MANNAR THIRUMALAI NAICKER COLLEGE
(A Linguistic Minority Co-Educational Autonomous Institution)
RE-ACCREDITED WITH"A"GRADE BY NAAC
(Affiliated To Madurai Kamaraj University)
PASUMALAI, MADURAI – 625 004

DEPARTMENT OF INFORMATION TECHNOLOGY

SOFTWARE ENGINNERING

E-CONTENT

Prepared By

K.SANDYA M.Sc.,M.Phil.,SET

SOFTWARE ENGINEERING

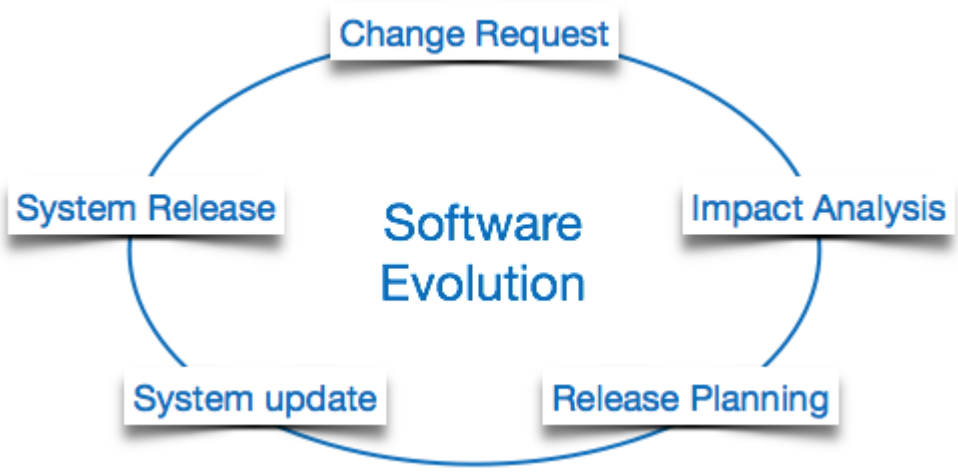
UNIT-1

DEFINITION

Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.

SOFTWARE EVOLUTION

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

SOFTWARE EVOLUTION LAWS

Lehman has given laws for software evolution. He divided the software into three different categories:

- **S-type (static-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
- **P-type (practical-type) -** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- **E-type (embedded-type) -** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

E-TYPE SOFTWARE EVOLUTION

Lehman has given eight laws for E-Type software evolution -

- **Continuing change -** An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful.
- **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
- **Conservation of familiarity -** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth-** In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality -** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems-** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation -** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability -** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

The changing nature of software:

System software

It is collection of programs written to service other programs. Typical programs in this category are compiler, editors, and assemblers. The purpose of the system software is to establish a communication with the hardware.

Application software

It consists of standalone programs that are developed for specific business need. This software may be supported by database systems.

Engineering/scientific software

This software category has a wide range of programs from astronomy to volcanology, from automative stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing. This software is based on complex numeric computations.

Embedded software

This category consists of program that can reside within a product or system. Such software can be used to implement and control features and functions for the end-user and for the systemitself.

Web applications

Web application software consists of various web pages that can be retrieved by a browser. The web pages can be developed using programming languages like JAVA, PERL, CGI, HTML, DHTML.

Artificial intelligence software

This kind of software is based on knowledge based expert systems. Typically, this software is useful in robotics, expert systems, image and voice recognition, artificial neural networks,

SOFTWARE PROCESS FRAMEWORK:

Software Process Framework:

A process framework establishes the foundation for a complete *software process* by identifying a small *number of framework activities* that are applicable to all softwareprojects, regardless of size or complexity. It also includes a set of *umbrella activities* that are applicable across the entire software process. Some most applicableframework activities are described below.

