SESSION-12
SOFTWARE RELIABILITY AND MAINTENANCE

DR. BHARATI WUKKADADA

- reliability
- Reliability Metrics
- Reliability growth modelling
- Software reverse engineering
- Software maintenance costs
- Estimation of maintenance costs
- Video
- Bing Videos
- Bing Videos
- Bing Videos

BING VIDEOS

Trustworthiness or dependability

Reliability

Reliability is *the probability that a product will perform successfully under specified operating conditions for a given period of time.*

Reliability is Quality over Time

Reliability Indices

Failure Rate (λ)- A Reliability index that represents the **rate at which your product fails.**





Mean Time To Failure (MTTF) – The reliability index for ***non-repairable units*** represents the *mean time to failure.*

Mean Time Between Failure (MTBF) – The reliability index for ***repairable units*** represents the *mean time between failure.*

Failure Rate (λ)- A Reliability index that represents the **rate at which your product fails.**

$$Failure\ Rate\ (\lambda) = \frac{Number\ of\ Failures}{Operating\ Time\ (Cycles)} = Failures\ Per\ Hour$$

Mean Time To Failure (MTTF) – The reliability index for ***non-repairable units*** represents the *mean time to failure.*

Mean Time Between Failure (MTBF) – The reliability index for ***repairable units*** represents the *mean time between failure.*

$$MTTF = \theta = \frac{Operating\ Time\ (Cycle)}{Number\ of\ Failures}$$

$$MTBF = \theta = \frac{Operating\ Time\ (Cycle)}{Number\ of\ Failures}$$

MTTF / MTBF

MTTF and MTBF are reflections of the **reliability** of your product- **θ** (Theta).

$$\textit{Failure Rate}\ (\lambda) = \frac{\textit{Number of Failures}}{\textit{Operating Time}} \qquad \textbf{MTTF \& MTBF} = \theta = \frac{\textit{Operating Time}}{\textit{Number of Failures}}$$

XAMPLE: 20 units are put on test and run at their normal operating condition for **1,000 hours**.

If **6** of those units fail, at the following hours (**550, 480, 680, 790, 860, 620**),

what is the **mean time to failure of the product?**

$$\theta = \frac{550 + 480 + 680 + 790 + 860 + 620 + (14 * 1{,}000)}{6} = \frac{17{,}980}{6} = \textbf{2,996 Hours to Failure}$$

✅ **Step 1: Sum of failed units' operating hours:**

$$550 + 480 + 680 + 790 + 860 + 620 = 3,980$$

✅ **Step 2: Add operating hours of 14 units that did not fail:**

$$14 \times 1,000 = 14,000$$

✅ **Step 3: Total Operating Time:**

$$3,980 + 14,000 = 17,980$$

✅ **Step 4: Divide by number of failures (6):**

$$\theta = \frac{17,980}{6} = 2,996.67 \text{ hours to failure (MTTF)}$$

Even though only 6 units failed, we include the full operational time of non-failed units to get a better average. That's why **MTTF = 2,996 hours** — a strong reliability indicator.[units 20* hrs 1000, so total 20000, in that only 29996.67 failures]

RELIABILITY

- Reliability of a software specifies probability of failure free operation for a given time duration with respective to a software
- It is a dynamic characteristic / attribute in nature where number of failures it encounter
- Reliability must be

  stable, consistent user experience

system breakdown,dataloss, downtime,inaccurate , delays should not happen
calulate FR-failure rate

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

RELIABILITY : Introduction

Reliability of a Software specifies probability of failure free operation for a given time duration.

* It is a dynamic system characteristic
  - presence of faults
  - ↳ a function of number of SW failures.

It is an execution event where the SW behaves unexpectedly.

Not all SW faults have equal probability of manifestation / execution.

↳ Removing SW faults from those SW parts which are rarely used makes little improvement in Reliability.

Operational Profile of SW : the way in which SW is used by the users
- The kind of I/P that are supplied to the SW.

Failure of a SW depends on 2 factors :
1. No. of faults being evaluated in SW.
2. Operational profile of execution of SW.

Types of Time

1. Execution Time : The actual CPU time that the SW takes during its execution
2. Calendar Time → Normal/Regular time that we use on a daily basis.
3. Clock Time : Actual time that is elapsed while the SW is executing
   ↓
   including the SW time that the SW spends while waiting in system.

SOFTWARE RELIABILITY

- Reliability of S/w products denotes it's **trustworthiness** or **dependability**.
- It is defined as : ***"probability of the product working correctly over a given period of time."***
- **Reliability of the system improves, if the number of defects in it are reduced**

1. the number of **latent defects** in the system.

2. **the perceived reliability** of the product

3. **Reliability of the product** also **depends on the exact location of the errors.**

Latent Defect is a popular term in the dictionary of **software testing**. This is a **defect** that is not known to the customer unless he faces an unforeseen situation but at the same time the developer or the seller is aware of the **defect**. ... **Latent defects** can be identified effectively with inspections. Sep 11, 2014

Can a poor user interface influence **reliability**? Yes. If the user is unable to operate a service, it may be **perceived** as poor **reliability**

- The **operational profile** is a quantitative characterization of how a system will be used that shows how to increase productivity and reliability and speed development by allocating development resources to function on the basis of use.

1 LATENT DEFECT

- Testers, while ensuring the quality and performance of the product come across various defects. Some are easy to identify while others are masked or hidden in the product and require intensive measures to be uncovered. Among these defects, there is an unusual one, that remains hidden until the circumstances reveal it and it becomes a hindrance in the on-going process. This type of defect is known as Latent Defect and is commonly observed in software testing. So, let us understand what latent defect connotes to and the impact it has on the software product, before as well as after the delivery.
- What is Latent Defect in Software Testing?
- One of the unusual defects found during software testing, Latent Defect is an error that hasn't occurred yet, and that can lead to failure whenever an accurate set of conditions are not fulfilled. It is a systematic flaw that accompanies the software during the production process and even passes the initial testing, including the pre-production testing and extended use.
- Present in the system for a long time, latent defect is uncovered by the users in the real-world environment, when a particular task is performed in the absence of regular scenarios or due to an unusual or rare set of conditions.
- Latent Defect Example
- February has 28 days, except in a leap year. The system might is not able to consider the leap year or an extra day in February, which results in a latent defect.

2, PERCEIVED RELIABILTY

Frequently asked questions

1. Which three parts does user perceived reliability contribute to?

- A user interface that is easy to understand and operate,
- Lack of technical errors,
- No installation problems.

•Can a poor user interface influence reliability?

2. Can a poor user interface influence reliability?

Yes. If the user is unable to operate a service, it may be perceived as poor reliability though in the sense of technical reliability will probably not.

3. How important is user perceived reliability for real-time person-to-person communication??

How important is user perceived reliability for real-time person-to-person communication??

It is important -- or very important. If the users cannot operate a system, they will not use it. For Real-time systems with person-to-person communication, several results show it is essential.

4. How can user perceived reliability be improved?

How can user perceived reliability be improved?

it is here suggested 3 ways:

- Enhance user interface so it is easy to understand and operate – choose a system that can be operated by an end-user, not only demonstrated. The user need to be able to use the service anytime, and that is when the service is needed.
- Enhance technical reliability – this is mainly the service provider's responsibility and can be revealed by talking to other users. If their attitude to technical errors is not acceptable, do something with it. The experience is that this will not solve by passive waiting.
- Be careful with installations – this is about installing, upgrading and remaining the installation. To install for first time may result in new demands: Am I able of getting a public, fixed IP- address with my subscription and am I able to traverse through the firewall? Upgrading the installation: This can lead to be asked question the installer does not know answer to (opening certain Ports, identify microphone driver or license key received some time earlier. Remaining the installation: This means that equipment used for the communication service (e. g. microphone, camera) has been borrowed for other purposes and need to be found and reinstalled.

- It also depends upon how the product is used i.e. on it's **execution profile.**

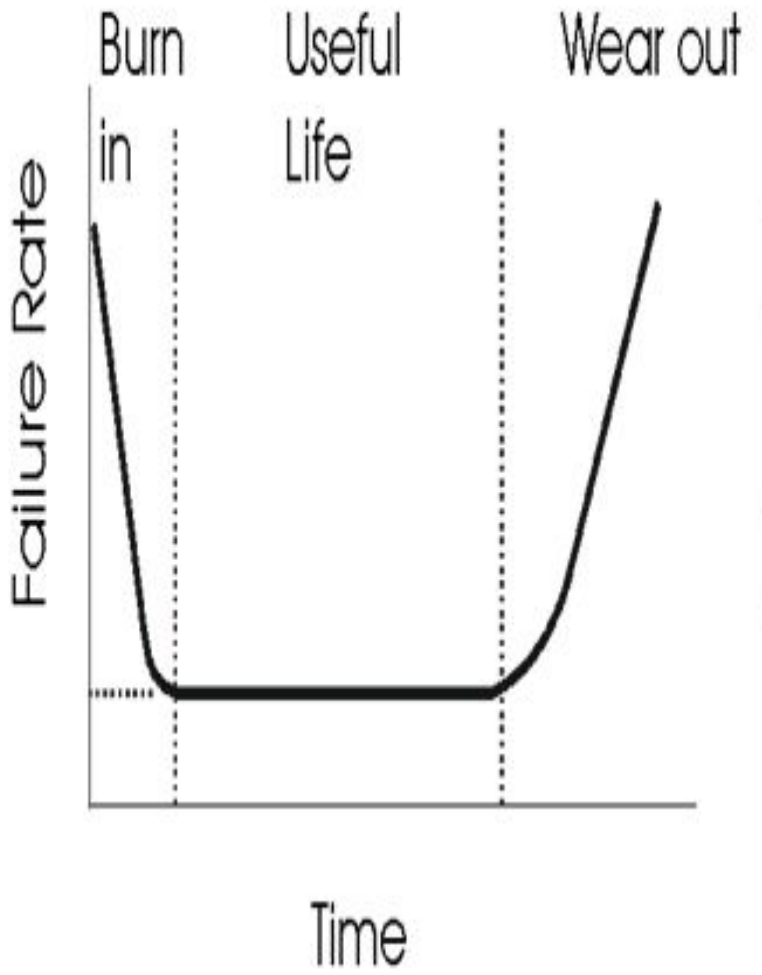
  - Executing only **correctly implemented functions reliability of the product will be high**.

