Project:

↳ A project is a series of tasks that need to be completed in order to reach a specific goal.

↳ A project involves multiple people (or individual) with different skill sets working together to produce a quality product.

↳ Characteristics:

(i) Every project has a unique and distinct goal.
(ii) Project is not a routine activity or day-to-day activity.
(iii) Project comes with a start time & end time.
(iv) Every project has temporary lifetime and is completed if goal is achieved.
(v) Every project must have adequate resources:
- time,
- cost / finance
- manpower
- knowledge bank.

↳ Thus, a Software project is a project which follows complete procedure of software development from requirement gathering to testing & maintainance, carried out according to the execution methodology.

↳ The s/w project has temporily lifeline and is completed within a specific amount of time.

Ioy. of s/w project fails.

↳ Causes of project failure:

(i) Poor planning:

→ It is one of the main reason for s/w project failure.

→ Lack of goal and planning ~~cause~~ makes it hard to know exactly what and when something needs to be done.
It leads project delay and ultimately causes failure.

(ii) Lack of leadership:

→ Leadership plays a major critical factor for s/w development.

→ Unable to provide clear and concise direction to the team member can cause project failure.

(iii) Poor Communication:

→ For a successful project compition and assure the quality of the product, there must be effective coordination and interaction between every individual involved on the project.

→ And, poor communication and interaction bet^n the team members can lead to project failure.

(iv) Inadequate use of resources:

→ Improper placement can lead to several problems, including schedule delays, cost overruns & quality defects.

→ If the resources are badly managed, the s/w product will fail to be delivered on time & within budget.

(v) Inability to overcome challenges:

→ Even with adequate resources, great team and have proper plan, sometime challenges might be faced while completing s/w project successfully.

→ Challenges might be:
(a) Defining project's scope correctly.
(b) Balancing client's need with those of the developer

#Solutions:

1) Defining the project's goal:

→ Before starting the project, there must be made clear goals in the mind.

→ so, few things must be considered when setting goals for software project:
> What do you want to achieve?
> Is this possible to do?
> How will you measure your progress?
> What are the potential consequences of not reaching your goals?
> Are your goals time related?

2) Establish project team structure wisely:

→ The team member must be chosen wisely for the project.

→ It ensures that our project will be completed on time, within budget and adequate resources.

→ Tips: - Define role & responsibility,
- Assign tasks & responsibity to individual,
- Create clean line of comm?

3) Create a project plan:

→ Proper project plan is essential steps on the success of any project or else it will be different difficult to determine what need to be done, how it will be done & when the project completes.

→ A good report/plan includes:
- A schedule that identifies when each task should be completed,
- A list of resources needed,
- A list of risk & potential problems,
- An estimation of how much work will be required.

4) Set realistic deadlines:

→ It is always important to set the ~~rela~~ realistic deadlines for the project so that everyone knows how much time is actually they have for completion of the project.

→ Too high deadline can lead to frustration on the part of stakeholders.

→ Too low deadline may not allow enough time to achieve the goal.

5) Communicate effectively with stakeholders.

→ It is important to have clear and concise interaction with the stakeholders so that everyone understands what is required and can work to achieve the project's goal.

6) Analyze the project's goal /challenges & opportunities

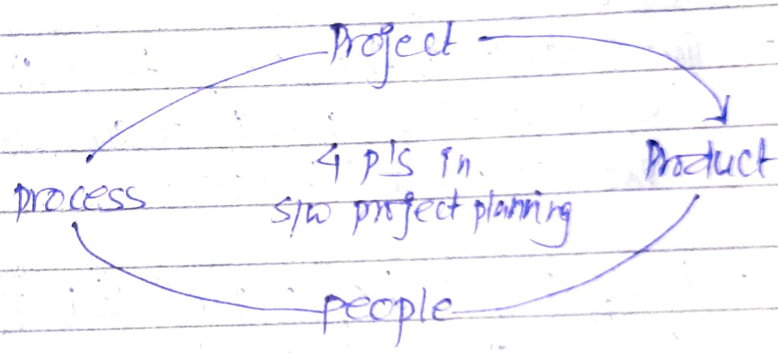
→ When working on a project, the challenges that might occur in the development phases must be closely analyzed and the measure to overcome them must be defined.