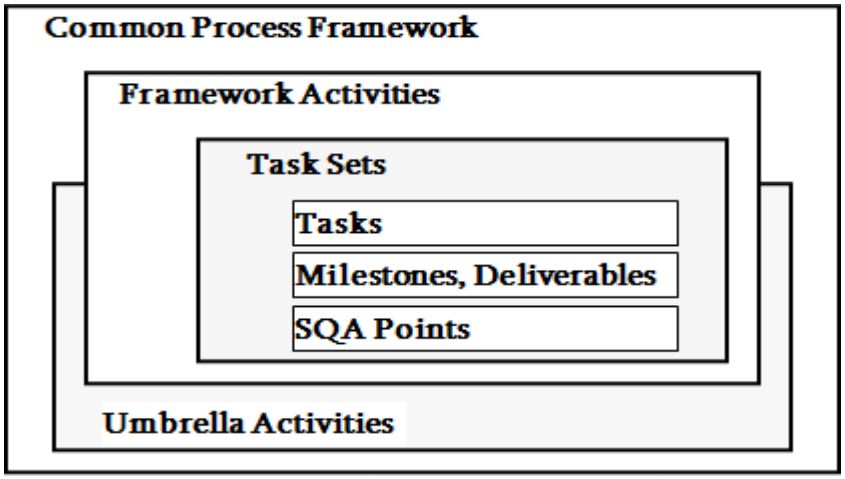


Figure: Chart of Process Framework

Elements of software process:

They are different elements of software process.

Communication:

This activity involves heavy communication with customers andother stakeholders in order to gather requirements and other related activities.

Planning:

Here a plan to be followed will be created which will describe the technicaltasks to be conducted, risks, required resources, work schedule etc.

Modeling:

A model will be created to better understand the requirements and designto achieve these requirements.

Construction:

Here the code will be generated and tested.

Deployment:

Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

Definition:

CMMI is used to evaluate and measure the development of software process in organization.

Objectives of CMMI :

1. Fulfilling customer needs and expectations.
2. Value creation for investors/stockholders.
3. Market growth is increased.
4. Improved quality of products and services.
5. Enhanced reputation in Industry.

.

CMMI Model – Maturity Levels :

In CMMI with staged representation, there are five maturity levels described as follows :

1. **Maturity level 1 : Initial**
   - processes are poorly managed or controlled.
   - unpredictable outcomes of processes involved.
   - ad hoc and chaotic approach used.
   - No KPAs (Key Process Areas) defined.
   - Lowest quality and highest risk.
2. **Maturity level 2 : Managed**
   - requirements are managed.
   - processes are planned and controlled.
   - projects are managed and implemented according to their documented plans.
   - This risk involved is lower than Initial level, but still exists.
   - Quality is better than Initial level.
3. **Maturity level 3 : Defined**
   - processes are well characterized and described using standards, proper procedures, and methods, tools, etc.
   - Medium quality and medium risk involved.
   - Focus is process standardization.
4. **Maturity level 4 : Quantitatively managed**
   - quantitative objectives for process performance and quality are set.
   - quantitative objectives are based on customer requirements, organization needs, etc.
   - process performance measures are analyzed quantitatively.
   - higher quality of processes is achieved.
   - lower risk
5. **Maturity level 5 : Optimizing**
   - continuous improvement in processes and their performance.
   - improvement has to be both incremental and innovative.
   - highest quality of processes.
   - lowest risk in processes and their performance.

CMMI Model – Capability Levels

A capability level includes relevant specific and generic practices for a specific process area that can improve the organization's processes associated with that process area. For CMMI models with continuous representation, there are six capability levels as described below :

1. **Capability level 0 : Incomplete**
   - incomplete process – partially or not performed.
   - one or more specific goals of process area are not met.
   - No generic goals are specified for this level.
   - this capability level is same as maturity level 1.
2. **Capability level 1 : Performed**
   - process performance may not be stable.
   - objectives of quality, cost and schedule may not be met.
   - a capability level 1 process is expected to perform all specific and generic practices for this level.
   - only a start-step for process improvement.
3. **Capability level 2 : Managed**

- process is planned, monitored and controlled.
- managing the process by ensuring that objectives are achieved.
- objectives are both model and other including cost, quality, schedule.
- actively managing processing with the help of metrics.

