SOFTWARE ENGINEERING (SE-207)

DEFINITION

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software and the application of engineering to the development of software in a systematic method is called software engineering.

COMMON ISSUES

- Doesn't fulfil need of customer
- Hard to extend and improve: a programmer leaves a trap hole for the purpose of extension and improvement. They are hidden and only visible to programmer.
- Bad documentation: If not properly documented, others can continue that project.
- Bad quality: Features are not according to the user's requirements.
- More time and cost than expected

SE PROBLEM SOLVING APPROACH

1) Analysis

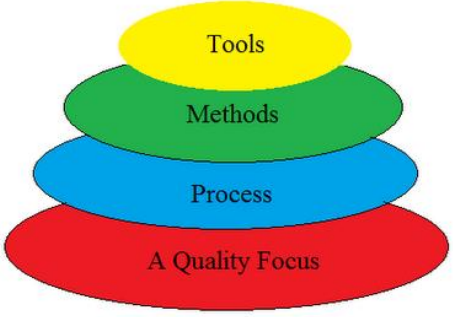
Find the problem, understand it and divide the problem into chunks so that its solution will become easy.

2) Synthesis

Combine the chunks to form larger chunks so as to design/create interfaces and database. In short, the problem is deeply analyzed during synthesis.

SE LAYERED TECHNOLOGY

Software development is totally a layered technology. That means, to develop software one will have to go from one layer to another. The layers are related and each layer demands the fulfillment of the previous layer. Figure below is the upward flowchart of the layers of software development.



- **Quality focus**

Any engineering approach must rest on an quality. Software engineering must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

- **Process models**

SE process is the glue that holds all the technology layers together and enables the timely development of computer software. It forms the base for management control of software project.

- **Methods**

SE methods provide the "Technical Questions" for building Software. Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

- **Tools**

SE tools provide automated or semi-automated support for the "Process" and the "Methods". Tools are integrated so that information created by one tool can be used by another.

GENERIC VIEW

Definition phase

The definition phase focuses on “what”. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. During this, three major tasks will occur in some form:

- System and information engineering
- Software project plan
- Requirement analysis

Development

The development phase focuses on “how”. That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized,

how the design will be translated into a programming language, and how testing will be performed. During this, three specific technical tasks should always occur:

- Software design
- Code generation
- Software testing

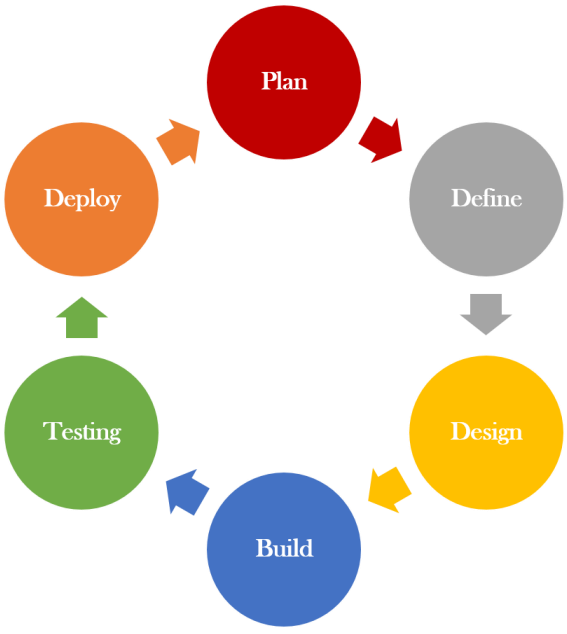
Support phase

The support phase focuses on “change” associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of change are encountered during the support phase:

- Correction of the bugs and errors
- Adaptation: To adapt or embed already existing features in your system
- Enhancement: Keeping in view the future requirements or extensions, the software should be made powerful as to improve the previous features or add some new according to the user needs.
- Prevention: Computer software declines due to change, and because of this, preventive maintenance, often called software re-engineering, must be conducted to enable the software to serve the needs of its end users.

SDLC (Software Development Life Cycle)

SDLC is a conceptual framework to be used as a guide in making a diagnosis, understanding a developmental process and to produce a continuous process. SDLC models are made to manage the level of complexity. SDLC follows traditional approach. They are the simplest models and due to this reason, many developers still follow SDLC models rather than Agile.



