turmoil around the products requirement.
Threat to success increases if issues are not resolved as the project progresses.

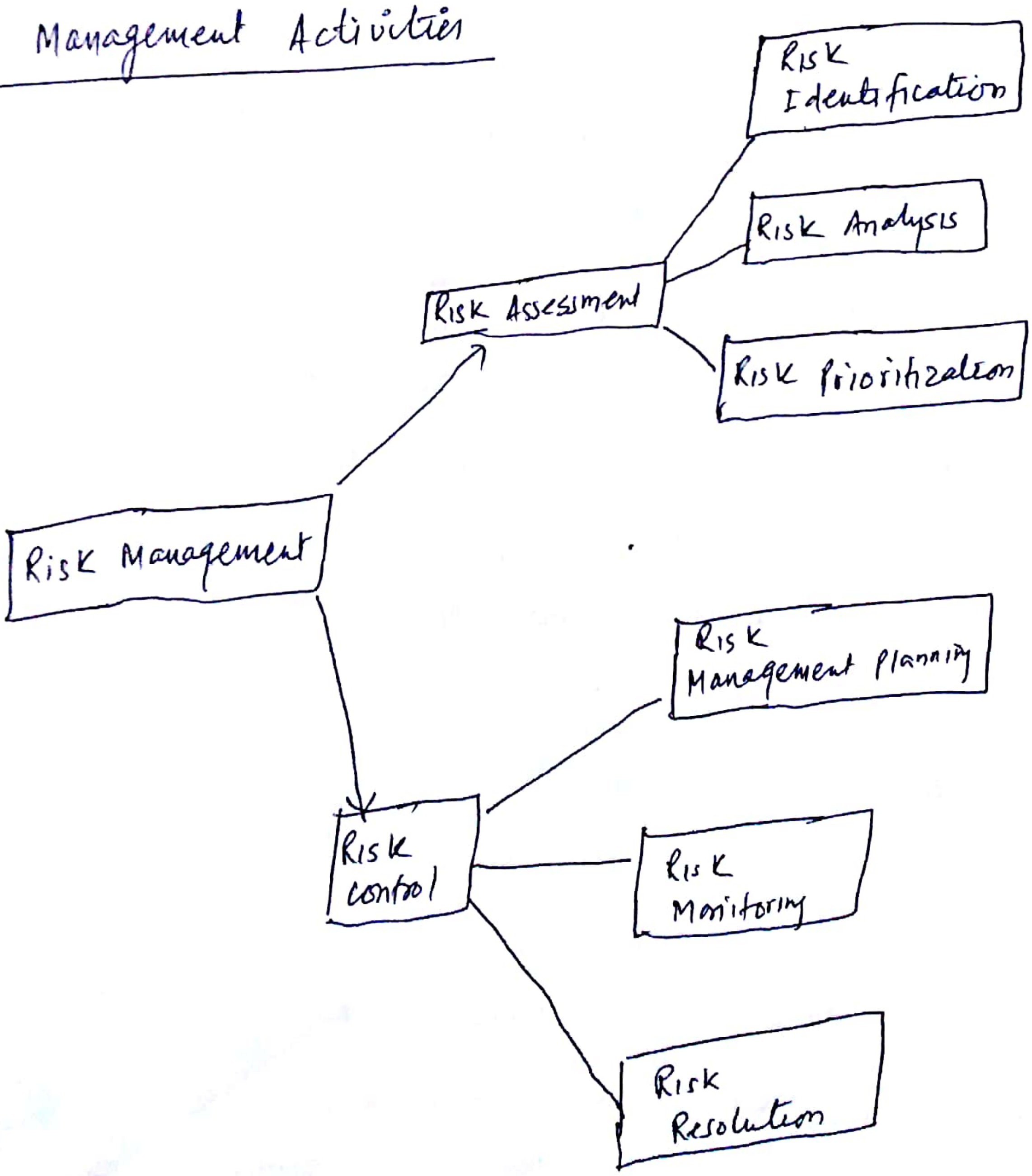
③ Management Issues - Defined project tracking processes and clear roles and responsibilities, can address some of these risk factors:

→ Inadequate planning and task identification.
→ Inadequate visibility into actual project status.
→ Unclear project ownership and decision making.
→ Unrealistic commitments made
→ Manager or customer with unrealistic expectation
→ Staff personality conflicts
→ poor communication

(4) Lack of Knowledge - The rapid rate of change of technologies and the increasing change of skilled staff mean that our project teams may not have the skills we need to be successful.