4. **Capability level 3 : Defined**
   - a defined process is managed and meets the organization's set of guidelines and standards.
   - focus is process standardization.
5. **Capability level 4 : Quantitatively Managed**
   - process is controlled using statistical and quantitative techniques.
   - process performance and quality is understood in statistical terms and metrics.
   - quantitative objectives for process quality and performance are established.
6. **Capability level 5 : Optimizing**
   - focuses on continually improving process performance.
   - performance is improved in both ways – incremental and innovation.
   - emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

Software Process Models
Or
Software Development Life Cycle Model

- It is the processes or methodology being selected for the development of project with an aim and goal.
- It specify the various stages in which a development is carried out.
- There are many factors to select a software process models, like project cost, development team, time duration, objective, perspective etc.

PHASES

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

2) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

5) Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.
In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

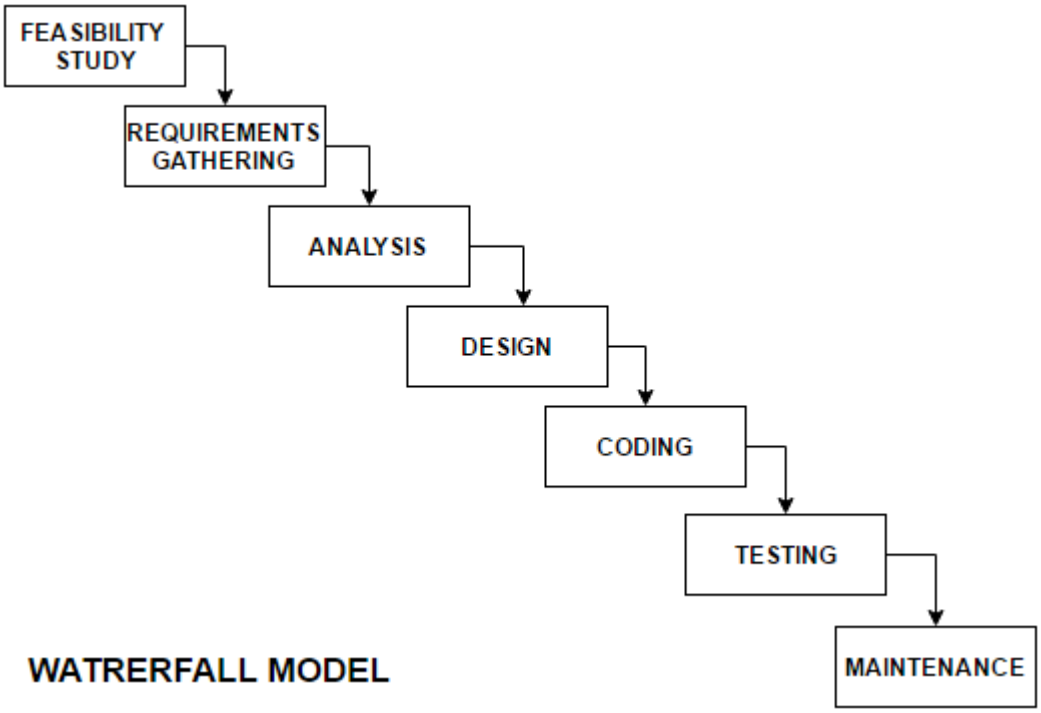
6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

Waterfall Model

- It was the first model introduced.
- It is also known as linear sequential life cycle model.
- It is very simple and understand to use.
- It is basic of all the process development models.
- It is a theoretical model, not to use practically.
- It is called Waterfall because stages goes top to bottom like a natural waterfall.
- When one stage gets completed then only move to next stage.
- Not supposed to come back to previous stage.
- This is considered conventional or classical software life cycle model.

Following is the illustration of Waterfall model:



WATRERFALL MODEL

Advantages of Waterfall Model:

- Simple and easy to understand and use.
- Phases do not overlap.
- Phases are executed one at a time.
- Each phase has specific output and a review process, which makes it easy to manage.
- Best for small projects where requirements are very well understood.

Disadvantages of Waterfall model:

- When model in execution it is very difficult to go back and make changes in previous stage.
- Risk is very high.
- Not suitable for complex and object oriented projects.
- Not suitable for large projects.
- Not suitable for when requirements are not understood, or requirements are not fixed.