- **Plan**

Plan includes requirement gathering and analysis to check how much the idea can be put into action. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. This is basically the brainstorming phase which include cost, budget, estimation etc.

- **Define**

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. This includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand etc. SRS (Software Requirement Specifications) is also developed. The project team analyzes the scope of the project and plans the schedule and resources accordingly

- **Designing**

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design.

- **Building**

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

- **Testing**

Software testing is done by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

- **Deployment**

After successful testing the product is delivered / deployed to the customer for their use. As soon as the product is given to the customers they will first do the beta testing. If any changes are required or if any bugs are caught, then they will report it to the engineering team. Once those changes are made or the bugs are fixed then the final deployment will happen.

PERSPECTIVE PROCESS MODEL

It defines a distinct set of activities, actions, tasks, milestones and work products that are required to engineer high quality software.

The activities may be:

- **Linear**

It is a step by step approach. Activity of one phase is completed before going to the next.

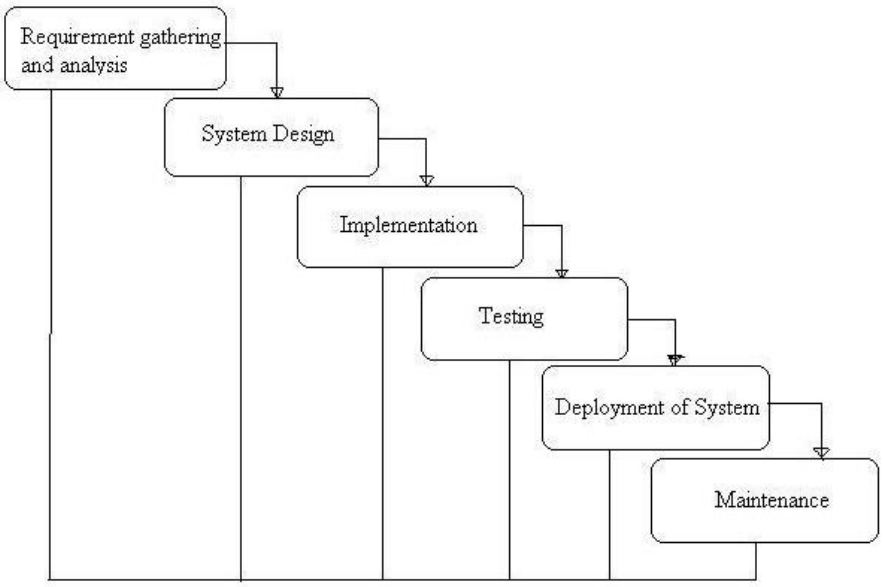
- **Incremental**

The incremental build model is a method of software development where the product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.

- **Evolutionary**

They are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete version of the software.

- **WATERFALL MODEL**



CHARACTERISTICS

- Each phase of development is completed before going to the other.
- Linear, sequential model.
- Overlapping is not allowed, can't execute two phases parallel.
- Output of one phase is the input of the other.