→ Also, grab every possible opportunity as possible that is profitable /beneficial for the project.

4 p's of software development:

→ There are few components while software project planning:

(i) Product
(ii) Process
(iii) People
(iv) Project



(i) Product:

- This the the end result (i.e. deliverable) of the project.
- The project manager must clearly define the product scope to ensure the successful delivery of the product.

(ii) Process:

- In every s/w planning, a clearly define process is the key for successful product development.
- It regulates how the development process is carried out within the specified deadline.
- It has several steps involved: documentation, implementation, deployment and interaction phase.

(iii) People:

- They are the human resources having different set of skills involved on the project for the successful product development.

- They contribute their efforts in exchange of financial benefits.

- Stakeholders, employee, IT professionals, team leaders, etc are some assigned roles.

(iv) Project:

- The project is the set of task that needs to be completed within a specified deadline and estimated budget.

- Every project has a project manager who plays critical role on successful development of product.

S/w project management:

↳ It is the process of managing, allocating and timing resources to develop the software product successfully within the budget and schedule.

↳ During the development process, the developer & end users must have knowledge about the length, cost and complexity of the project.

↳ It provides sub-discipline for s/w mgmt which allows s/w planning, monitoring, controlling & implementing of ideas.

↳ Tasks involved:

1) Problem Identification,
2) Problem definition,
3) Project planning,
4) Project organization
5) Resource Allocation,
6) Project scheduling,
7) Tracking, Reporting & Controlling,
8) Project Transmission.

↳ Objectives:

1) Used for project planning, scheduling, resource allocation
2) Provides better flexibility for change mgmt.
3) It keeps time and efforts optimized and keeps the project on track.
4) It ensures the product is developed within time and budget

Conventional s/w management:

↳ Conventional s/w mgmt practices are sound in theory about practically it is still tied to old tools and technologies.

↳ The best thing about a s/w is its flexibility. i.e. It can be programmed to do almost anything.

↳ And, the worst thing about a s/w is also its flexibility.
i.e. "Almost anything" characteristics of the s/w makes it.

Waterfall model (In theory):

↳ Waterfall model is baseline for most of the s/w development process and is conventional s/w development framework.

↳ Three primary points were made:
① The two essential process for development of computer programs are:

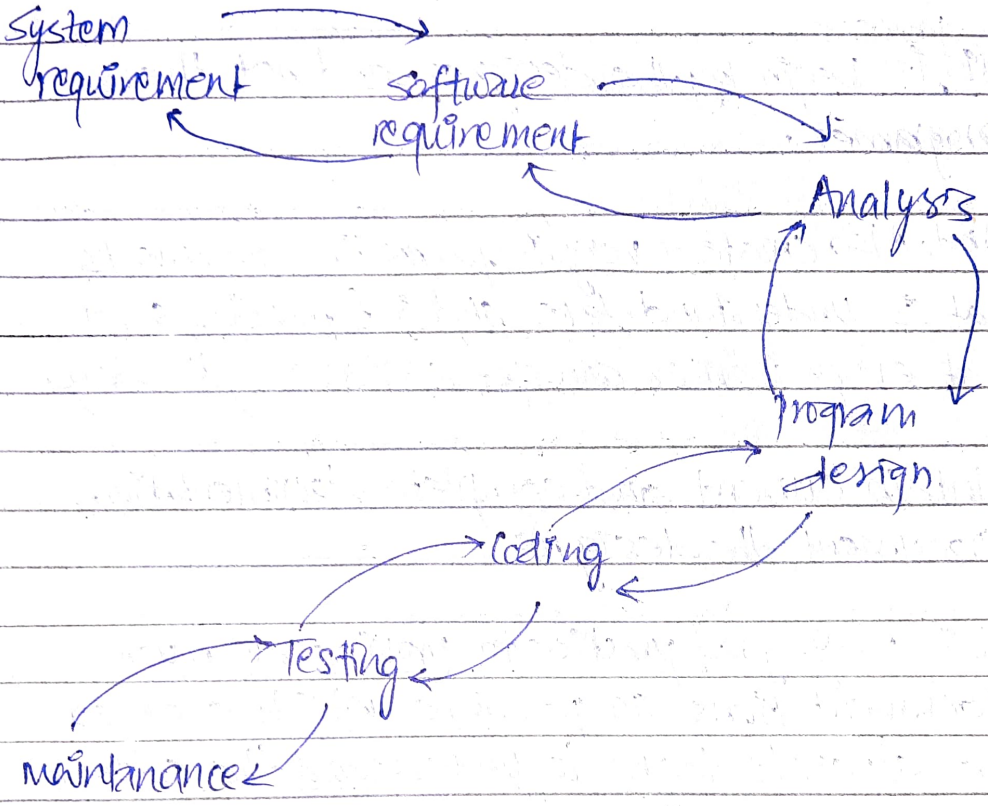
(Analysis) → (Coding)