When to use Waterfall model:

- When the requirements are fixed, understood.
- Product is understood.
- Technology is understood.
- When the project is short.
- It is costly if project requirements are frequently changing.

Why does the Waterfall model sometimes fail?

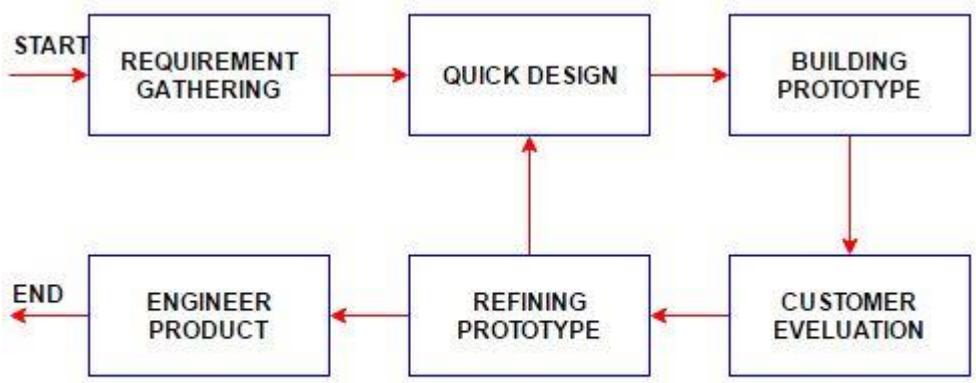
Real projects rarely follows the sequential approach which Waterfall model follows. This creates problem when error occurred and we can't go back to the previous stage to correct it.

PROTOTYPE MODEL

- It requires that before carrying out actual software its prototype (model) must be created.
- Prototype model is a toy implementation of system.

- A prototype usually a demo version of actual system, possibly with limited functionality, low reliability, and inefficient performance compare to actual system.
- Detailed information is not available in it.
- Idea behind it is to create a prototype to gather the basic requirements.
- Prototype is built on the basis of current available requirements.
- It gives the "actual feel" of the system.
- Prototype is not complete system many of the details are not built into the prototype.
- The goal is to provide system with overall functionality.

Following is the illustration of Prototype model:



PROTOTYPE MODEL

Advantages of Prototype Model:

- Working model of system is provided, so user gets a better understanding.
- Errors can be detected and corrected easily.
- User feedback quickly and easily available which leads to a better solution.
- Missing functionality can be easily identified.
- Users are actively involved in development of system.

Disadvantages of Prototype Model:

- Leads to complexity of the system, as scope of the system may expand beyond the original plan,
- Incomplete application sometimes leads to confusion of actual system working.

When to use Prototype Model:

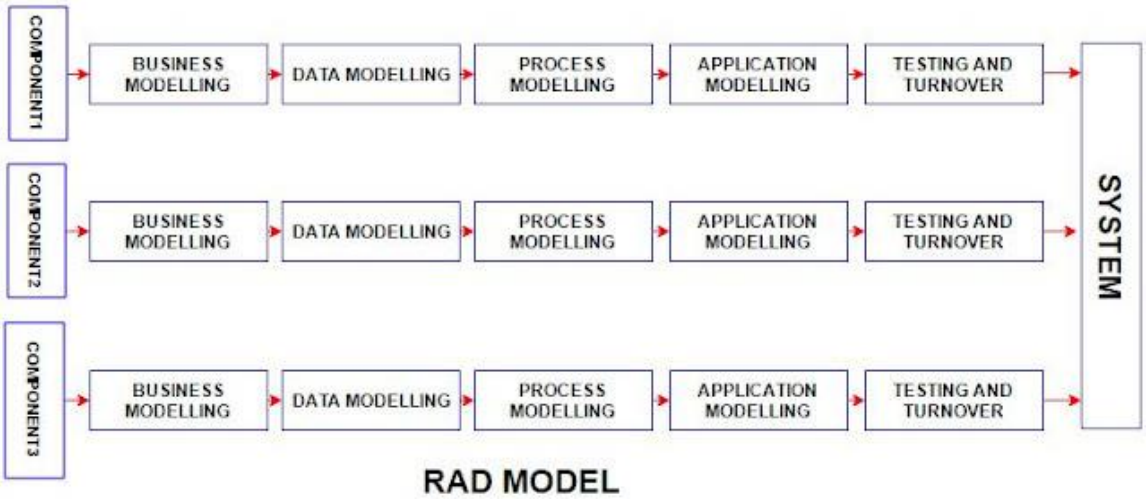
- When lot of interaction with the user is needed.
- In case of online system, web interfaces, where end user interaction is very high. Prototype model is best suited.
- When consistent feedback from user is required.