ADVANTAGES

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

DISADVANTAGES

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

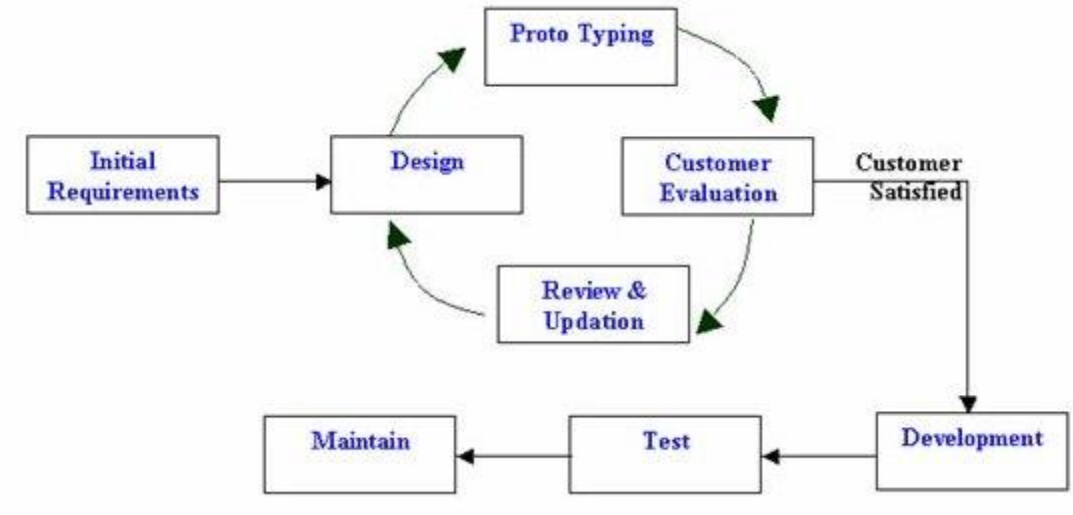
WHEN TO USE

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short.

- **PROTOTYPING MODEL**

Prototype: Prototype is a working model of software with some limited functionality.

The basic idea in **Prototype model** is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system.



Horizontal Prototype:

Horizontal provides a broad view of an entire system focusing on user interaction. Clarify scope and requirements. Demonstrates outer layer of human interface only, such as windows, menus, and screens

Vertical Prototype:

Vertical demonstrate the exact functionality of a small section of the entire product.

ADVANTAGES

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

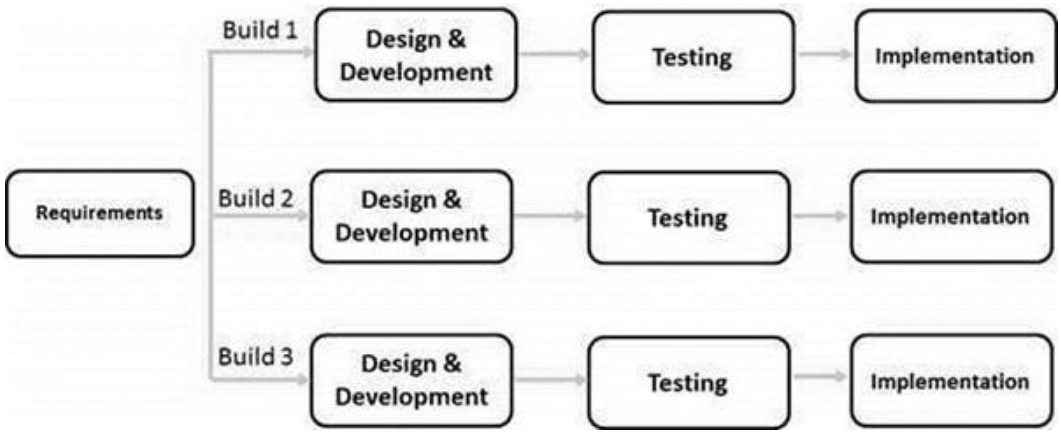
DISADVANTAGES

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

WHEN TO USE

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

▪ ITERATIVE MODEL



CHARACTERISTICS

- It focuses on initial, simplified implementation which then progressively gains more complexity.
- At each iteration, design modifications are made and new functional capabilities are added.
- The basic idea behind this method is to develop a system through repeated cycles and in smaller portion of time.
- Requirements are divided into various builds. During each iteration, the model goes through design, implementation and testing phase until the complete system is ready as per the requirement.