- Analysis & coding both involves creative work that directly contributes to the usefulness of end product.

fig: Two basic building blocks for program development.

(2) In order to manage and control the intellectual developme of the software following steps must be followed:
- system requirement definition.
- ~~program~~ s/w requirement design.
- Program design & testing.

- These three processes are added to the analysis & design.



(3) Since, Testing is done at the end of the cycle, this might cause the system failure and also invites risks.

↳ The 5 improvements to the waterfall model are:

(1) Complete program design before analysis and coding:

↳ It means that program designs must be done before coding & implementation.

↳ By this technique, the program designer assures that the software will not fail in any circumstances.

↳ This process is begin by the designer and not the programer.

↳ And, Also create a very informative documents that is understandable and informative such that every worker can get elementary knowledge.

(2) Maintain current and complete documentation (Document the design):

→ It is very good practice to provide as much documentations as possible which enables the seperate team to understand the software design through the docs and make modifications.

(3) Do the job twice, if possible (Do it twice):

→ If the program is developed for the first time, it has less features & functionalities

→ So, the final delivered product is the 2nd version of the product.

→ Most modern iterative development follows "Do it N times" approach

④ Plan, control and monitor testing:

→ While developing a s/w product, testing plays a very critical role and has greatest risk in terms of cost and time.

→ Proper testing is done as:

(i) Execute the testing as early as possible,
(ii) Employ test specialist who were not involved in development.
(iii) Employ visual inspection to find the visible error spots (like: wrong addressing, syntax).
(iv) Test every logical parts.
(v) Finally test on the target machine.

⑤ Involve the customer:

→ Involvement of the customer/client is very important and provide early feedbacks that can make s/w development more efficient.

→ steps:

- Acceptance testing at the end of development,
- Preliminary testing of design.
- Critical s/w review during program design.

Waterfall model (In practice):

↳ Most of the s/w projects still follows the conventional s/w management approach.

↳ And, these projects might face the following symptoms: ~~and~~

(1) Protracted (Delayed) Integration:

- In conventional model, the entire system was designed on the paper and ~~every~~ implemented at once and then integrated.

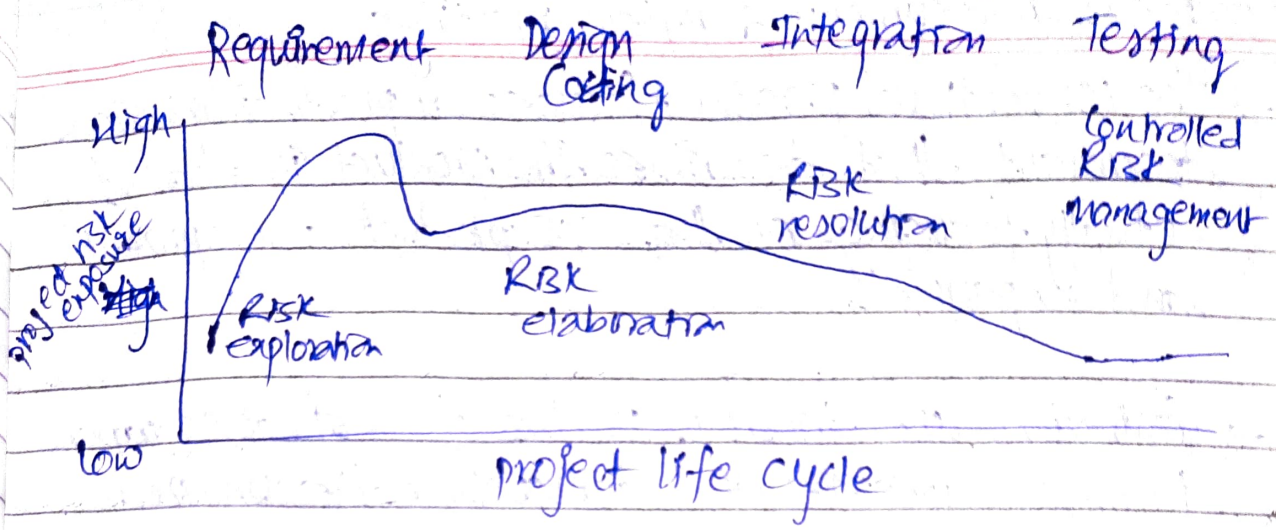
- This process makes it possible to test the product at the end after development only.