RAD MODEL

- Rapid Application Development Model.
- It is a type of Incremental Model.
- It is high speed adaptation of Waterfall model.
- In it projects are developed in component as mini projects.
- Than mini projects are assembled in a single project.

- It takes very short period of time to construct a project.
- Customers can give feedback easily on mini projects as well as on complete project.

Following is the illustration of RAD Model:



RAD MODEL

Advantages of RAD Model:

- Increases re-usability of components.
- Less development time.
- Quick initial responses.
- Encourages customer feedback's.
- Involvement of end users from very beginning solves lot of development issues.

Disadvantages of RAD Model:

- Projects which can be developed into mini projects/components can use RAD Model.
- Requires strong and skilled developers team.
- Cost is very high..

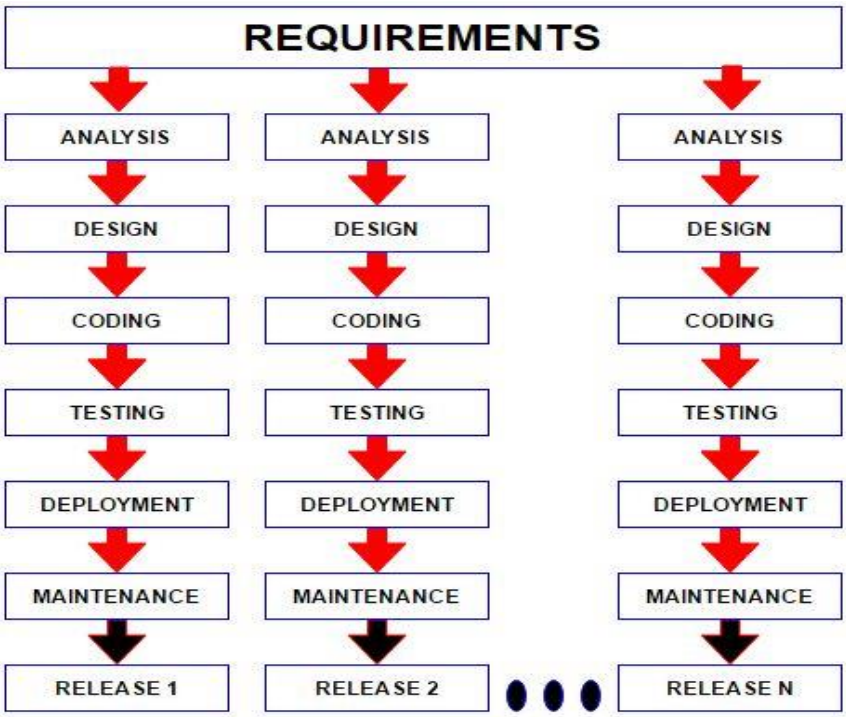
When to use RAD Model:

- When there is need to develop system within short period of time (2-3 months).
- When there is high availability of developers.
- When budget is high enough to afford the development cost.

INCREMENTAL MODEL

- Incremental model as the name indicated produces many versions.
- Each version with a new functionality.
- In it whole requirements is divided into various builds.
- Each new build carry new functionality over previous build.
- It seems like a multi "Waterfall Model cycle".
- In this each module passes through requirements, analysis, design, coding, testing, deployment, maintenance etc.
- A first working software is produced in first release of this model.
- Development of new builds continues until the complete system is released.
- In this whole product got ready step by step.

Following is the illustration of Incremental Model:



Advantages of Incremental Model:

- Generates working software quickly.
- More flexible.
- Less costly.
- Gives time to think new function.
- Able to fulfill all and the new requirements of end users without stopping application use.
- Easier to test and debug during smaller iteration.
- Customer can respond to each build.
- Easier to manage.