ADVANTAGES

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

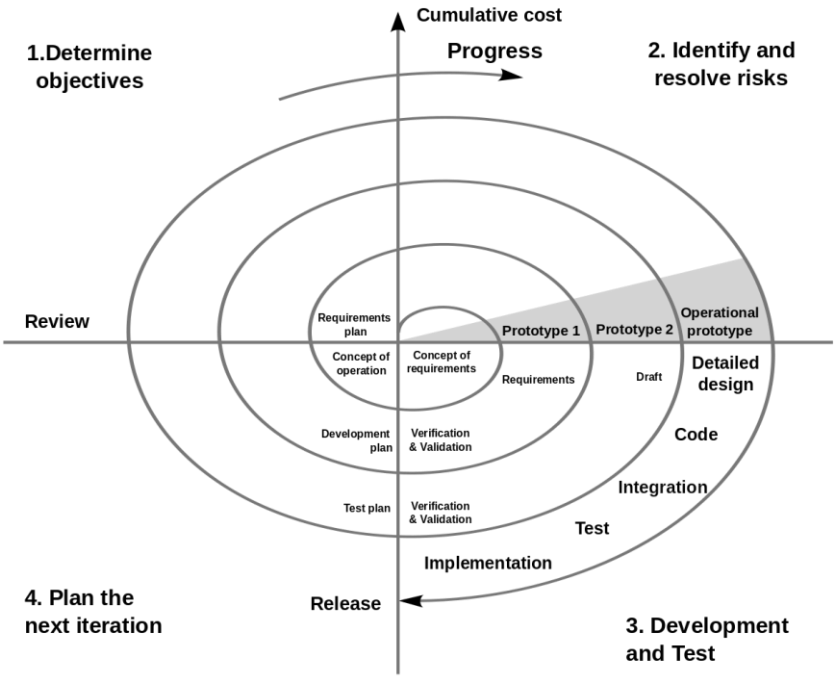
DISADVANTAGES

- Each phase of an iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

WHEN TO USE

- Requirements of the complete system are clearly defined and understood.
- When the project is big.
- Major requirements must be defined; however, some details can evolve with time.

- **SPIRAL MODEL**



Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Customer Communication: if the customer agrees, the project is implemented otherwise it goes to the next spiral.

ADVANTAGES

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

DISADVANTAGES

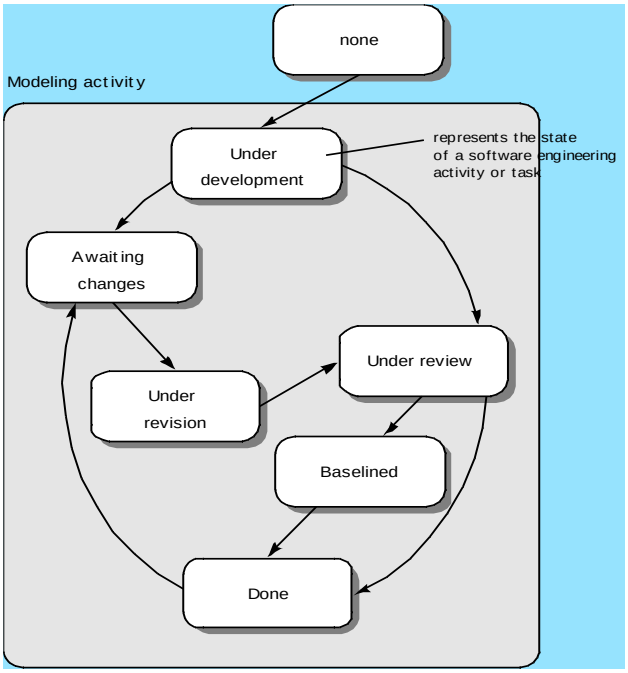
- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

WHEN TO USE

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

- **CONCURRENT DEVELOPMENT MODEL**

CHARACTERISTICS



- The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states.
- The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.
- It is a method of designing and developing products, in which the different stages run simultaneously, rather than consecutively.

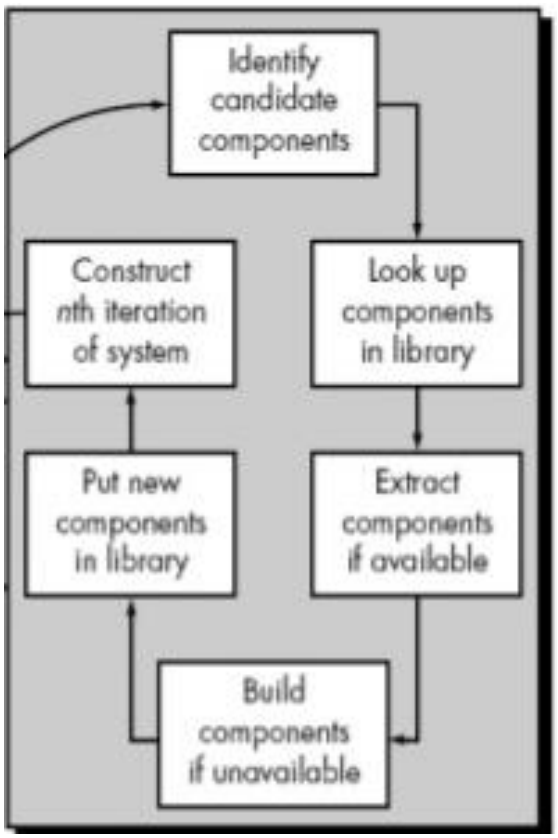
Concurrency can be achieved in two ways:

- System and components activities occur simultaneously and can be modeled using stable-oriented approach.
- A typical client/server application is implemented with many components, each of which can be designed and realized concurrently.

ADVANTAGES

- **Competitive Advantage-** reduction in time to market means that businesses gain an edge over their competitors.
- **Enhanced Productivity-** earlier discoveries of design problems means potential issues can be corrected soon, rather than at a later stage in the development process.
- **Decrease Design and Development Time-** make products which match their customer's needs, in less time and at a reduced cost

▪ COMPONENT BASED DEVELOPMENT



Component Based Software Engineering aims at reducing the cost of software production and improving the quality of a system by building it using selected components and integrating it together into one piece employing well-defined software architecture.

CHARACTERISTICS

The component-based development model incorporates the following steps:

- Available component-based products are researched and evaluated for the application domain in question.
- Component integration issues are considered.
- Software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.

ADVANTAGES

- The components should be designed in a way that enables them to be used in different applications and different scenarios. This saves a lot of cost and is much more productive since time has to be spent only on customizing an already existing component.
- Each component should be able to be replaced by a similar one, thus if slightly different functionality is required or the current component is obsolete or no longer suitable, it can be quickly substituted.
- The components need to be designed to be integrated into different environments and contexts.
- Each of the components can be further adjusted to provide additional functionality.
- The components can interact using only their interfaces, therefore hiding the local details such as state, processes or variables.

AGILE METHODOLIGIES

- **MANIFESTOES**

1) Individuals and interactions over processes and tools
2) Working software over comprehensive documentation
3) Customer collaboration over contract negotiation
4) Responding to a change over following a plan

- **AGILE SOFTWARE DEVELOPMENT PRINCIPLES**

1) Customer's satisfaction by early and continuous delivery of valuable software
2) Welcome changing requirements even in late development
3) Working software is delivered frequently in weeks rather than months
4) Close daily cooperation between business people and developers
5) Projects are built around the motivated individuals
6) Face to face conversation is best form of communication
7) Working software is principle measure of progress
8) Sustainable development able to maintain constant phase
9) Continuous attention to technical excellence and good design
10) Simplicity, art of maximizing, amount of work not done in essential
11) Best architecture; requirements and design emerge from self-organizing teams
12) Regularly teams reflect on how to become more effective and adjust accordingly

- **AGILE SOFTWARE DEVELOPMENT METHODS**

- **EXTREME PROGRAMMING**

XP ACTIVITIES

✓ Coding

The necessary coding is done using pair-programming technique.

✓ Testing

Testing is done to check errors or bugs in the code when done with the coding.

✓ Listening

The basis of extreme programming is a continuous mechanism of customer involvement through feedback during the development phase. Apart from the customer, the developer also receives feedback from the project manager.

✓ Designing

Designing is done so as to get a clear understanding about what you are coding and testing.

XP PRACTICES

❖ Fine scale feedback

- Pair Programming: two or more people code together
- Planning Game

  Release planning: Requirements are gathered and commitments are done regarding project

  Iteration planning: The developers are involved to plan the activities and tasks for iteration.

- Test Driven Development: unit testing is done; each small chunk is tested at every phase.
- Whole Team: Customer and end-users are involved in tem.

❖ Continuous process

- Continuous Integration: Small chunks are integrated
- Design Improvement: The design is further improved at every release.
- Small Releases: Small releases for frequent feedback; reduces chances of errors.