  - **If we select i/p data such that only those function which contain errors are invoked, the reliability of the system will be very low**

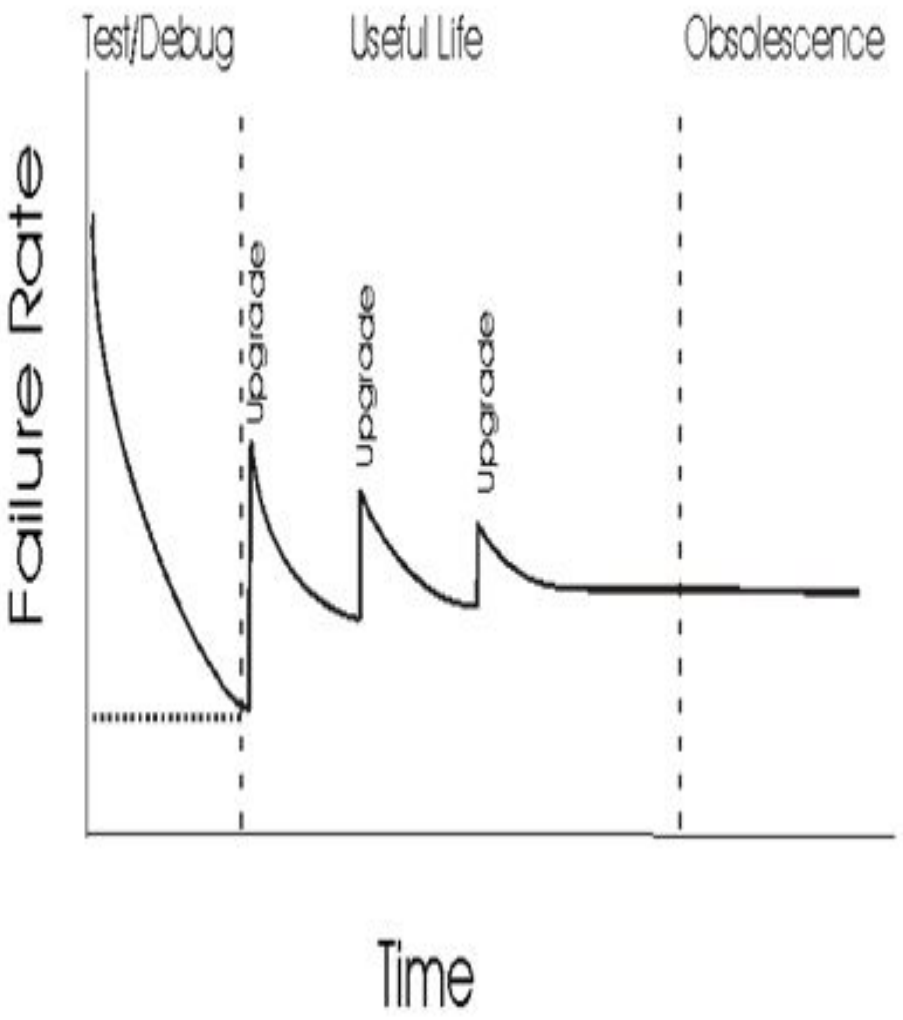
- the reliability of s/w product is **observer-dependent** & can not be determined absolutely.
  - E.g. Library Automation s/w
    - **Functions that library members use** (issue book, search book) are error free
    - **Function that librarian use** (create member, delete member) have many bugs.

WHY SOFTWARE RELIABILITY IS DIFFICULT TO MEASURE

- **Reliability improvement due to fixing a single bug depends on where the bug is located**
- **Perceived reliability** of a software product is **highly observer- dependent**
- **Reliability keep changing as the errors are detected and fixed**



Hardware Product



Software Product

RELIABILITY METRICS

Reliability metrics are measurable indicators that assess the stability, dependability, and overall quality of a software system. Common reliability metrics include:

1. **Uptime**: The amount of time a system is available for use. gremlin.com
2. **Service Level Agreements (SLAs)**: Agreements that define the expected level of service reliability.
3. **Mean Time Between Failures (MTBF)**: The average time between system failures.
4. **Mean Time to Resolution (MTTR)**: The average time taken to resolve system failures.

RELIABILITY METRICS

- **Reliability metrics** are used to quantitatively expressed the **reliability** of the **software** product. The option of which **metric** is to be used depends upon the type of system to which it applies & the requirements of the application domain.
- According to ANSI, "**Software Reliability** is defined as the probability of failure-free **software** operation for a specified period of time in a specified environment". ...
- **Metrics** used early can aid in detection and correction of requirement faults that will lead to prevention of errors later in the **software** life cycle.

RELIABILITY METRICS

- Metrics or techniques used **to estimate the quantitative reliability** of s/w product are called reliability metrics.
- By using this we can specify **the level of reliability required for a product can be specified in the SRS**
- **A good reliability metric should be observer-independent**, so that different people can agree on degree of reliability that the system has.

Reliability Metrics

1. Probability of failure on demand (POFOD)

- It is a measure of the likelihood that the system will behave in an unexpected way when some demand is made on it

eg: Safety-critical system.

2. Rate Of Occurrence Of Failure (ROCOF)

A measure of frequency of occurence with which unexpected behaviour is likely to be observed.

eg ROCOF = $\frac{9}{100}$ → SW will fail 9 times out of 100 operational unit times.

3. Mean Time To Failure (MTTF)

- A measure of time interval between observed failures.
- Useful when system is stable and no changes are being made to it.

↳ Indication of how long the system will be operational before failure occurence.

4. Availability : Measure of how likely the system is to be available for use.

Availability = $\frac{10}{100}$

MTBF = MTTF + MTTR

- MTBF → Mean Time Between Failure
- MTTF → Mean Time To Failure.
- MTTR → Mean Time To Repair

Availability = $\left(\frac{MTTF}{MTTF + MTTR}\right) \times 100\%$

2 formulas

1 Probability of Failure on Demand (POFOD)

- It measure **the likelihood of the system failing when a service request is made.**
- **Considering here is “chances”**
- **Eg: safety Critical system**

- E.g.- a POFOD of 0.001 means that 1 out of every 1000 service request would result in failure.

2, Rate of Occurrence of Failure (ROCOF)

- It measure the frequency of occurrence of unexpected behavior (i.e. failure)
- It happens frequently
- Obtained by observing the behavior of s/w product in operation over a specified time interval &
- then calculating the total number of failures during this interval.

3. Mean Time to Failure (MTTF)

- **A measure of time interval between observed failures.**
- **Useful when system is stable and no changes are bring made to it [indication of how long the system will be operational before failure occurance**
- **It is the average time between two successive failures, observed over a large number of failures.[find out time of interval between 2 failures]**
- **Let the failure occur at the time instants $t_1, t_2, \ldots, t_n$.**

Then MTTF can be calculated as

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

- **Only run-time is considered in the time measurements (Boot time etc. is not considered)**

4. Mean Time to Repair (MTTR)

- **Once failure occurs, some time is required to fix the error.**
- MTTR **measures the average time it takes to track the errors** causing the failure & **then to fix them**.

- mean time to repair

4. Mean Time Between Failures (MTBF)

MTBF = MTTF + MTTR

- E.g. – if MTBF of 300 hrs. indicate that once a failure occurs, the next failure is expected to occur only after 300 hrs.

$$\text{Availability} = \left(\frac{MTTF}{MTTF + MTTR} \right) \times 100\%$$

6. Availability

- It is the **likelihood of the system made available for use over a given period of time**.
- This metric considers:
  1. **The no. of failures occurring during a time interval &**
  2. **The repair time of the system when the failure occurs**.
- This metric is used for systems like **telecommunication** or **operating system**, which are never supposed to be down

AVAILABILITY MANAGEMENT

Objective: *ITIL Availability Management* aims **to define, analyze, plan, measure and improve** all aspects of the availability of IT services.

Part of: Service Design

Process Owner: Availability Manager

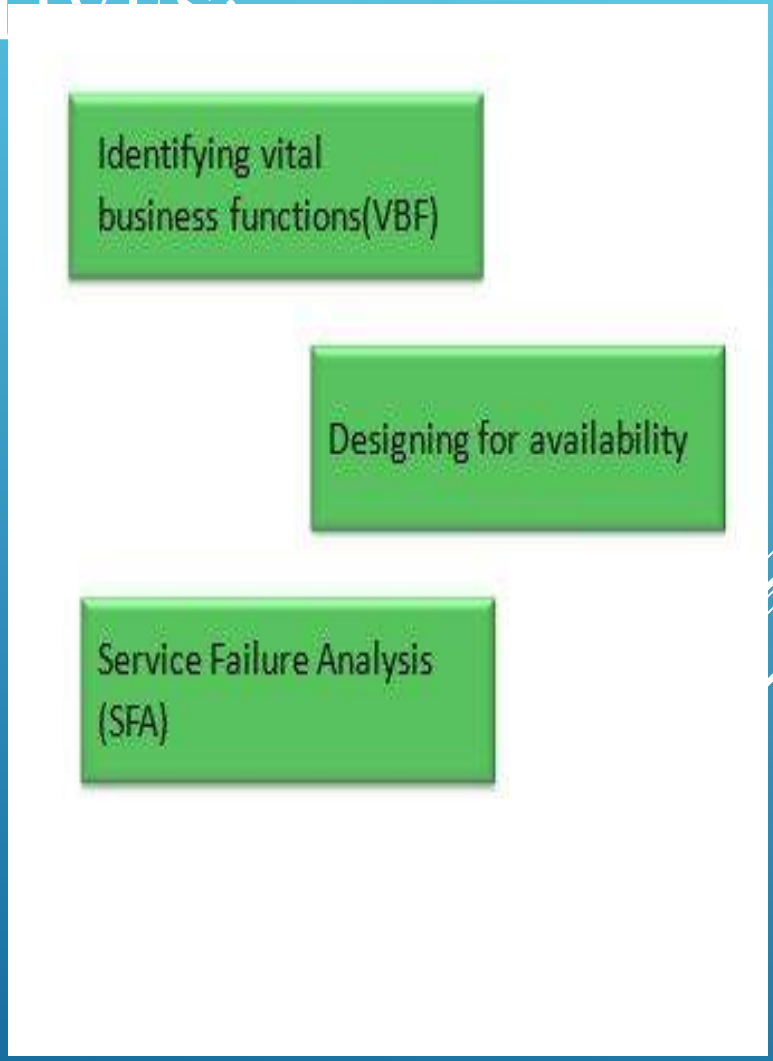
DON'T FORGET

- Service availability is at the core of customer satisfaction and business success:

  poor service performance is defined as being unavailable.

ITIL AVAILABILITY MANAGEMENT SUB-PROCESSES AND THEIR PROCESS OBJECTIVES:

- **Design Services for Availability**
- to fulfill the agreed availability levels.
- **Availability Testing**
- **Availability Monitoring and Reporting**
  - comparing achieved vs. agreed availability



- Availability Management Process

1. Reactive activities
2. Proactive activities

Why Availability Management

- To ensure services are available when the customer needs them
- Influences
  - Business **Demand**
  - **Cost** required to meet demand
- **Complexity** of IT Infrastructure
  - Levels of **Redundancy**
  - **Reliability** of the Infrastructure
  - Level of Maintenance
- Processes and procedures used by Services
- Human Factors
  - Skill sets

HOW TO MEASURE IT ?

Availability: the ability of a service, component to perform its agreed function when required.

$$\text{Availability (\%)} = \frac{\text{(Agreed Service Time (AST) - downtime)}}{\text{AST}} \times 100\ \%$$

Mean Time Between Service Incidents (MTBSI)
or Mean Time Between Failures (MTBF):

$$\text{Reliability (MTBSI in hours)} = \frac{\text{Available time in hours}}{\text{Number of breaks}}$$

$$\text{Reliability (MTBF in hours)} = \frac{\text{Available time in HRS- Total downtime in HRS}}{\text{Number of breaks}}$$

THE AVAILABILITY MANAGEMENT PROCESS DEPENDS HEAVILY ON THE MEASUREMENT OF SERVICE AND COMPONENT ACHIEVEMENTS WITH REGARD TO AVAILABILITY.

situation where a 24 x 7 service has been running for a period of 5,020 hours with only two breaks, one of 6 hours and one of 14 hours, would give the following figures:

Availability = (5,020–(6+14)) / 5,020 x 100 = 99.60%

Reliability (MTBSI) = 5,020 / 2 = 2,510 hours

Reliability (MTBF) = 5,020–(6+14) / 2 = 2,500 hours

MICROSOFT ONLINE SERVICES

Service Level Commitment. The minimum "Monthly Uptime Percentage" for a Service is calculated by the following formula:

$$\frac{\text{User Minutes} - \text{Downtime}}{\text{User Minutes}} \times 100$$

If the Monthly Uptime Percentage falls below 99.9% for any given month, you may be eligible for the following Service Credit:

Monthly Uptime Percentage	Service Credit
< 99.9%	25%
< 99%	50%
< 95%	100%

Service Credit Claim. If we fail to meet the minimum Monthly Uptime Percentage described above for a Service, you may submit a claim for a Service Credit.