Disadvantages of Incremental Model:

- Total cost is higher than Waterfall model.
- Each new build requires with a new functionality.
- Requirements can go beyond the scope decided.
- Needs good planning and design.

When to use Incremental Model:

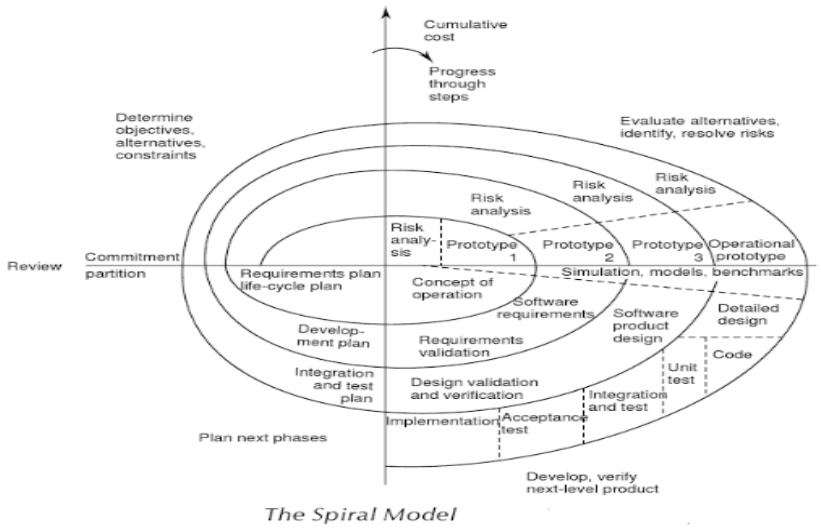
- When a new technology is being used.
- When there are some high risk features and goals.
- When there is a need to get the product early in the market.
- When some requirements can be evolve with time.
- When requirements of the system easily understood and well defined.
-

SPIRAL MODEL

- Spiral model is an evolutionary software model.
- Spiral model may be viewed as a Meta model, because it can accommodate any process model.

- Spiral model focuses on identifying and eliminating high risk problems.

Following is the illustration of Spiral model:



The Spiral Model

- First Quadrant :It determine the objective and alternative solution possible for the phase under consideration.
- Second Quadrant: We evaluate different alternatives based on objective and constraint. To resolve risk.
- Third Quadrant: It emphasises development of strategies to resolve the uncertainties and risks.
- Fourth Quadrant: We determine the objective that should be full filled in next cycle to get complete system.

Characteristics of Spiral Model:

- It is cyclic not linear like Waterfall model.
- Each cycle of Spiral Model consist of four stages.
- Each stage is represented by quadrant of Cartesian Diagram.
- Radius of Spiral represent cost accumulated so far in the process.
- Angular dimension represent progress in process.

Advantages of Spiral Model:

- It is risk driven model.
- It is very flexible.
- Less documentation is needed.
- It uses prototyping.
- It is more realistic model for software development.

Disadvantages of Spiral Model:

- Not suitable for small projects.
- Cost is very high.
- Rely on risk assessment expertise.
- Excellent management skills needed.
- Involvement of different persons makes it complex too.

Limitations of Spiral Model:

- Software development has no strict standard.
- Particular phase has no particular beginning and end.

When to use Spiral Model:

- Spiral model is used when experimenting on technology.
- When trying out new skills.
- When the user is not able to offer requirements in clear terms.
- When system is very complex with lot of functions and facilities.
- When requirements is not clear.
- When the intended solution has multi user, multi functions, multi features, multi locations applications to be used on multiple platforms.