❖ Shared understanding

- Coding Standards: we follow certain formats so that chunks are not dispersed
- Collective Code Ownership: all people working on the project are owners. Everyone is allowed to make changes in it
- Simple Design: system design should be kept simple. It is easier to understand and refactoring and collective ownership is made possible.
- System Metaphor: the system's structure is kept easier to elaborate.

❖ **Programmer welfare**

- Sustainable Pace: XP emphasizes on the limited no. of hours a week for every member, based on their sustainability.

APPLICATION

- Involve new or prototype technology, where the requirements change rapidly, or some development is required to discover unforeseen implementation problem.
- Are research projects, where the resulting work is not the software product itself, but domain knowledge.
- Are small and more easily managed through informal method.

ADVANTAGES

- Extreme programming methodologies emphasis on customer involvement
- This model helps to establish rational plans and schedules and to get the developers personally committed to their schedules which are surely a big advantage in the XP model
- This model is consistent with most modern development methods so, developers are able to produce quality software

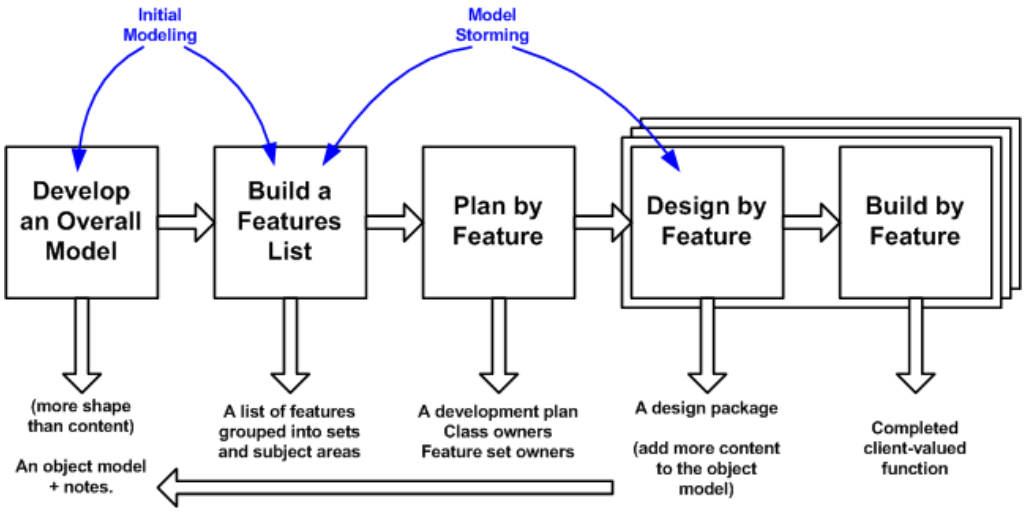
DISADVANTAGES

- This methodology is only as effective as the people involved, Agile does not solve this issue
- This kind of software development model requires meetings at frequent intervals at enormous expense to customers
- It requires too much development changes which are really very difficult to adopt every time for the software developer
- In this methodology, it tends to impossible to be known exact estimates of work effort needed to provide a quote, because at the starting of the project nobody aware about the entire scope and requirements of the project

- **FEATURE DRIVEN DEVELOPMENT**

Feature: It is the functionality which must contain action, result and object. Like on clicking button, whatever happens is a feature.

Feature Driven Development is an iterative software development methodology intended for use by large teams working on a project using object-oriented technology. This type of model is good for organizations that are transitioning from a phase-based approach to an iterative approach, this methodology also known as an FDD methodology.



Developing an overall model

Domain and the development team members work together to create an object model of the domain problem. Their goal is to propose the model for the domain area. They are always under the watchful eye of a Chief Architect who also gives them guidance. After the teams have proposed their own models, one of the proposed models or a merge of models is selected and it becomes the model for that domain area. This helps the team have a clear overview of the entire project.

Building a feature list

Once your team has developed an object model, the next step is to identify the features that are valuable to the client. The features are the building blocks of the project and help the team navigate the processes.

Plan by feature

After the feature list is completed, the next step is to produce the development plan and assign ownership of features (or feature sets) as classes to programmers.