- So, it causes testing to consume 40% of the resource while development.

Activity	Cost
Management	5%
Requirements	5%
Design	10%
Code & unit testing	25%
Integration & testing	40%
Deployment	5%
Environment	5%

(2) Late risk resolution:

- In this model, the risk is resolved later.
- Early during the life cycle, when the requirements are specified, the actual risk exposure are highly unpredictable.



(3) Requirements-Driven Functional Decomposition:

- Here, the complete requirements are specified unambiguously before other development activities.
- It treats all requirements equally important and assume those requirements to be constant over the s/w development life cycle. (Rare condition).

(4) Adversarial Stakeholder relationships:

- The following sequence of events are performed on most contractual s/w efforts:

1) The contractor prepares a documentation that contains the progress of the project and intermediate artifacts.
2) The ~~customer~~ contractor expects the response from the client ~~customer~~ (typically within 15-30 days).
3) The contractor then incorporate the clients/customer comments and submit the new docs within the 15-30 days.

(5) Focus on Documents & review meetings:

→ The conventional process focuses on producing various documents that describes the product instead of actually focusing on the s/w product development and expansion.

→ Contractors are driven to produce tons of docs paper to meet milestone and demonstrate the progress to the customer.

→ Frequent review meetings results in low engineering value and high cost in terms of effort and schedule involved in preparation and conduct.

Conventional s/w mgmt performance: [Book pg 18]

1) Every s/w development schedule can be compress only upto 25% of actual schedule.

2) Finding & fixing s/w problems after development cost 100 times more than the early dev stage.

3) Every 1% you spend on development, you will spend 2% in maintainance.

4) S/w development and maintance cost are primarily the f^2 of no of source line of code.

5) Variations among people account for the biggest difference in s/w productivity.

→ Hiring wrong people for the job can cause multiple failure in the s/w development.

6) The overall ratio of software & hardware cost is still growing.
→ In 1955, it was 15:85.
→ In 1985, it was 85:15.

7) Only about 15% of the s/w development effort is given to coding / Implementation:
→ Coding is not only the important activity, beside this there are other important activities in s/w development which involves around 85% of efforts.

8) Walkthrough, ~~caught~~ catches 60% of the errors:
→ Walkthrough may not catch the errors that matters and also not catching them in early stage.

→ It might catch errors and defects like: design issue, styling problems, syntax errors and logical errors.

→ But, the other level defects like: control conflict, performance bottleneck, etc cannot be caught.

9) 80% of contribution comes from 20% of the contributors:
→ This statement is true across all the s/w discipline.
→ Like:
> 80% of error comes from 20% of components.
> 80% of progress is made by 20% of people.
> 80% of rework is caused by 20% of component.
> 80% of resource is caused by 20% of components

Software Economics & cost Estimation:

→ Software Economics is the study of how project resources can be allocated for the development of software in an effective manner.

→ It have 5 basic parameters:

1) Size 2) Process 3) Personnel 4) Environment 5) Quality.

(1) Size:

→ The size of the end product is generally measured in terms of the source code (line of code) or number of functional instructions.

(2) Process:

→ The process is the disciplined regulations applied for the completion of the software development.

→ The process must have the ability to avoid the unwanted / unnecessary activities that might create problem while developing the s/w. (i.e. Rework, comm² overhead etc)

(3) Personnel:

→ They are the individual having different set of skills sets for developing the s/w product.