IT ONLINE SERVICES AVAILABILITY

Table 1: Availability Values of Money Collection Options

Application or Component	SLA Minutes (based on core hours)	Outage Minutes	Availability Percent
Banking application	8 a.m. to 6 p.m. = 600 minutes	5 minutes	99.17 percent
Third-party administration application	8 a.m. to 6 p.m. = 600 minutes	10 minutes	98.33 percent
Voice recognition application	8 a.m. to 6 p.m. = 600 minutes	5 minutes	99.17 percent
Website	24 hours x 7 days = 1,440 minutes	15 minutes	98.96 percent
Payment system database	24 hours x 7 days = 1,440 minutes	6 minutes	99.58 percent
Account information system database	24 hours x 7 days = 1,440 minutes	3 minutes	99.79 percent
Overall	**6,120 minutes**	**44 minutes**	**99.28 percent**

Service level agreement

CLASSIFICATION OF FAILURES

1. **Transient**
   - **Occur only for certain input values while invoking a function of the system.**
2. **Permanent**
   - **Occur for all input values while invoking a function of a system.**
3. **Recoverable**
   - **When recoverable failures occur, the system recovers with or without operator intervention.**

4. **Unrecoverable**

- **The system may need to be restarted.**

5. **Cosmetic**

- **These failures cause minor irritations, & do not lead to incorrect results**

- **E.g. – mouse button need to be clicked twice to invoke a specific function**

RELIABILITY GROWTH MODELING

RELIABILITY GROWTH MODEL

- A reliability growth model is a mathematical model of how software reliability improves as errors are detected and repaired.
- A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained.
- Thus, reliability growth modeling can be used to determine when to stop testing to attain a given reliability level.
- Although several different reliability growth model are availbe , we will discuss few:

1. Jelinski-moranda model
2. Little wood and Verall's model
3. Step function model

RELIABILITY GROWTH MODELING

- It is a **mathematical model of how the s/w reliability improves as errors are detected & repaired.**
- This model tells us **how the system reliability changes over time during the testing process**.
- It can be **used to predict when a particular level of reliability is likely to be attained**
- It is **used to determine when to stop testing to attain a given reliability level.**

JELINSKI & MORANDA MODEL

- This is a simple step function model where the **reliability increases by a constant increment each time a fault is discovered & repaired**

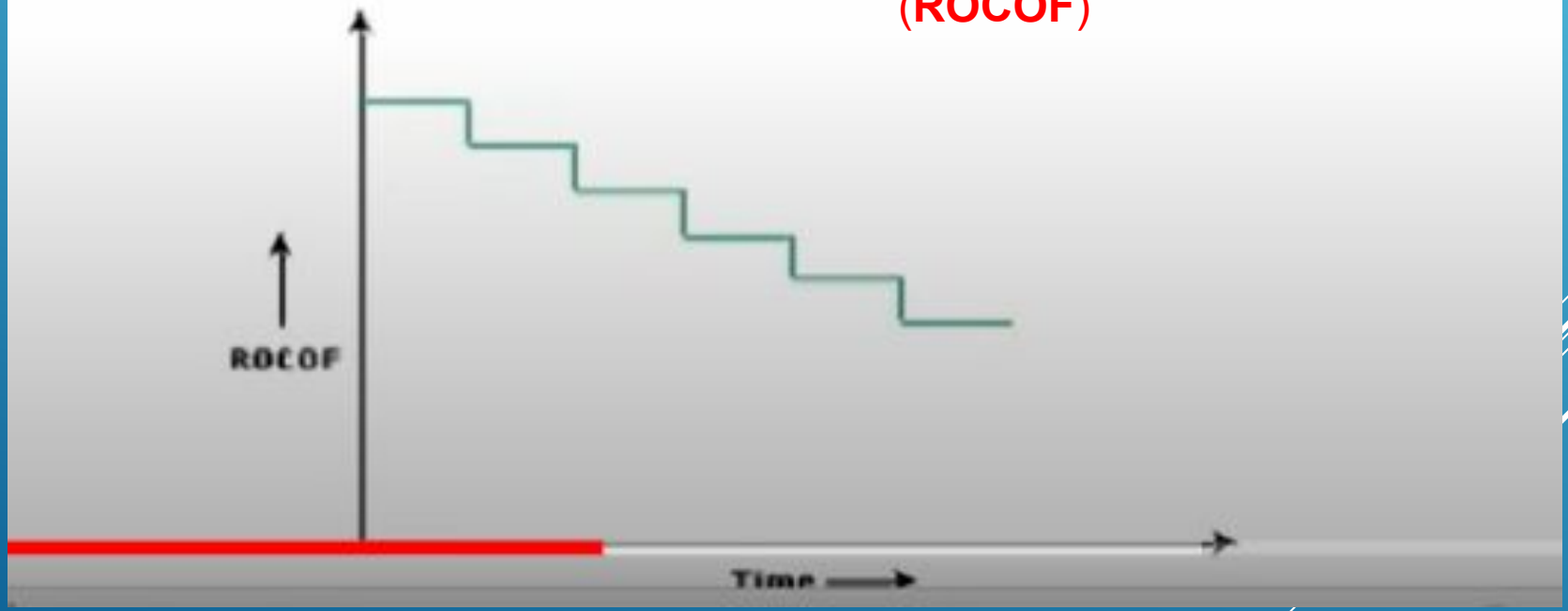
- This model **assumes that s/w repairs are always correctly implemented** so **that the no. of s/w faults & associated failures decreases in each new version of the system**

- **It assumes that all errors contributes equally to reliability growth**

JELINSKI AND MORANDA MODEL

The simplest reliability growth model is a step function model where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. Such a model is shown in fig.. However, this simple model of reliability which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since it is already known that correction of different types of errors contribute differently to reliability growth.

Rate of occurrence of failure (**ROCOF**)



Jelinski-moranda model

- At beggining it assumes it has N faults
- Each fault in code is independent
- Time interval between software failure are different
- No new faults are introduced.
- Failure rate is proportional to the number of faults remaining in the software

The failure intensity is given by

$$\lambda(t) = \phi\,(N - i + 1)$$

Rate of occurrence of failure (**ROCOF**)
It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. ... A **ROCOF** of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric

LITTLEWOOD AND VERALL'S MODEL

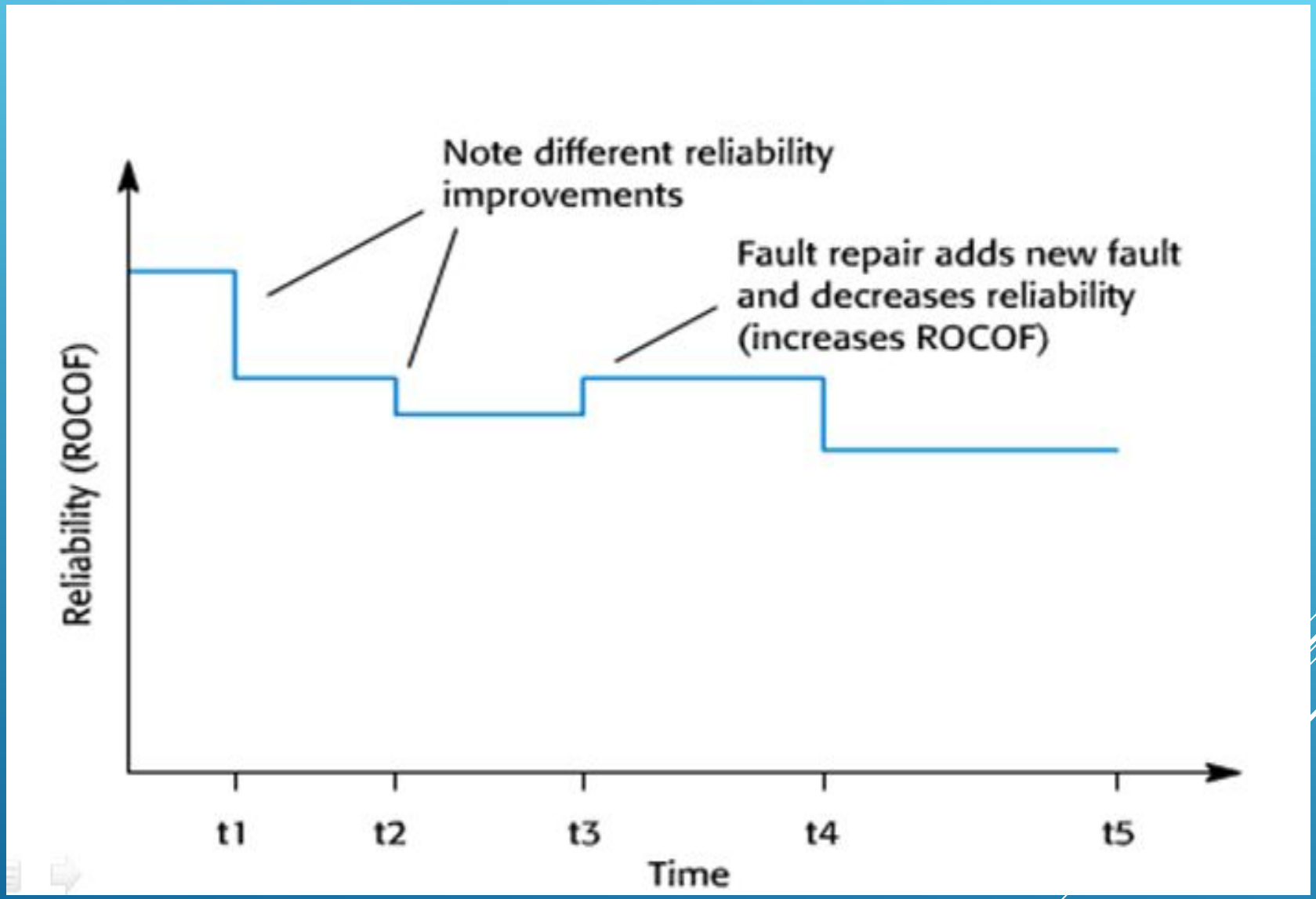
This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases (Fig. 13.3). It treat's an error's contribution to reliability improvement to be an independent random variable having Gamma distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as test continues.

LITTLEWOOD & VERALL'S MODEL

- This model **allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors**.