AGILE PROCESS MODEL

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

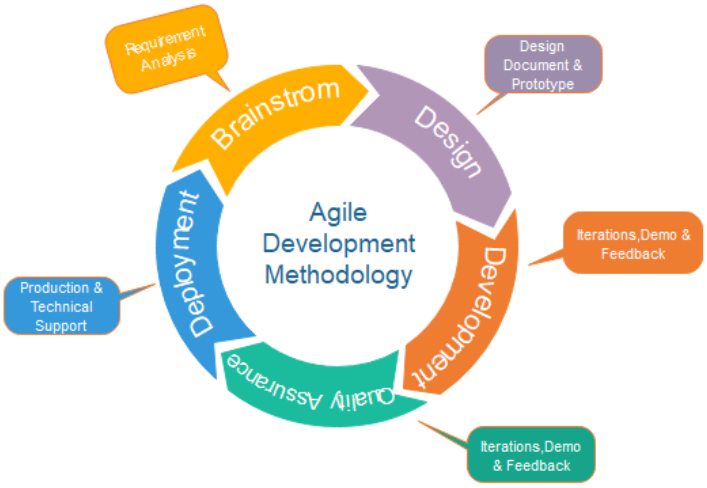


Fig. Agile Model

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

- **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process
- **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
- **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Crystal:

There are three concepts of this method-

1. Chartering: Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.
2. Cyclic delivery: under this, two more cycles consist, these are:
    A. Team updates the release plan.
    B. Integrated product delivers to the users.
3. Wrap up: According to the user environment, this phase performs deployment, post-deployment.

Dynamic Software Development Method(DSDM):

DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions. The techniques used in DSDM are:

1. Time Boxing
2. Moscow Rules
3. Prototyping

The DSDM project contains seven stages:

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

Feature Driven Development(FDD):

This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

Lean Software Development:

Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phases.

1. Eliminating Waste

2. Amplifying learning
3. Defer commitment (deciding as late as possible)
4. Early delivery
5. Empowering the team
6. Building Integrity
7. Optimize the whole

Advantage(Pros) of Agile Method:

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

Disadvantages(Cons) of Agile Model:

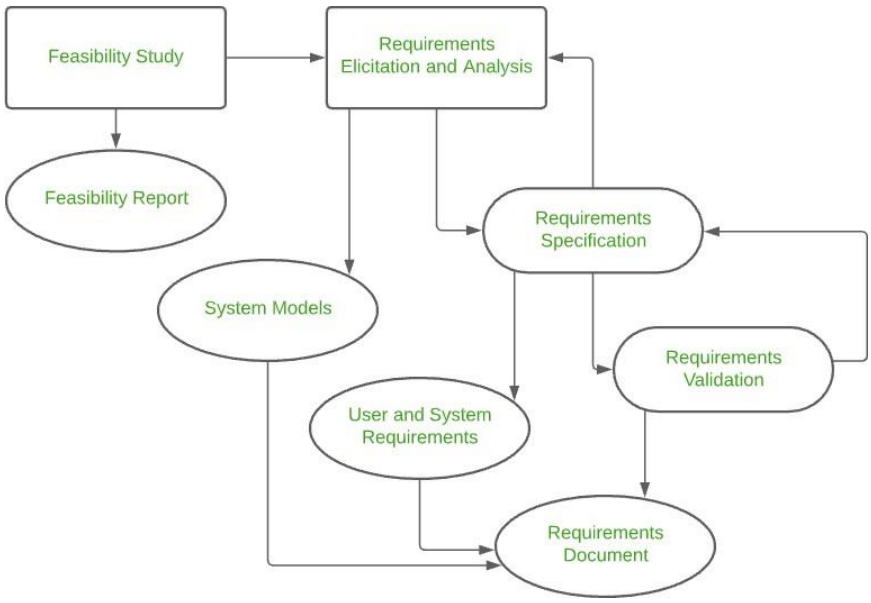
1. Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
2. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

UNIT 2

SOFTWARE REQUIREMENT TASKS

Requirements engineering is a broad domain that focuses on being the connector between modeling, analysis,design, and construction. It is the process that defines, identifies, manages, and develops requirements in a software engineering design process. This process uses tools, methods, and principles to describe the system'sbehavior and the constraints that come along with it.

Requirements engineering is the most important part every business must follow, in order to build and releasea project successfully, as it is the foundation to key planning and implementation.



Requirements Engineering Process

Requirements Engineering Tasks: The software requirements engineering process includes the following steps of activities:

1. **Inception**
2. **Elicitation**
3. **Elaboration**
4. **Negotiation**
5. **Specification**
6. **Validation**
7. **Requirements Management**

Let's discuss each of these steps in detail.