→ They use their experience with the computer science & appl² domain issues of the project.

(4) Environment:

→ It is made up of tools & techniques available to support efficient s/w development and to automate the development process.

⑤ Quality:

→ It is the quality of the product that is determined by various factors: reliability, performance & adaptability.

↳ The relationship between these parameters is given by:

$$\text{Effort} = (\text{size}^{\text{Process}})(\text{personnel})(\text{Environment})(\text{Quality})$$

[Book pg 22]

Three generations of software development:

1) Conventional development (1960s & 1970s):

→ During this generation, organizations used various custom tools, custom processes all all components of custome that are built using primitive language.

→ The size is 100% custom.

→ It was considered bad due to following reasons:

- It was ~~already~~ always costly and over budget & schedule.
- It also doesnot fulfill the requirements that are necessary such as components, symbolic languages like fortran, PL1 etc.
- The quality of performance was always poor and less than great.

2) Transition (1980's & 1990s):

→ During the generation, the repeatable process were used by the organizations.

→ They used off-the shelf tools and more likely to use custom components that are developed using the high level language.

→ The size is 30% component-based and 70% is custom.

→ This project performance is generally unpredictable and (decision about good or bad) is

→ Transition development is infrequently on budget & schedule.

→ Some of commercial components were available. i.e. GUI, database and networking with OS. But, due to their complexity it was not enough for desired business performance.

3) Modern (2000's & later):

→ Modern practices, processes are generally managed & measured.

→ They use off-the shelf and integrated automated environment / tools.

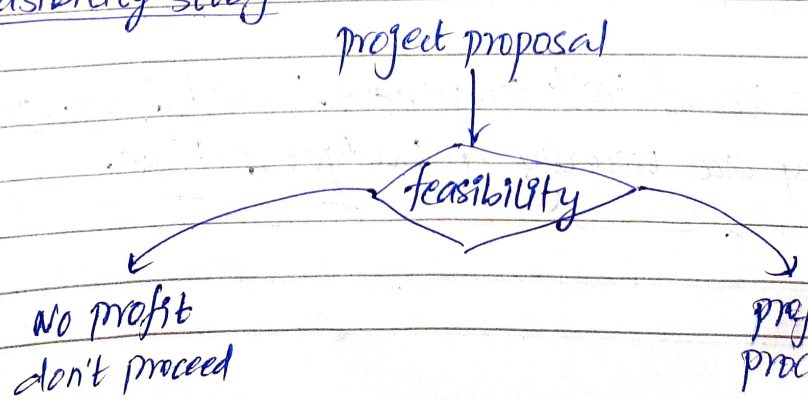
→ The size of 70% component-based & 30% custom.

→ They are usually on budget & time.

Conventional	Transition	Modern
- 1960s - 1970s - waterfall model - Functional design - Diseconomy of Scale	- 1980s - 2000s - Process Improvement - Encapsulation-based - Diseconomy of Scale	- 2000s - later - Iterative dev - Component bas - Return on Investm
Environments/tools: - custom	Env/tools: - off-the-shelf - seperate	Env/tools: - off-the-shelf - Integrated
Size: 100% custom	Size: 30% component 70% custom	Size: 30% custom 70% component based
Process Ad hoc	Process repeatable	Process Managed/ measured
Performance: - predictably bad - Always: • Over budget • over schedule	- Unpredictable - Infrequently • on budget • On schedule	- Predictable - Usually: • on budget • on schedule

* Importance of software economics:

1) feasibility study:



Software cost estimation:

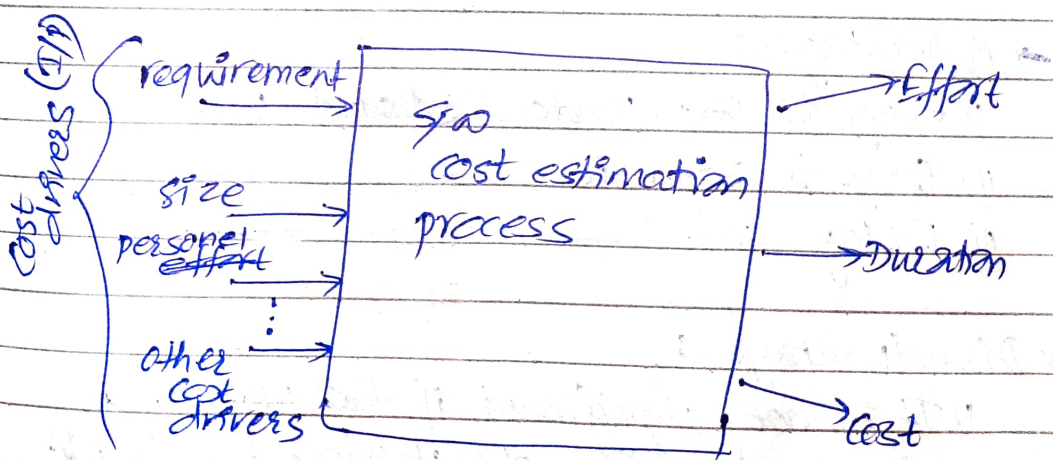
2) Return on investment (ROI)
3) Cost estimation, effort estimation & quality estimation.
4) Help project manager to allocate resources in most efficient way.

Software Cost Estimation:

↳ The cost of development are primarily the costs of the effort involved, so the effort computation is used in both cost and schedule estimate.

↳ The initial cost estimate may be used to determine the budget of the project and set a price for the software for the customers.

↳ Thus, S/w cost estimation is the process of determing the cost of the software development with the set of inputs which is processed & o/p is generated.



→ There is always debate among developers and vendors of s/w cost estimation model & tools.
They are:
i) Which cost estimation model to use,
ii) Whether to measure s/w size in source LOC or in functional metrices/points.
iii) What constitute to a good estimate.

→ There are two ways for measuring s/w:
i) Source Line of Code (SLOC)
ii) Number of function points.

i) SLOC:
→ SLOC metric is any line of text in a code that is not comments and blank lines.
→ A SLOC contains of all lines containing program header file, variable declaration and executable or non executable codes.

→ It can only be utilized to compare or estimate projects that utilize the same language and that are programmed using the same coding standard.

→ Advantages:
i) Easy to understand and use.
ii) Easy to automate
iii) Widely used.

→ Disadvantages:
i) It is not significant if the program is developed using object oriented approach.

ii) Cross project and cross organization comparision is not appropriate using SLOC.

iii) Depends on developers psychology & experience

iv) Only utilized if the project utilizes the same language and are programmed using same coding standard.

(ii) Functional point Metrices:

↳ It is an element of software development which helps to approximate the cost of development early in the process.

↳ It may measure the functionality from user's point of view.

↳ Advantages:

1) It is language independent.
2) Works on comparision between cross project and cross organization.
3) Donot depend on developer's experience and psychology.
4) It works well than SLOC since, it provides metrices for program developed by using components, autogenerated code and OOP etc.,

↳ Disadvantage:

1) Difficult to automate,
2) Not easily understandable compared to SLOC.
3) Primitive defination are abstract and measurement are not easily derived directly from the evolving artifacts.

SLOC	FP
1) In this metrics, the Line of Source code is counted except for blank Line & comments.	1) In this metrices, the functional approximation of the project is done from user's point of view
2) It is based on analogy	2) It is specification based.
3) It is language dependent	3) It is language independent
4) Cannot provide approximation for cross project and cross organization.	4) Provides approximation for cross project & cross organization.
5) Easy to understand & use metrice	5) Not easy to understand & use metrices.
6) Provides better & precise estimation at the later phase of development.	6) Provides better & precise estimation at the early stage of development.
7) It is design-oriented.	7) It is user-oriented
8) It is used for calculating the size of computer program.	8) It is used for data processing system.
9) It can be used to calculate & compare the productivity of the programmers.	9) It can be used to ~~portray~~ portray the project time.

→ Staffing principles:

(a) The principle of Toptalent:

→ Use better and fewer people.

→ The quality of people involved in project is important so it is better to use less people with more skills.

Team size

→ Being grossly over or under the size is bad for team dynamics because it results in too little or too much pressure on individuals to perform.

(b) The principle of job matching:

→ fit the tasks to the skills and motivation of the people available.

→ The skill set of each people involved in the project is different.

→ It means that the best programmer may not be a suitable choice for the system architect and manager & vice versa.