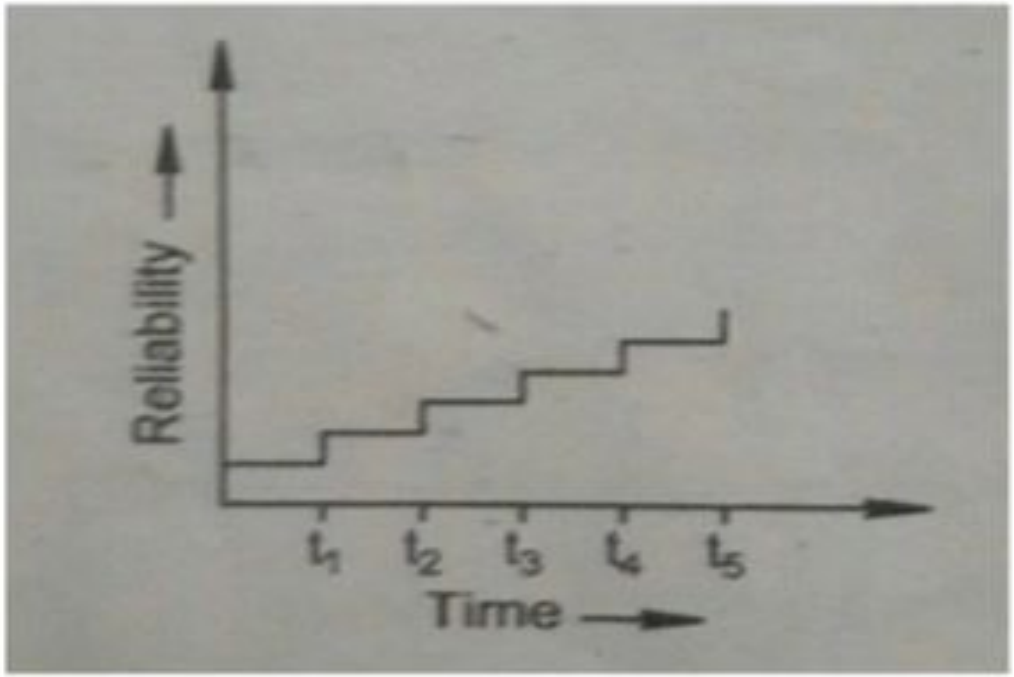
- It also says that **as faults are repaired, the average improvement in reliability per repair decreases.**

- This shows that **each repair does not result in equal amount of reliability improvement**

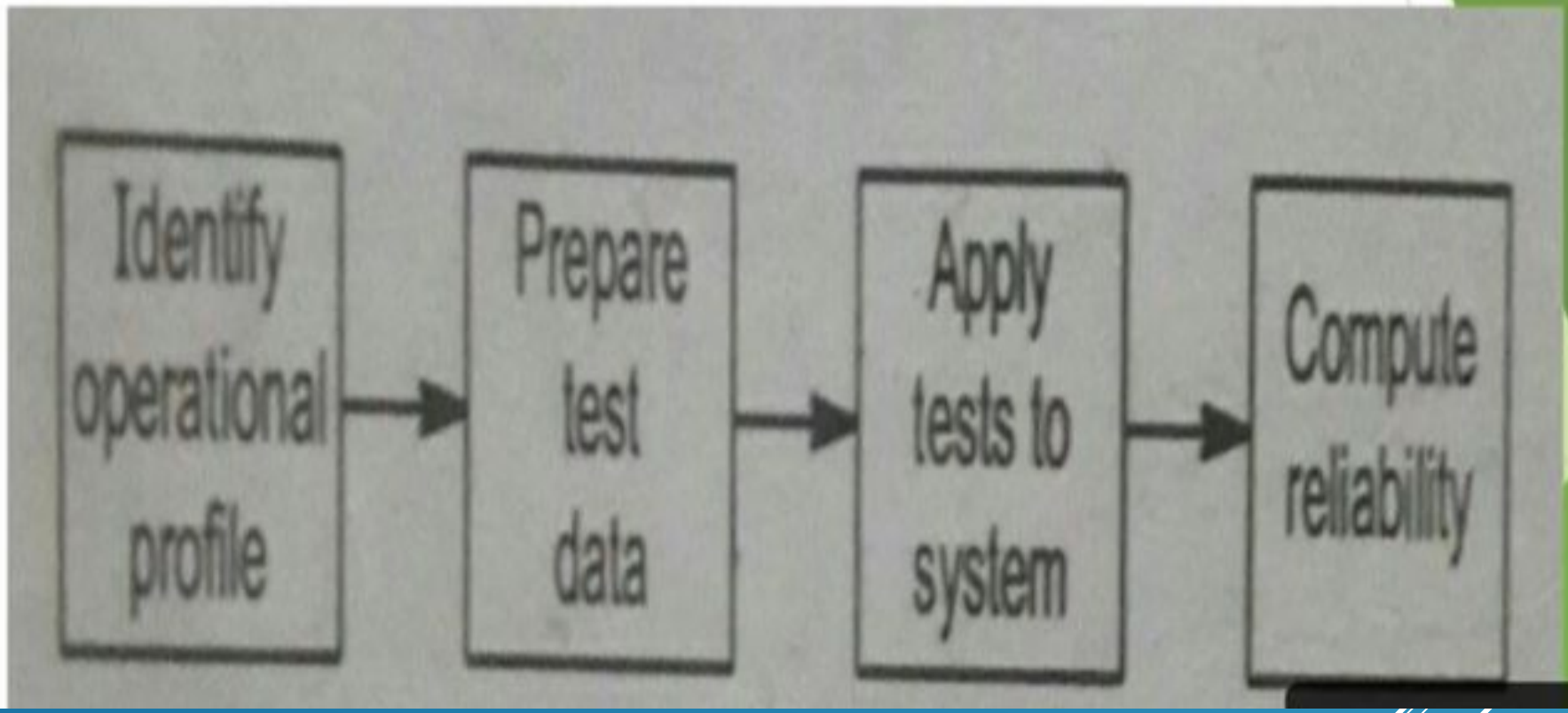

Note different reliability improvements
Fault repair adds new fault and decreases reliability (increases ROCOF)
Reliability (ROCOF)
t1
t2
t3
t4
t5
Time

- It is a constant growth model finding error and repair and this is not feasible in real world

Step function model



Reliability Measurement Process



RELIABILITY TESTING TUTORIAL: WHAT IS, METHODS, TOOLS, EXAMPLE

- checks whether the software can perform a failure-free operation for a specified period of time in a specified environment.

RELIABILITY TESTING CAN BE CATEGORIZED INTO THREE SEGMENTS

1. Modelling
2. Measurement
3. Improvement

FACTORS INFLUENCING SOFTWARE RELIABILITY

1. The number of faults presents in the software

2. The way users operate the system

WHY TO DO RELIABILITY TESTING

1. To find the **structure of repeating failures**.
2. To find the **number of failures occurring** is the specified amount of time.
3. To discover the main cause of failure
4. To conduct **Performance Testing** of various modules of software application after fixing defect

TYPES OF RELIABILITY TESTING

- Feature Testing
- Load Testing
- Regression Test

HOW TO DO RELIABILITY TESTING

- Establish reliability goals
- Develop operational profile
- Plan and execute tests
- Use test results to drive decisions
- The key parameters involved in Reliability Testing are:-
  - Probability of failure-free operation
  - Length of time of failure-free operation
  - The environment in which it is executed

STEP 1) MODELLING

- 1. Prediction Modelling
- 2. Estimation Modelling

Issues	Prediction Models	Estimation Models
Data Reference	It uses historical data	It uses current data from the software development.
When used in Development Cycle	It will be usually created before the development or testing phases.	It will be usually used at the later stage of Software Development Life Cycle.
Time Frame	It will predict the reliability in the future.	It will predict the reliability either for the present time or in the future time.

STEP 2) MEASUREMENT

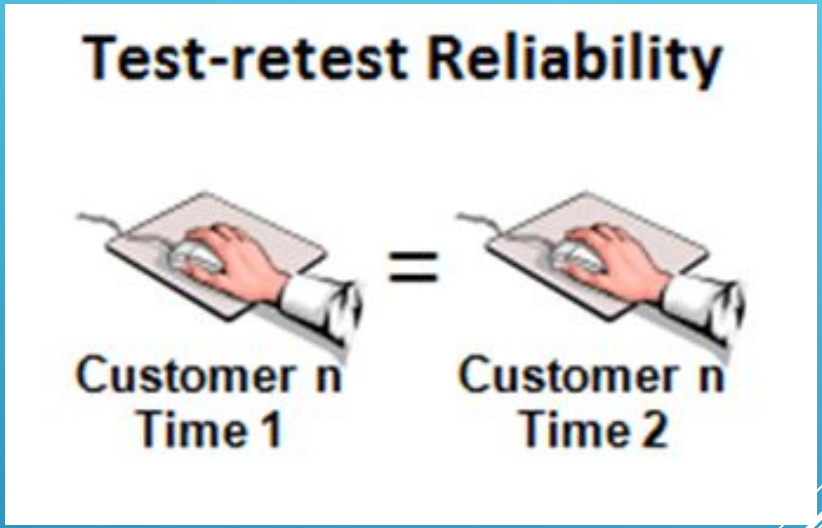
1. **Product Metrics:-**
   - **Software size**
   - **Function point metric**
   - **complexity**
   - **Test coverage metric**
2. **Project Management Metrics**
3. **Process Metrics**
4. **Fault and Failure Metrics**

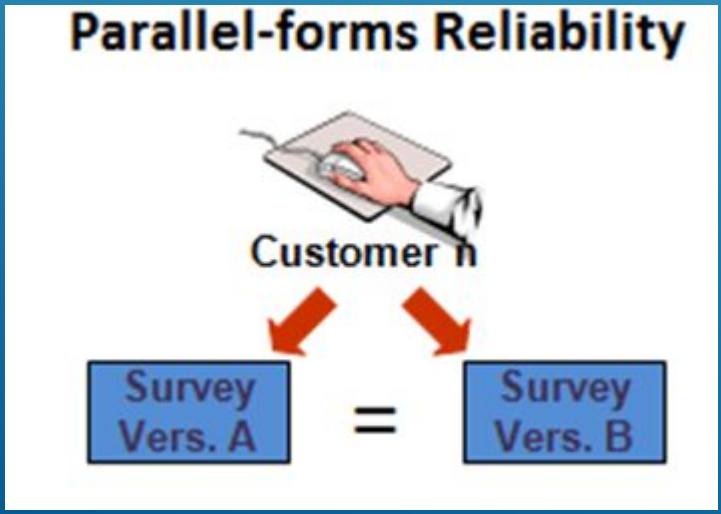
STEP 3) IMPROVEMENT

- Improvement completely depends upon the problems occurred in the application or system, or else the characteristics of the software.

EXAMPLE METHODS FOR RELIABILITY TESTING

- There are mainly three approaches used for Reliability Testing
  1. Test-Retest Reliability
  2. Parallel Forms Reliability
  3. Decision Consistency





- DECISION CONSISTENCY
- After doing Test-Retest Reliability and Parallel Form Reliability, we will get a result of examinees either pass or fail.
- It is the reliability of this classification decision that is estimated in decision consistency reliability.

5 ASPECTS OF QUALITY IN A BUSINESS CONTEXT:

1

- Producing – providing something.

2

- Checking – confirming that something has been done correctly.

3

- Quality Control – controlling a process to ensure that the outcomes are predictable.

4

- Quality Management – directing an organization so that it optimizes its performance through **analysis** and **improvement**.

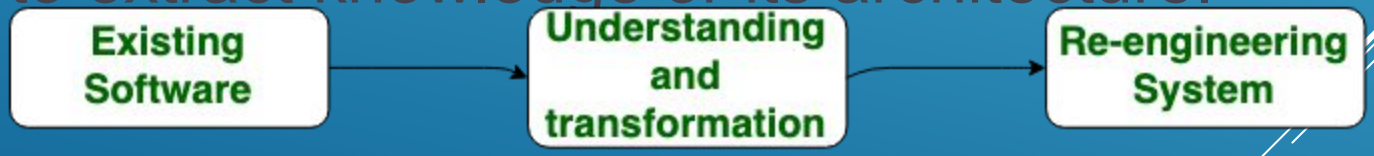
5

- Quality Assurance – obtaining confidence that a product or service will be satisfactory. (Normally performed by a purchaser)

SOFTWARE REVERSE ENGG

REVERSE ENGINEERING:

- Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse.
- In this, the information are collected from the given or exist application.
- It takes less time than forward engineering to develop an application. In reverse engineering the application are broken to extract knowledge or its architecture.



Reverse Engineering

REVERESE ENGINEERING

In reverse engineering or backward engineering, the information are collected from the given application.

Reverse Engineering or backward engineering is low proficiency skill.

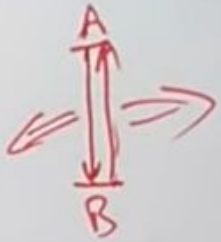
While Reverse Engineering or backward engineering takes less time to develop an application.