Design by feature

A design package is produced for each feature. A chief programmer selects a small group of features that are to be developed within two weeks. Together with the corresponding class owners, the chief programmer works out detailed sequence diagrams for each feature and refines the overall model. Next, the class and method prologues are written and finally a design inspection is held.

Build by feature

After a successful design inspection for each activity to produce a feature is planned, the class owners develop code for their classes. After unit testing and successful code inspection, the completed feature is promoted to the main build.

PRIMARY ROLES

- Project Manager - the administrative lead for the project
- Chief Architect - owns overall design of the system, responsible for collaboratively facilitating the system design
- Development Manager - manages day-to-day resource assignments and development activities. May be combined with Project Manager or Chief Architect.
- Chief Programmer - experienced developers who lead feature teams and facilitate iterations.
- Class Owner - experienced developers
- Domain Experts - people with deep business knowledge who provide input into tasks the system is required to perform.

ADVANTAGES

- FDD Helps to move larger size projects and obtain repeatable success
- The simple five processes help to bring work done in a short time and easiest manner
- This type of model is built on set standards for software development industry, so it helps easy development and industry recognized best practices

DISADVANTAGES

- Not an ideal methodology for smaller projects so, it is not good for an individual software developer
- High dependency on the main developer means the person should be fully equipped for an act as coordinator, lead designer, and mentor

- No written documentation provided to clients in this methodology so, they are not able to get a proof for their own software

- **LEAN SOFTWARE DEVELOPMENT**

MUDA: any activity in your process that does not add value.

MURA: Any variation leading to unbalanced situations. Unevenness, inconsistent, irregular. Mura exists when workflow is out of balance and workload is inconsistent and not incompliance with the standard.

MURI: Any activity asking unreasonable stress or effort from personnel, material or equipment. OVERBURDEN

LEAN PRINCIPLES

1. Eliminate waste

Unnecessary code, features and functionality are eliminated.

2. Amplify learning

Create solutions to unique customer problems.

3. Decide as late as possible

Delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions

4. Deliver as fast as possible

The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration.

5. Empower the team

Let the working team design their own working procedures.

6. Build integrity in

- Perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, its price and how well it solves problems.
- Conceptual integrity: it means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness.

7. See the whole

"Think big, act small; fail fast, learn quickly"!

You must have a clear picture of your project.

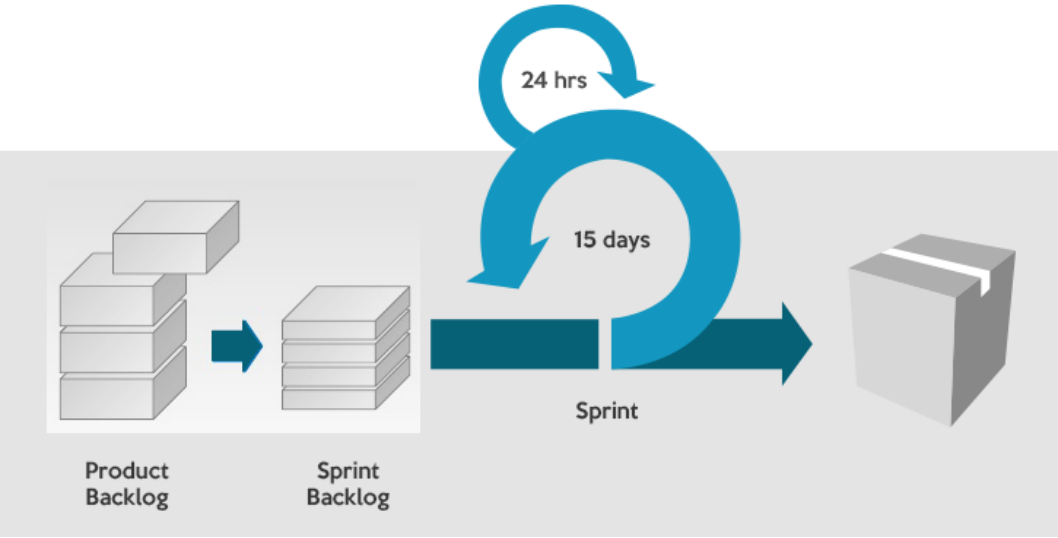
ADVANTAGES

- The early elimination of the overall efficiency of the development process certainly helps to speeds up the process of entire software development which surely reduces the cost of the project
- Delivering the product early is a definite advantage. It means that development team can deliver more functionality in a shorter period of time, hence enabling more projects to be delivered
- Empowerment of the development team helps in developing the decision-making ability of the team members which created more motivation among team members