(c) The principle of career progression:

→ An organization does best in the long run by helping its people to self-actualize.

→ Organizations training programs having educational value and project training programs are helpful for the ~~organization~~ individuals to develop their career.

→ This will be very benefitial for the average & below average individuals in learning & self actualization.

(d) The principle of team balance:

→ Select people who will complement and harmonize with one another.

→ The psychology of team member will balance the team and improves the team efficiency.

→ Friendly environment between the team members.

(e) The principle of phaseout:

→ Keeping a misfit on the team doesn't benefit anyone.

→ A misfit gives you a reason to find a better person or to live with fewer people.

→ A misfit demotivates other team members and also will destroy the team balance.

Round Trip Engineering (RTE):

↳ It is a functionality of software development tool that synchronizes two or more software artifacts (related to each other) such as, source code, models, configuration files and even documentations.

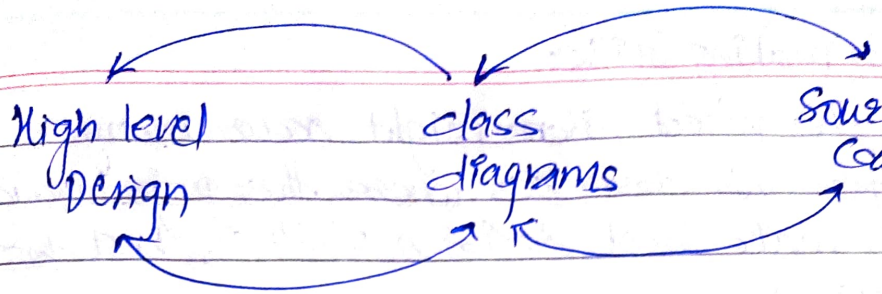
↳ The need for round trip engineering arises when same information is to be presented in multiple artifacts and they need to be consistent.

↳ Example: If any changes are done to the source code, then as a result it becomes missing in the corresponding UML diagram.

↳ The key characteristic that makes RTE different from forward & reverse engineering is the ability to synchronize the existing artifact that evolved concurrently by incrementally updating each artifacts to reflects changes made to the other artifacts.

↳ Another characteristic is the capability to automatically update the artifacts in response of automatically detecting the changes.

↳ RTE based on UML need 3 basic components of s/w development:
- > Source code Editor,
- > UML Editor,
- > UML structure visualization.



Synchronization of artifacts.

S/w project manager:

↳ The software project manager is the person who is responsible for planning, scheduling, budgeting, executing and successfully delivering the software to the client.

↳ They require many leadership quality in order to enhance the team effectiveness and produce high quality product.

↳ The attributes of project manager are:

(i) Hiring skills:

→ Choosing best person with adequate skill for a particular task is very important.

→ Hiring person who is misfit for the project can cause problem in the team members and demotivates them.

(ii) Customer interface skill:

→ The project manager must attend multiple discussion meetings with the client from requirement gathering to continuous feedbacks and also at the end for acceptance testing.

→ So, the project manager must posses better skills for interacting with the customers.

iii) Decision making skills:

→ During the project, there might occur many changes and obstacles where the project manager must make best decision which is best for the project.

→ The project manager must make decisions wisely such that the project is always on track.

iv) Team building skills:

→ The manager must decide the size and structure of the team.

→ Perfect people must be placed to their corresponding job such that they give their best for the task.

→ It means that, a best programmer may not be the best fit for system architect job and vice versa.

v) Selling skills:

→ The project manager must have skills to convience the client about their product developed.

→ They must be able to sell their product to the customer and users.

Principle of Conventional Software Engineering:

1) Make Quality:
→ Quality of s/w must be measured or quantified and mechanisms put into place to motivate achieving the goal.

2) Large or high quality of software is possible:
→ There are various techniques that can be implemented for achieving higher s/w quality product.

3) Give end-product to customer as early as possible:
→ It is important to understand the customers realistic need and inorder to get their feedback it is necessary to showcase the result to them as early as possible.

4) Determine the problem before writing the requirements:
→ Since, the experienced developer might already have a solution for a problem but before implementing it check for alternative solutions.

5) Evaluate design Alternatives:
→ After requirements are determined, the architecture and algorithm must be examined.