The nature of reverse engineering or backward engineering is Adaptive.

In reverse engineering, production is started by taking existing product.

The example of backward engineering are research on Instruments etc.

Reverse Engineering

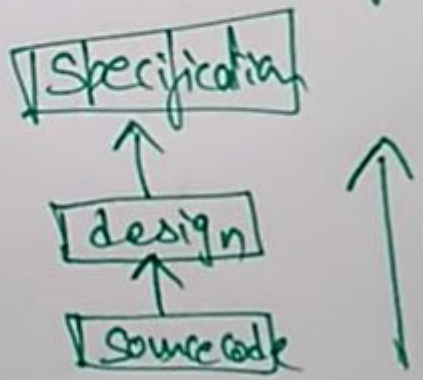


Forward Engineering:

In s/w development life cycle, we start from specification phase, analysis, design & implementation

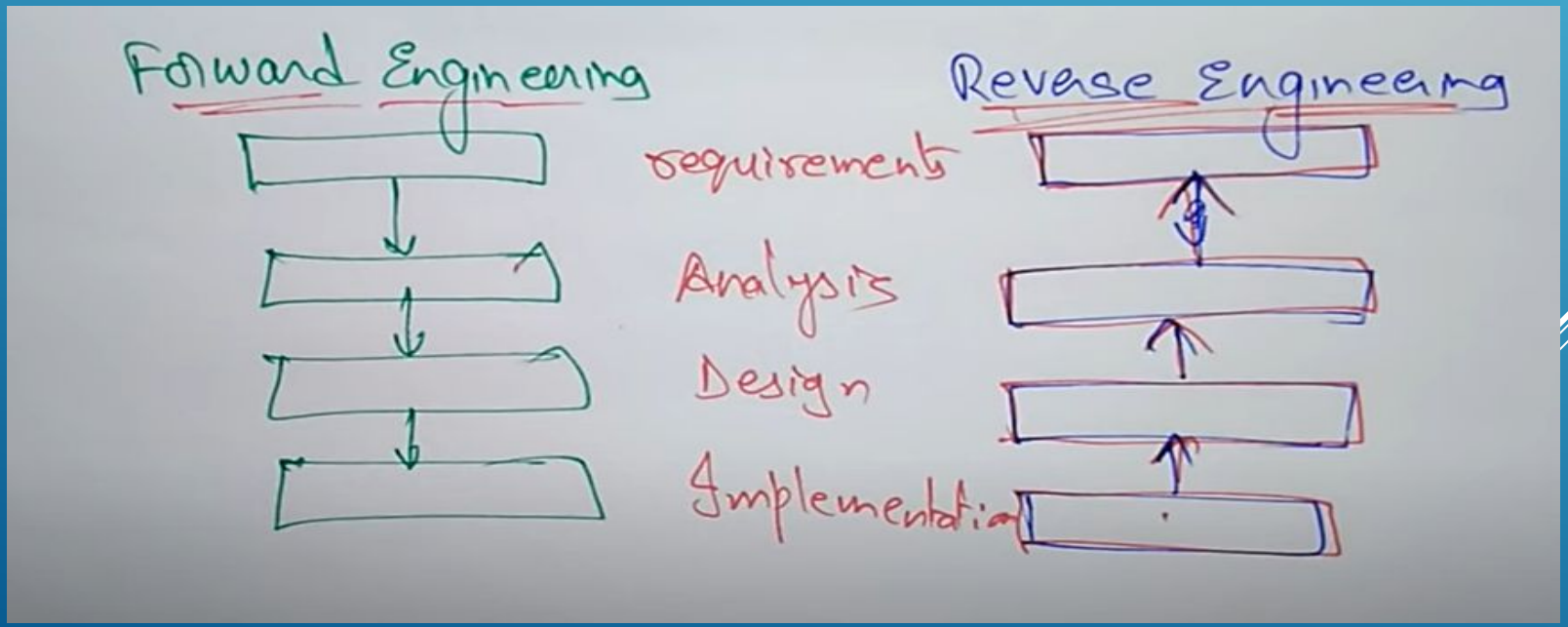
Reverse Engineering:

Process of creating of design document from source code & the specification doc from design doc



REVERSE ENGG

- ✔ Program comprehension
- ✔ Analysis existing s/w with a view to understand its design and specification
- ✔ A process which analyses a product t find out the design aspects and its functions
- ✔ Builds a program db and generates information from this



REVERSE ENGG goals:

- ✔ Cope up with complexity
- ✔ Recover lost information
- ✔ Detect the effects
- ✔ Synthesize higher abstraction
- ✔ Facilitate reuse

REVERSE ENGG Activities:

- ✔ Understanding process
- ✔ Understanding data
- ✔ Understanding user interfaces

USAGE OF REVERSE ENGG

- Malware developers often use reverse engineering techniques to find vulnerabilities in an operating system to build a computer virus that can exploit the system vulnerabilities.
- Reverse engineering is also being used in cryptanalysis to find vulnerabilities in substitution cipher, symmetric-key algorithm or public-key cryptography.

Reverse Engineering

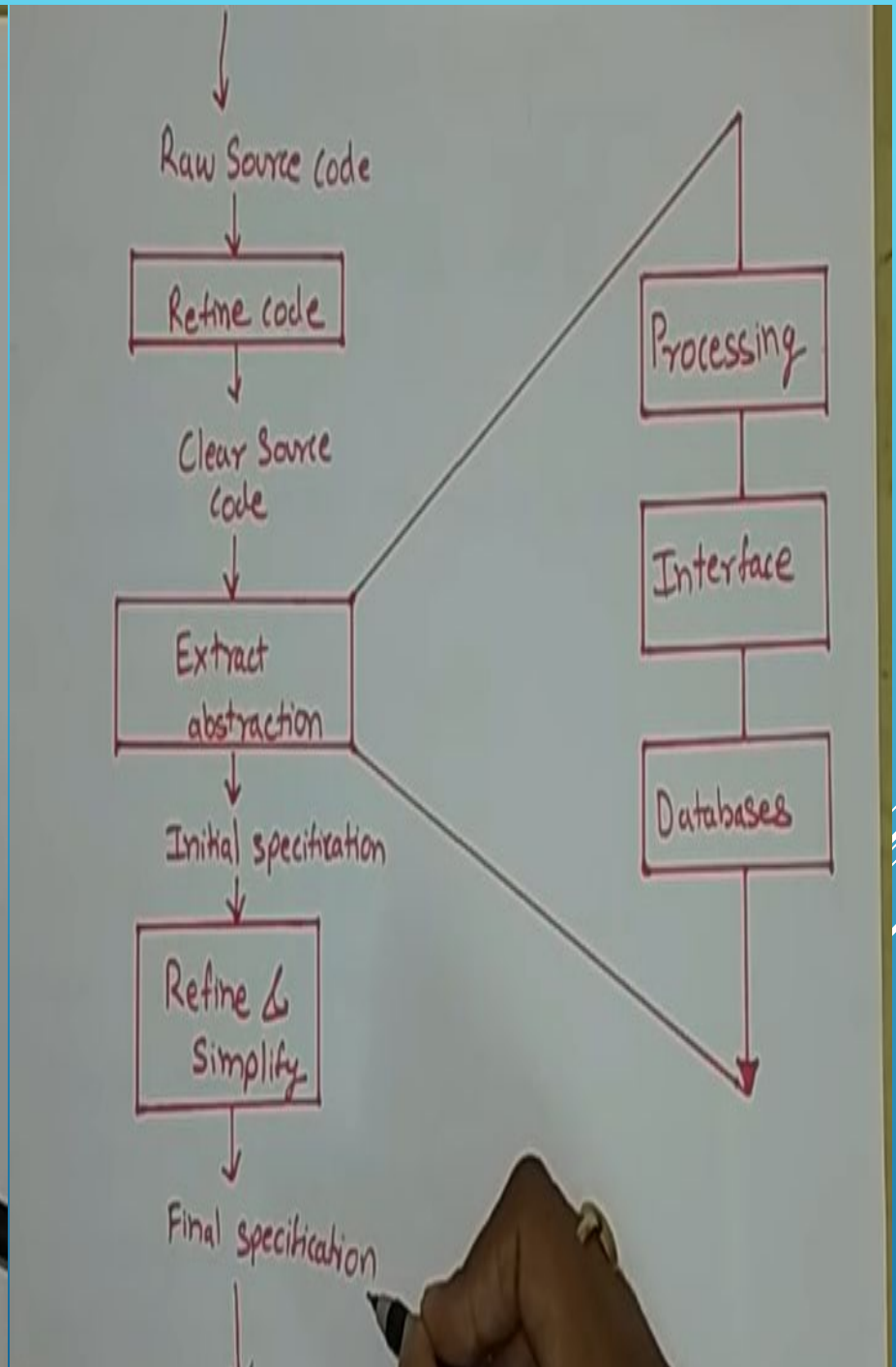
- System is analysed at higher level of abstraction.
  Design Dekh ke source code

- Knowledge and design information is extracted from the source code

  Source code se Naya Design

- Purpose :-
  - Used as learning tools
  - Security auditing
  - Enabling additional features
  - Developing compatible products cheaper than that are currently available in the market.

USED



REVERSE ENGINEERING

It is the process followed so as to find unknowns and difficult information about a Software.

Why Is It Important ?

It is important in case when software lacks proper documentation, is highly unstructured or its structure has degraded through maintenance works.

Purpose: Recovering information from the existing code/any intermediate documents
→ program understanding at any level is included in R.E.

Uses of Reverse Engineering

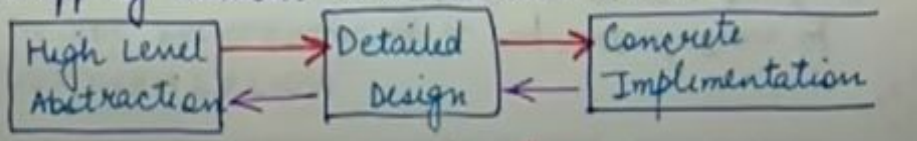
- ▲ Program Understanding
- ▼ Re-documentation or document generation
- ▲ Recovery of design details
- ▼ Identify Reusable components.
- ▲ Business Rules implied on SW.
- ▼ Understanding high level system description
- ▲ Identify components that require restructuring.

What is not in scope of Reverse Engineering ?

* Re-design * Re-structuring
* Enhancement of system functionality.

Major Tasks

1. Mapping the program to the (problem it represents in) application domain.
2. Mapping between concrete and abtract levels

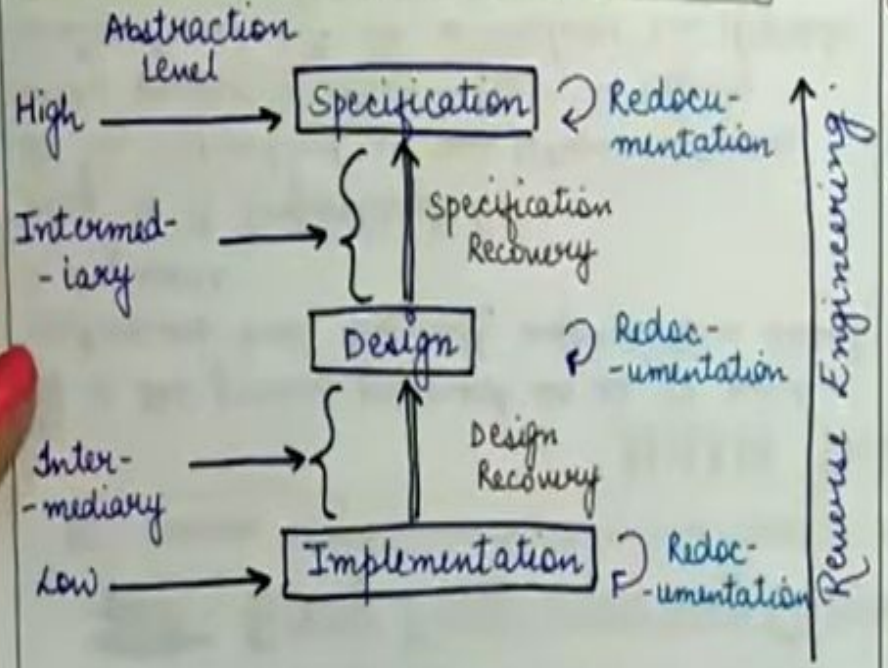


3. Rediscover high level structures
4. Find missing links between program syntax and semantics.
5. Extract Re-usable component
   → to find the number of components that can be reused from this code.