DISADVANTAGES

- Success in the software development depends on how disciplined the team members are and how to advance their technical skills
- The role of a business analyst is vital to ensure the business requirements documentation is understood properly. If any organization doesn't have a person with the right business analyst then this method may not be useful for them
- In this development model, great flexibility is given to developer which is surely great, but too much of it will quickly lead to a development team who lost focus on its original objectives thus, it hearts the flow of entire project development work

- **SCRUM**



Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. It is used in the following cases:

- Aggressive deadlines
- Complex requirements
- Design of uniqueness

In scrum, project moves forward in a series of iterations called Sprints (2-4 weeks).

SCRUM ROLES

Team: It comprises of 5-9 people.

Product owner: They are the stakeholders who create a prioritized wish list called product backlog.

Scrum master: He is the one who leads the project and keeps the team focused on its goals.

SCRUM EVENTS

- **Product Backlog**

The prioritized wish list created by the product owner in document form.

- **Sprint Planning**

Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

- **Daily Scrum**

Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours.

- **Sprint Review**

Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. There could have been a single deployment or many deployments during a Sprint which lead up to that Increment to be inspected.

- **Sprint Retrospective**

The whole work is presented to the Product owner and is asked for feedback. During the Sprint Retrospective, the team discusses:

  - What went well in the Sprint
  - What could be improved
  - What will we commit to improve in the next Sprint

ADVANTAGES

- In this methodology, decision-making is entirely in the hands of the teams
- This methodology enables project's where the business requirements documentation is not considered very significant for the successful development
- It is a lightly controlled method which totally empathizes on frequent updating of the progress, therefore, project development steps is visible in this method
- A daily meeting easily helps the developer to make it possible to measure individual productivity. This leads to the improvement in the productivity of each of the team members

DISADVANTAGES

- This kind of development model is suffered if the estimating project costs and time will not be accurate
- It is good for small, fast moving projects but not suitable for large size projects
- This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed within exact time frame

SOFTWARE REQUIREMENT ENGINEERING

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

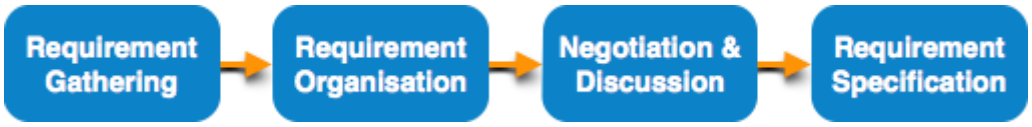
After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities

- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the following diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.
- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.

- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference

and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Types of Software Requirements

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

EXAMPLES -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging

- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have**: Software cannot be said operational without them.
- **Should have**: Enhancing the functionality of software.
- **Could have**: Software can still properly function with these requirements.
- **Wish list**: These requirements do not map to any objectives of software.

While developing software, ‘Must have’ must be implemented, ‘Should have’ is a matter of debate with stakeholders and negation, whereas ‘could have’ and ‘wish list’ can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system cannot be used in convenient way. A system is

said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Explicit Requirements: The Things You Wrote Down

Our first type of requirement is the explicit requirement. This is the simplest type and the easiest to test. Explicit requirements are most commonly found in documents communicated by stakeholders to the development team. They might take the form of an elaborate design specification, a set of acceptance criteria, or a set of wireframes.

Implicit Requirements: The Things Your Customers Will Expect

Implicit requirements are the second type. These are all the things that users are going to expect that were not captured explicitly. Examples include performance, usability, availability, and security. Users *expect* that their password will not be stored in plain text; that requirement need not be written down by anyone.

ANALYSIS MODELING

SOFTWARE TESTING

RISK MANAGEMENT