6) Use an appropriate and correct process model:
→ Appropriate process must be choosen based on the projects requirements.

7) Use different language for various phases:
→ Our industry's main goal is to provide simple sol² to complex problems and inorder to accomplish the goal different language can be used for diff. modules.

8) Minimize intellect distance:
→ We have to design the s/w structure which is close to the real world structure if possible.

9) Put techniques before tools:
→ An undisciplined s/w engineer with tool becomes very dangerous and harmful.

10) Get it right before we make it very faster:
→ It is easy to make a working program faster than the fast program to work.

11) Inspect the code:
→ Examining the detailed design and code is much better than testing.

12) Good mgmt is more important than good technology

13) People are key to success:
→ The right people with talent and trainings are key to success.

14) Follow with care:
→ Donot follow anything blindly what others are doing.

15) Take Responsibility:
→ When bridge collapse we ask, "What did the engineer do wrong?"
→ Similarly, we ask the same in terms of s/w.

16) Understand the customer's priority:
→ It is possible the customer would tolerate 90% of the functionalities delivered late if 10% is delivered on time.

17) The more they see, the more they need:
→ It means that the more the client functionality or performance we provide to the client, more functionality or performance they need.

18) Plan to throw one away:
→ Any new application, architecture, interface or algorithms generally work ~~rea~~ rarely for the first time.

19) Design for update and change:
→ We must update or accomadate change architecture, component and specification techniques we generally use.

20) Without documentation design is not design:
→

21) Be realistic while using tools:
→ Tools that s/w used make their clients or users more efficient.

22) Encapsulate:
→ It is simply means to hide the information which makes s/w easy & simpler to test & maintain.

23) Avoid tricks:
→ Most developer uses tricks to perform an op^n. So, avoid these tricks or it can harm the project.

24) Don't test own softwares:
→ S/w developers should not test their own s/w and let outsider testing team to test them.

25) Coupling & cohesion:
→ This is best way to measure s/w inherent maintainability & adaptability.

26) ~~Expect~~ ~~Except~~ for excellence:
→ Most employees will work in a far better way if we have high expectation for them.

Principles of Modern software Management:

1) Architecture - First approach:
2) Iterative life cycle
3) Component based Approach
4) Change management system
5) Round Trip management
6) Model Based Evolution
7) Objective quality control
8) Evolving levels of design
9) Establish a configurable process
10) Demonstration Based Approach.

1) Architecture-first approach:

→ In this approach, the main aim is on building a stronger architecture for our software.

→ All the ambiguities and flaws are identified and resolved during this phase.

→ Also take decision regarding designs such that we can produce high quality and productivity software.

2) Iterative life cycle phase:

→ In this phase, the four steps requirement gathering, design, implementation and testing are repeated again and again until we mitigate the risk factor.

3) Component based Approach:

→ In this approach, we reuse the previously defined functions for the spd development.

→ We reuse a part of code from our codebase as a component.

→ This allows us to avoid the requirement & design process to create a component since we are reusing already created component.

4) Change management system:

→ While developing a software, there might be certain change that is need to be done at any instance of time

→ for this, we must be ready to overcome the change and implemented changes are tested & certified.

5) Round Trip Engineering:

→ It is a functional tool where which allows synchronization between different artifacts containing same informations.

→ Example: If we are creating a class diagram based on the source code.
And, later we made few modifications to the code, then RTE helps in synchronizing the class diagram based on the modification.

6) Model Based Evolution:

→ This approach supports the semantically richgraphics & textual design notion.
[i.e. UML]

→ It is used in capturing design artifacts.

7) Objective Control Quality:

→ The objective of quality control is to improve the quality of the software product.

→ It involves Quality management plan, Quality metrices, quality checklists, quality baseline & improvement measures.

8) ~~Evolut~~ Evolving level of design:

→ Here, the designs are evolving and grows as per the ~~proje~~ project evolves.

↳ We have an evolving level of usecase, architecture and details.

9) Establish a configureable process:

→ Establish a configureable process that is economically scalable.

→ One process is not suitable for all the ~~processes~~ developments so we must configure the process which deals with various applications.

10) Demonstration Based approach:

→ In this approach, we mainly focus on demonstration ~~and~~ which helps in increasing the productivity and quality of our software.