4. Find missing links between program syntax and semantics.

5. Extract Re-usable component
   ↳ to find the number of components that can be reused from this code.

LEVELS OF REVERSE ENGINEERING



REDOCUMENTATION

It is the recreation of a representation of a system which is equivalent to the existing system and at the same level of abstraction

Goals: To create alternate views of system to improve understanding.

② Improve existing documentation
③ Create documentation for a program that has been recently modified.

DESIGN RECOVERY

Identify and extract higher-level abstractions
↳

The obtained design is used as a basis for future system modification or development of similar application.

SOFTWARE MAINTENANCE

SOFTWARE MAINTENANCE

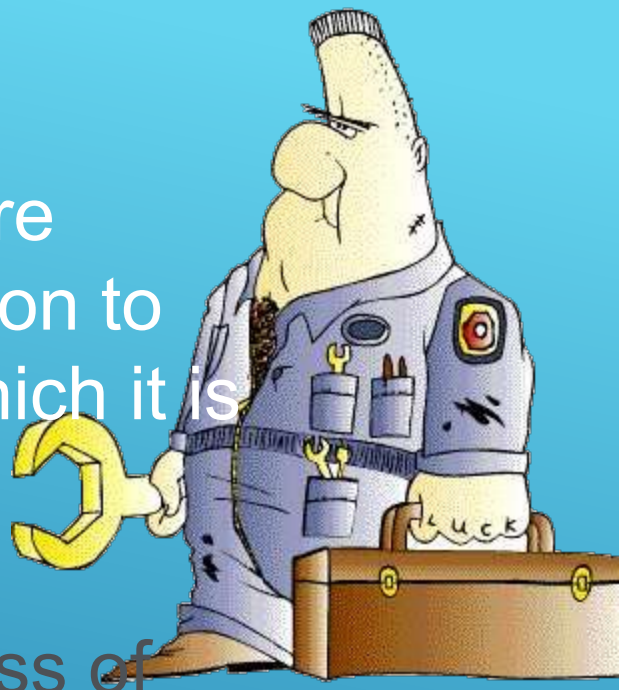
SOFTWARE MAINTENANCE

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

DEFINITIONS

The act of keeping, or the expenditure required to keep, an asset in condition to perform efficiently the service for which it is used.



Software Maintenance is the process of modifying a **software** product after it has been delivered to the customer. The main purpose of **software maintenance** is to modify and update **software** application after delivery to correct faults and to improve performance.

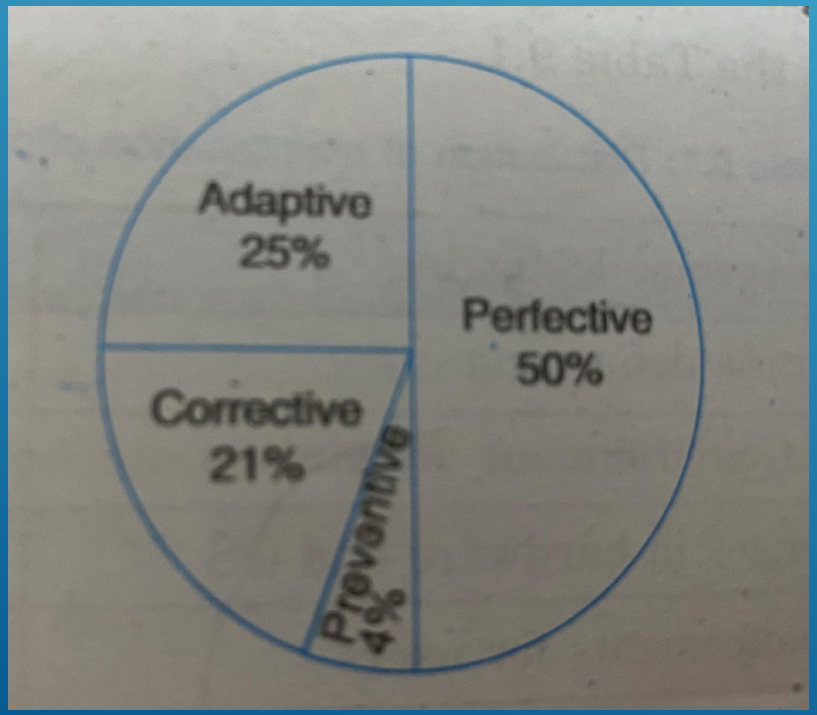
TYPES OF MAINTENANCE

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance 21%**- 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.
- **Adaptive Maintenance 25%** - 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. [Porting and migration]
- **Perfective Maintenance 50%**- 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 improve its reliability and performance[enhancement and scalbiility.]
- **Preventive Maintenance 4%** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.[s/w re-engg]

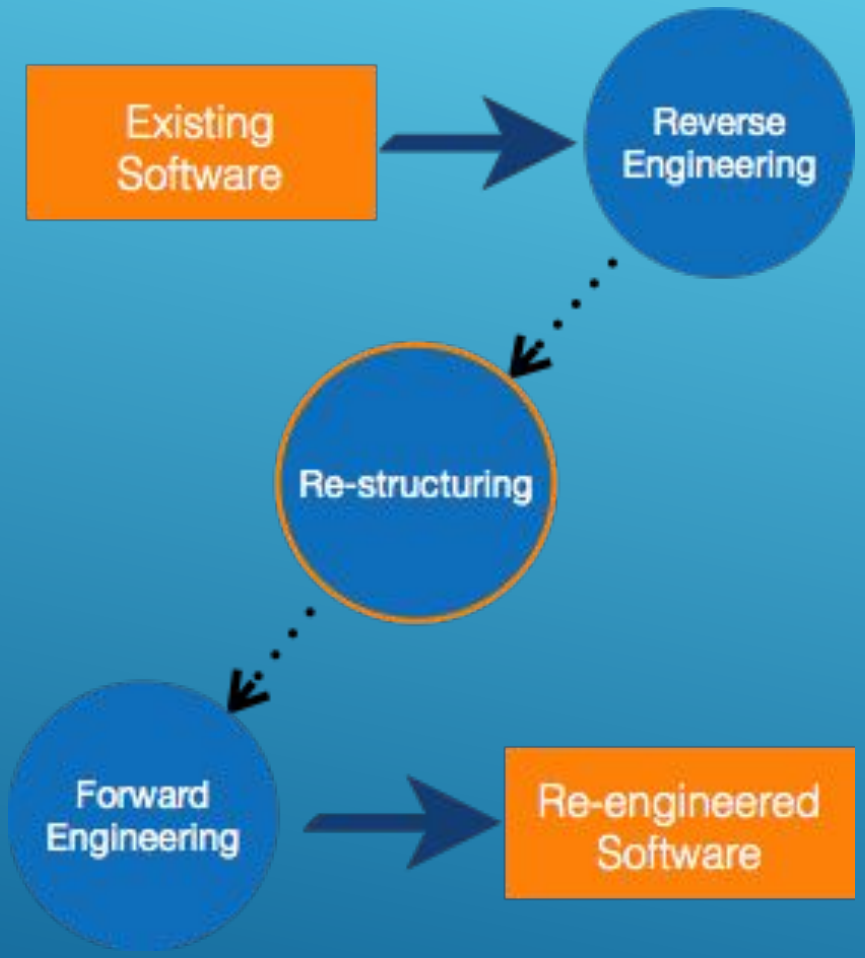
TYPES OF MAINTENANCE

1. Corrective maintenance:
2. Adaptive maintenance
3. Preventive maintenance
4. Perfective maintenance



SOFTWARE RE-ENGINEERING

- When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.
- Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.
- For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.
- Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.
-



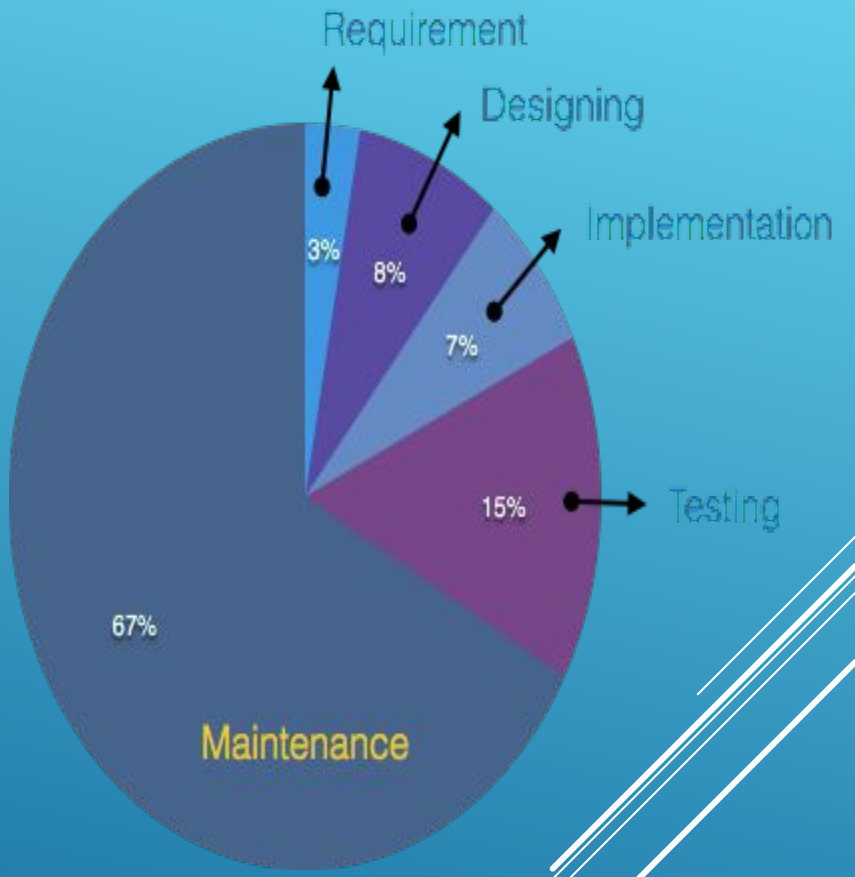
Re-Engineering Process

- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

COST OF MAINTENANCE

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

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 up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

MAINTENANCE ACTIVITIES

- IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.

- These activities go hand-in-hand with each of the following phase:
- **Identification & 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.Here, the maintenance type is classified also.
- **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. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.
- Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

- Quick-fix model
- Iterative enhancement model
- Reuse oriented model
- Boehm's model
- Taute maintenance model

MAINTENANCE MODELS-DO IT URSELF

SOFTWARE BREAKDOWN CAUSES

- basic conditions neglected
- inadequate skills
- operating standards not followed
- deterioration unchecked
- inherent design weakness

MAINTENANCE COSTS

Maintenance costs are usually greater than development costs by a factor of 2 to 100.

- The costs arise from both technical and nontechnical factors.
  - A deployed system is expensive to change
  - High cost of breaking an already working system
  - Maintenance costs increase over time and as the system evolves.

Reasons:

Maintenance changes ,degrades the original system structure.