→ Inadequate training
→ Poor understanding of methods, tools and technologies
→ Inadequate application domain experience
→ Ineffective, poorly documented or neglected processes

Risk Management Activities



Version Control - Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process. A version control system implements or directly integrated with four major capabilities

i) Project database (repository) that stores all relevant ~~information~~ configuration objects.

ii) A version management capability that stores all versions of a configuration objects.

iii) A make facility that enables you to collect all relevant configuration objects and construct a specific version of the software.

Change set - captures all changes to all files in the configuration along with the reason for changes and details of who made the changes and when.

Version control system establishes a change set.

Software Reliability

Reliability of software products essentially denotes its trustworthiness or dependability. Alternatively, reliability of a software product can also be defined as the probability of the product working correctly over a given period of time.

Reliability Metrics -

① **Rate of occurrence of Failure (ROCOF):**

ROCOF measures the frequency of occurrence of unexpected behavior (i.e. failure).

② **Mean Time to failure (MTTF):** MTTF is the average time between two successive failures, observed over a large number of failures.

let failure occur at the time instants $t_1, t_2, \ldots t_n$.

$$MTTF = \sum_{i=1}^{n} \frac{t_{i+1} - t_i}{(n-1)}$$

③ **Mean Time to Repair (MTTR):** MTTR measures the average time it takes to track the errors causing the failure and then to fix them.

④ **Mean Time Between Failure (MTBF):**

$$MTBF = MTTF + MTTR$$

⑤ **Probability of failure on Demand (POFOD):** POFOD measures the likelihood of the system failing when a service request is made.

Scanned by CamScanner

Software Reliability Models

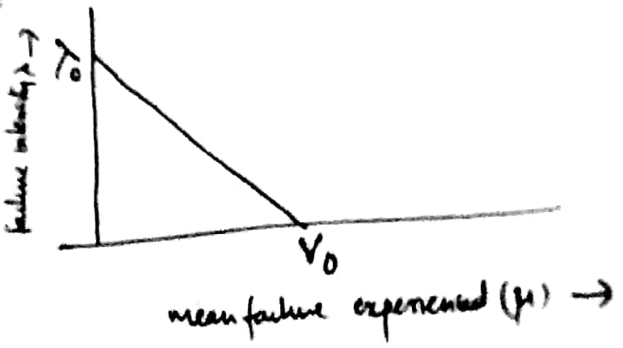
① Basic Execution Time Model

$$\boxed{\lambda(\mu) = \lambda_0 \left(1 - \frac{\mu}{V_0}\right)}$$

λ_0 : Initial failure intensity at the start of execution

V_0 : No of failure experienced, if program is executed for infinite time period

μ : Average or expected no of failure experienced at a given point in time



Decrement of failure intensity $\frac{d\lambda}{d\mu} = \frac{-\lambda_0}{V_0}$

Question: Assume that a program will experience 200 failures in infinite time. It has now experienced 100. The initial failure intensity was 20 failure/CPU hr. ① Determine current failure intensity. ② Find the decrement of failure intensity per failure.