Aging software results in high support costs.

MAINTENANCE COST FACTORS

Team stability

Maintenance costs are reduced if the same staff are involved with them for some time.

Contractual responsibility

The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.

Staff skills

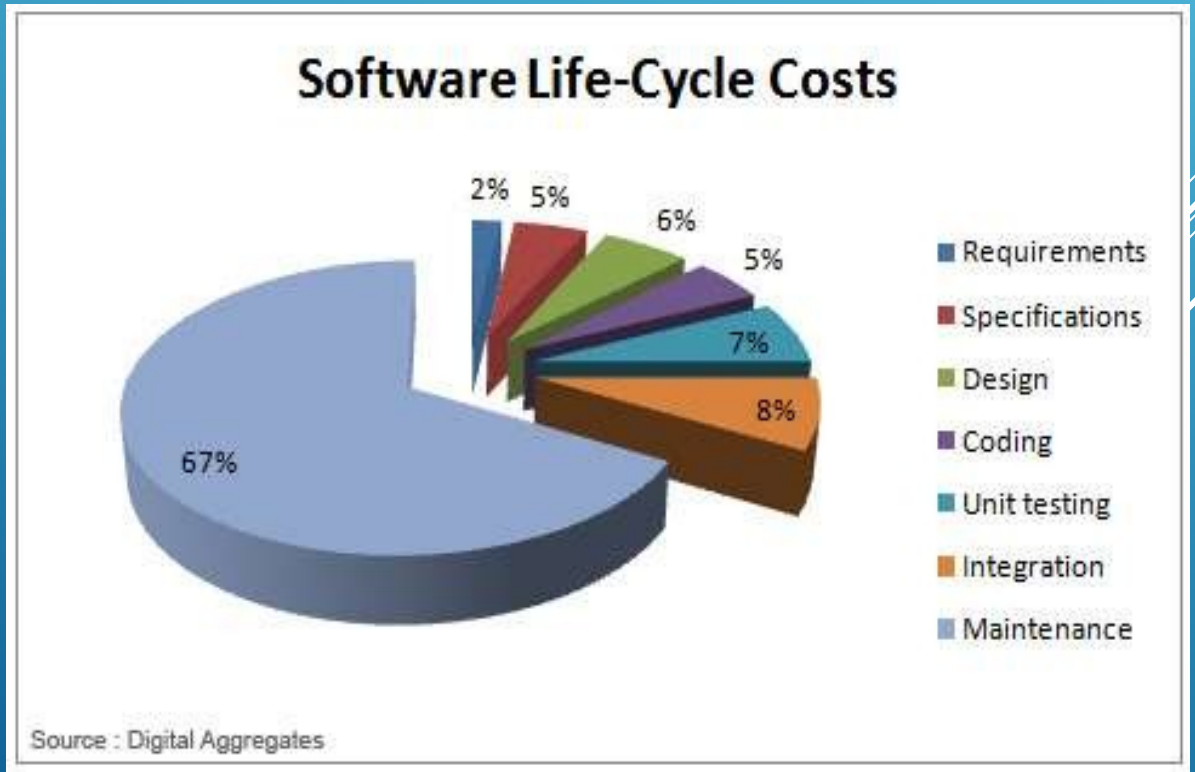
Maintenance staff are often inexperienced and have limited domain knowledge.

Program age and structure

As programs age, their structure is degraded and they become harder to understand and change.

STRATEGIES TO REDUCE MAINTENANCE COSTS:

- Correct slight defects in parts and jigs.
- Ensure basic equipment conditions are maintained
- Review basic operations
- Conduct physical analysis
- Adopt an analytical approach



MAINTENANCE PROBLEMS

- Someone else's program.
- Developer not available.
- Proper documentation doesn't exist.
- Not designed for change.
- Maintenance activity not highly regarded.

SOFTWARE MAINTANANCE - CHANGE MANAGEMENT

- **Objective:** *Change Management*
  - **aims to control the lifecycle of all Changes**.
  - **to enable beneficial Changes to be made, with minimum disruption to IT services.**

- **Part of**: Service Transition
- **Process Owner**: Change Manager

Change

The addition, modification or removal of anything that could have an effect on IT services. The scope should include changes to architectures, processes, tools, metrics and documentation, as well as changes to IT services and other CI's

Standard Change

A pre-authorized change that is LOW risk, relatively common and follows a procedure or work instruction – for example a password reset or provision of standard equipment to a new employee. **Requests for Change (RFC)** are not required to implement a standard change, and are logged and tracked using a different mechanism, such as a **service request**.

Normal Change

A change that is not an emergency change or standard change. Normal changes follow defined steps of the change management process.

Emergency Change

A change that must be introduced as soon as possible – for example, to resolve a major incident or implement a security patch. The change management process will normally have a specific procedure for handling emergency changes.

Major Release

Normally contain large areas of new functionality some of which may eliminate temporary fixes to problems.

A major upgrade or release usually supersedes all preceding minor upgrades, release and emergency fixes.

Minor Release

Normally contain small enhancements and fixes, some of which may already have been issued as emergency fixes.

A minor upgrade or release usually supersedes all preceding emergency fixes.

Emergency Release

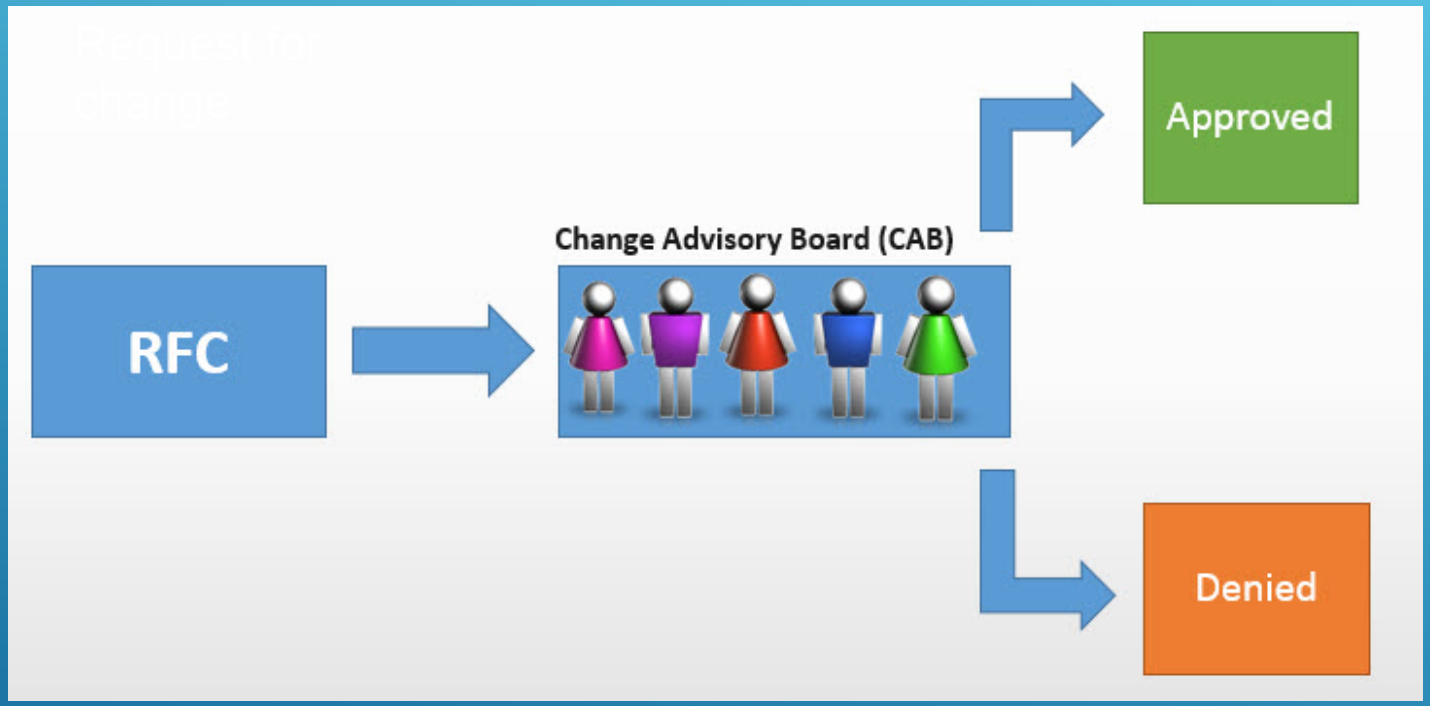
Normally contain corrections to a small number of know errors, or sometimes an enhancement to meet a high priority business requirement.

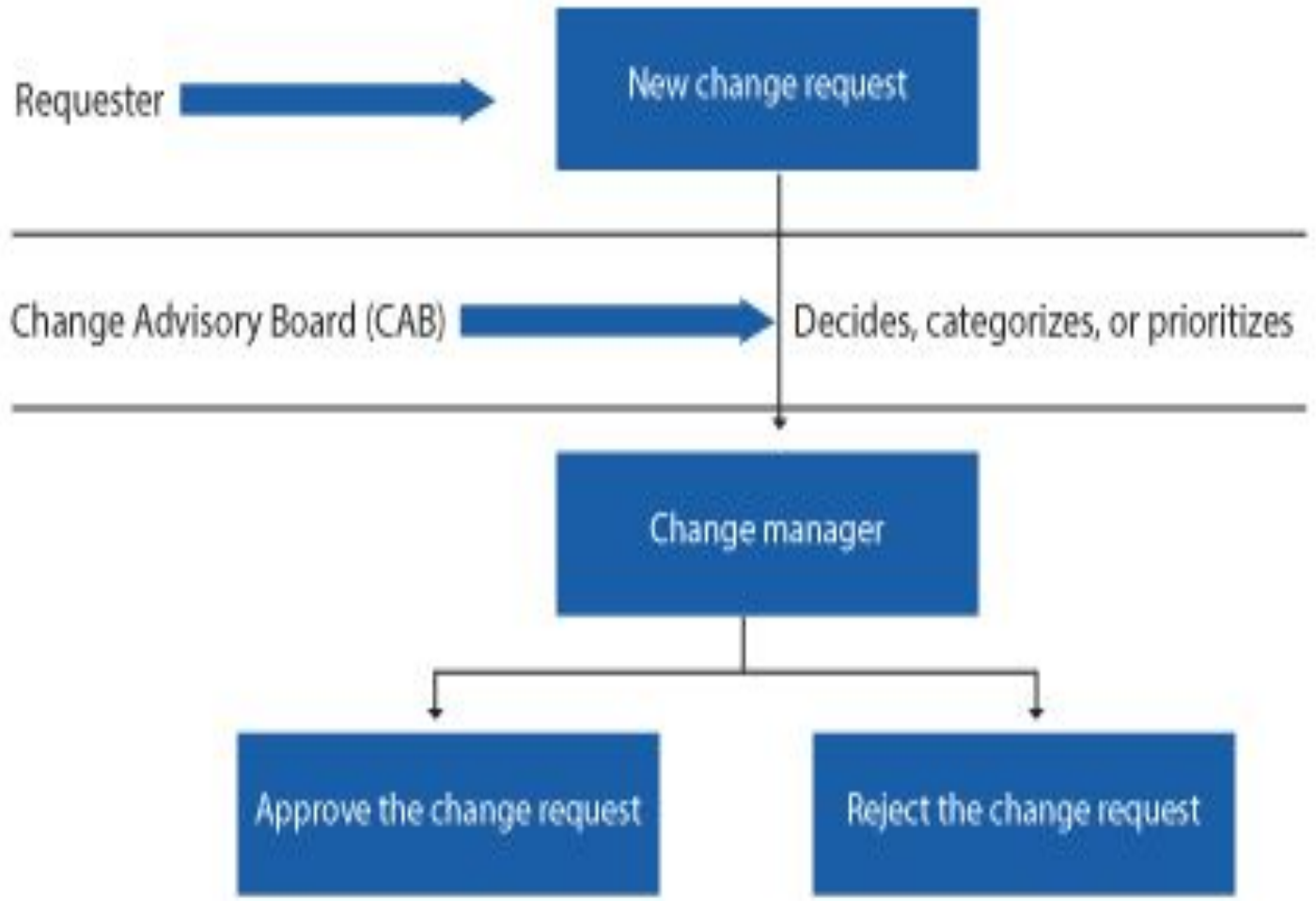
Known Error

A problem that has a documented root cause and a **workaround**. Known errors are created and managed throughout their lifecycle by problem management. Known errors may also be identified by development and suppliers.

Change Advisory Board

A group of people that support the assessment, prioritization, authorization and scheduling or changes. A change advisory board is usually made up of representatives from: all areas within the IT service provider; the business; and third parties such as suppliers.


Request for change
Approved
Change Advisory Board (CAB)
RFC
Denied


Requester
New change request
Change Advisory Board (CAB)
Decides, categorizes, or prioritizes
Change manager
Approve the change request
Reject the change request

Emergency Change Advisory Board

A subgroup of the change advisory board that makes decisions about emergency changes. Membership may be decided at the time a meeting is called, and depends on the nature of the emergency change.


Change Management Process
1 Request for Change
2 Impact Analysis
3 Approve / Deny
4 Implement Change
5 Review / Reporting

Change Management Roles

Change Initiator

- Initiates change requests
- Can have input into assessment and planning of the change, building and testing of the change and, implementing, assessing and reviewing the change

Change / Release Coordinator

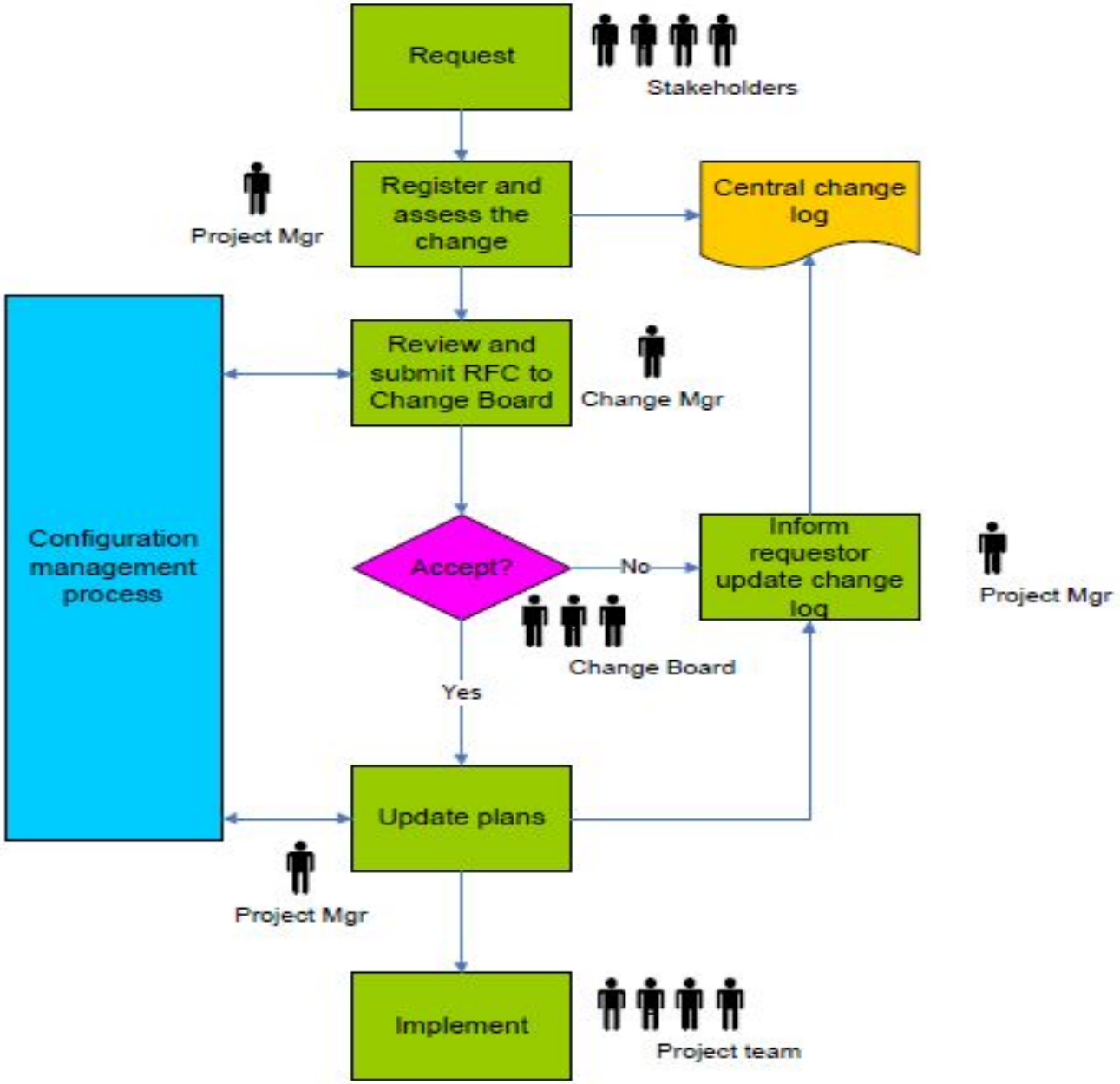
- Assesses change requests that are submitted for approval
- Monitors changes from the build to the test stages
- Coordinates risk and impact analysis
- Develops implementation plans by creating and assigning tasks
- Coordinate s building, testing and implementation of changes as per the Forward Schedule of Changes (FSC)

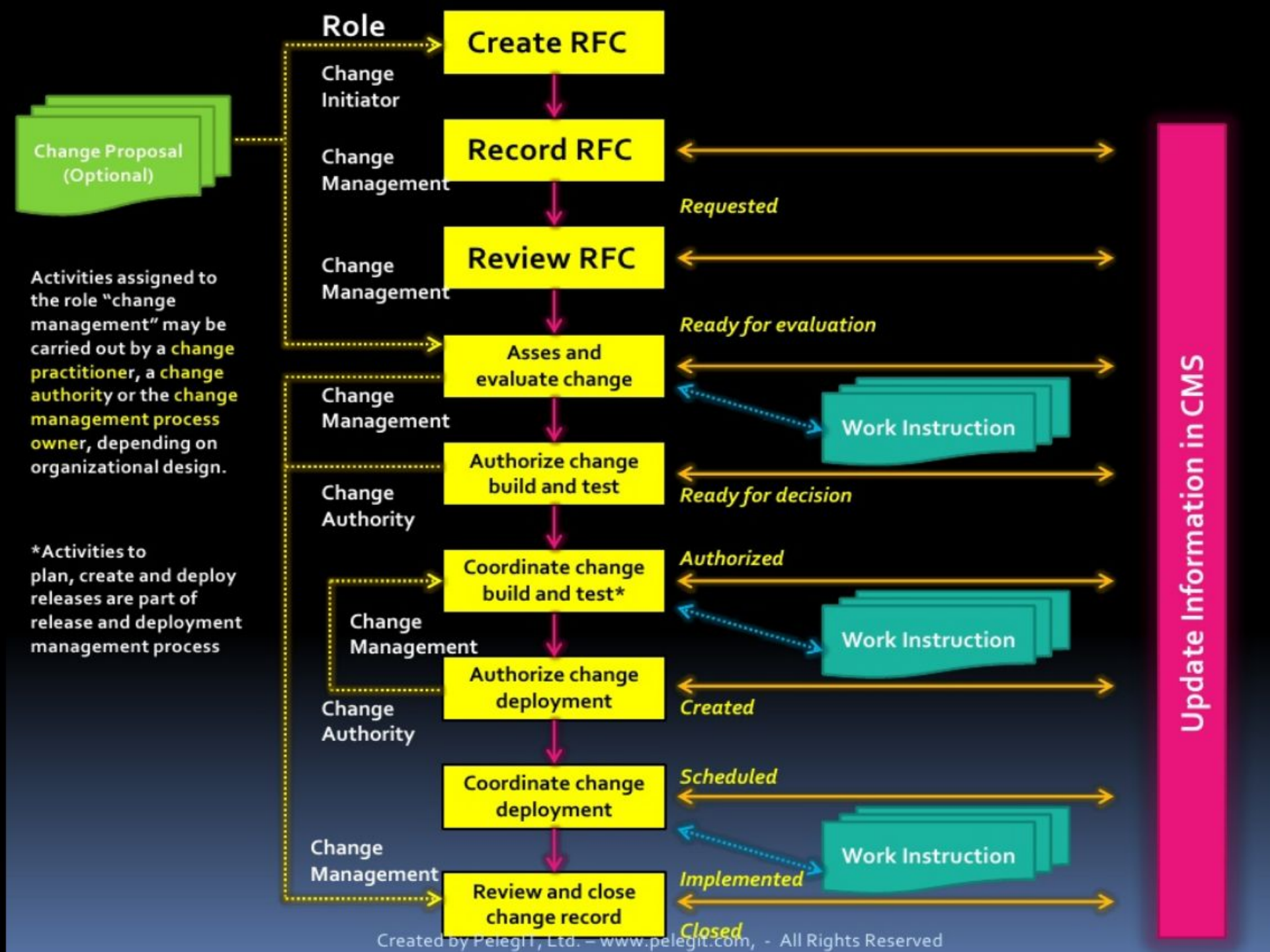
Change Manager

- Identified typically as the Change Management "process owner"
- Risk & Impact Analysis Review
- Ensures that the timing of changes don't conflict with other planned changes
- Facilitates the CAB Meetings
- Oversees obtaining approval for changes

CAB/CAB EC

- A group of people who have authority for assessing and advising on the implementation of major and significant changes and, meet/review/agree on pre-approved standard changes
- The Change Advisory Board /Emergency Committee (CAB/EC) is a subset of the full Change Advisory Board (CAB) which handles emergency change requests.


Request
Stakeholders
Project Mgr
Register and assess the change
Central change log
Review and submit RFC to Change Board
Change Mgr
Configuration management process
Accept?
No
Inform requestor update change log
Project Mgr
Change Board
Yes
Update plans
Project Mgr
Implement
Project team



Important steps of change:

1) The RFC is logged
2) An initial review is performed to filter RFC's
3) The RFC's are assessed – this may involve the CAB or ECAB
4) This is authorized by the change manager
5) Work orders are issued for the build change (Carried out by another group)
6) Change management coordinates the work performed (with multiple check points)
7) The change is reviewed
8) The Change is closed

Assessing and Evaluating Change

Impact	Typical Escalation Level
Standard	Excluded using a pre-defined form or template
Normal Changes	
Minor	Change Manager (CM) or other operational manager
Significant	Change Advisory Board (CAB)
Major	IT Management Board
Urgent Changes	
Normal	Change Management or Change Advisory Board (CAB)
Emergency	Emergency Change Advisory Board (ECAB)

7 R's of Change Management

1. Who **RISED** the change?
2. What is the **REASON** for the change?
3. What is the **RETURN** required from the change?
4. What are the **RISKS** involved in the change?
5. What **RESOURCES** are required to deliver the change?
6. Who is **RESSPONSIBLE** for the build, test and implementation of the change?
7. What is the **RELATIONSHIP** between this change and other changes?

- What is reliability
- Explain Reliability metrics
- What is Growth modelling
- Wha is software maintenance and estimation of maintenance costs

Thank you..