Solution: $V_0 = 200$ failures $\quad \mu = 100$ failure $\quad \lambda_0 = 20$ failure/CPU hr

① Current failure intensity $\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{V_0}\right)$

$$= 20\left(1 - \frac{100}{200}\right) = 10 \text{ failure/CPU hr}$$

② Decrement of failure intensity per failure

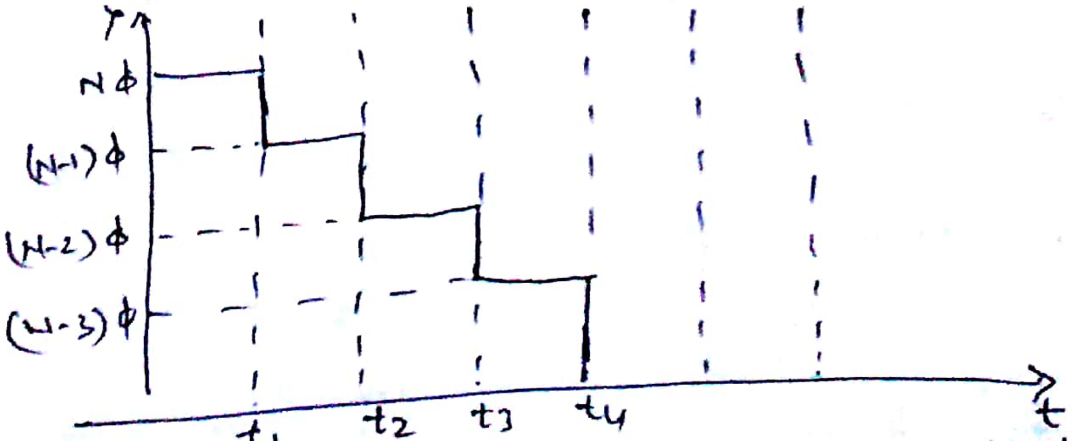
$$\frac{d\lambda}{d\mu} = \frac{-\lambda_0}{V_0} = \frac{-20}{200} = -0.1 \text{ CPU/hr}$$

(2) The Jelinski-Moranda Model : It is the earliest and probably the best known reliability model. It proposed failure intensity function in the form of

$$\boxed{\lambda(t) = \phi(N - i + 1)}$$

where ϕ = constant of proportionality
N = Total no of errors present
i = no of errors found by time interval t_i

This model assumes that all the failures have the same failure rate. It means that failure rate is a step function and there will be an improvement in reliability after fixing a fault.



Qn: There are 100 errors estimated to be present in a program. 60 of them have been experienced. Calculate failure intensity with a given value of $\phi = 0.03$. What will be failure intensity after 80 errors experience?

Sol^n $N = 100$, $i = 60$, $\phi = 0.03$

$\lambda(t) = \phi(N - i + 1)$
$= 1.23$ failures / CPU Hr.

after 80 failures
$\lambda(t) = 0.03(100 - 80 + 1) = 0.63$ failure / CPU Hr.

Constructive Cost Model (COCOMO)

COCOMO is a hierarchy of software cost estimation models, which includes basic, intermediate and detailed models.

① Basic Model - Basic model aims at estimating, in a quick and rough fashion, most of the small to medium sized software projects. Three modes of software development are considered in this model:
organic, Semi-detached and Embedded.

In the organic mode, a small team of experienced developers develop software in a very familiar environment. The size of the software development in this mode ranges from small (a few KLOC) to medium (a few tens of KLOC).

In Embedded mode of software development, the project has tight constraints.

The semi detached mode is an intermediate mode between the organic mode and embedded mode.

$$\boxed{\text{Effort } (E) = a_b(\text{KLOC})^{b_b}}$$ unit person-month

$$\boxed{\text{Development time } (D) = C_b(E)^{d_b}}$$ unit in months

$$\boxed{\begin{array}{l}\text{Average staff size } (SS) = \dfrac{E}{D} \\ \text{Productivity } (P) = \dfrac{\text{KLOC}}{E}\end{array}}$$

unit is person

unit is KLOC/PM

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	.38
Semi detached	3.0	1.12	2.5	.35
Embedded	3.6	1.20	2.5	.32

Basic

(2) Intermediate Model :

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	.38
Semi detached	3.0	1.12	2.5	.35
Embedded	2.8	1.20	2.5	.32

Intermediate

$$\boxed{E = a_i (KLOC)^{b_i} \times EAF}$$

$$\boxed{D = c_i (E)^{d_i}}$$

Effective Adjustment factor (EAF)