1. Inception: This is the first phase of the requirements analysis process. This phase gives an outline of howto get started on a project. In the inception phase, all the basic questions are asked on how to go about a taskor the steps required to accomplish a task. A basic understanding of the problem is gained and the nature of the solution is addressed. Effective communication is very important in this stage, as this phase is the foundation as to what has to be done further. Overall in the inception phase, the following criteria have to beaddressed by the software engineers:

- Understanding of the problem.

- The people who want a solution.
- Nature of the solution.
- Communication and collaboration between the customer and developer.

2. Elicitation: This is the second phase of the requirements analysis process. This phase focuses on gathering the requirements from the stakeholders. One should be careful in this phase, as the requirements are what establishes the key purpose of a project. Understanding the kind of requirements needed from the customer is very crucial for a developer. In this process, mistakes can happen in regard to, not implementing the right requirements or forgetting a part. The right people must be involved in this phase. The following problems can occur in the elicitation phase:

- **Problem of Scope:** The requirements given are of unnecessary detail, ill-defined, or not possible to implement.
- **Problem of Understanding:** Not having a clear-cut understanding between the developer and customer when putting out the requirements needed. Sometimes the customer might not know what they want or the developer might misunderstand one requirement for another.
- **Problem of Volatility:** Requirements changing over time can cause difficulty in leading a project. It can lead to loss and wastage of resources and time.

3. Elaboration: This is the third phase of the requirements analysis process. This phase is the result of the inception and elicitation phase. In the elaboration process, it takes the requirements that have been stated and gathered in the first two phases and refines them. Expansion and looking into it further are done as well. The main task in this phase is to indulge in modeling activities and develop a prototype that elaborates on the features and constraints using the necessary tools and functions.

4. Negotiation: This is the fourth phase of the requirements analysis process. This phase emphasizes discussion and exchanging conversation on what is needed and what is to be eliminated. In the negotiation phase, negotiation is between the developer and the customer and they dwell on how to go about the project with limited business resources. Customers are asked to prioritize the requirements and make guesstimates on the conflicts that may arise along with it. Risks of all the requirements are taken into consideration and negotiated in a way where the customer and developer are both satisfied with reference to the further implementation. The following are discussed in the negotiation phase:

- Availability of Resources.
- Delivery Time.
- Scope of requirements.
- Project Cost.
- Estimations on development.

5. Specification: This is the fifth phase of the requirements analysis process. This phase specifies the following:

- Written document.
- A set of models.
- A collection of use cases.
- A prototype.

In the specification phase, the requirements engineer gathers all the requirements and develops a working model. This final working product will be the basis of any functions, features or constraints to be observed. The models used in this phase include ER (Entity Relationship) diagrams, DFD (Data Flow Diagram), FDD (Function Decomposition Diagrams), and Data Dictionaries.

A software specification document is submitted to the customer in a language that he/she will understand, to give a glimpse of the working model.

6. Validation: This is the sixth phase of the requirements analysis process. This phase focuses on checking for errors and debugging. In the validation phase, the developer scans the specification document and checks for the following:

- All the requirements have been stated and met correctly
- Errors have been debugged and corrected.

- Work product is built according to the standards.

This requirements validation mechanism is known as the formal technical review. The review team that works together and validates the requirements include software engineers, customers, users, and other stakeholders. Everyone in this team takes part in checking the specification by examining for any errors, missing information, or anything that has to be added or checking for any unrealistic and problematic errors. Some of the validation techniques are the following-

- Requirements reviews/inspections.
- Prototyping.
- Test-case generation.
- Automated consistency analysis.

7. Requirements Management: This is the last phase of the requirements analysis process. Requirements management is a set of activities where the entire team takes part in identifying, controlling, tracking, and establishing the requirements for the successful and smooth implementation of the project.
In this phase, the team is responsible for managing any changes that may occur during the project. New requirements emerge, and it is in this phase, responsibility should be taken to manage and prioritize as to where its position is in the project and how this new change will affect the overall system, and how to address and deal with the change. Based on this phase, the working model will be analyzed carefully and ready to be delivered to the customer.

Initiating the Requirements Engineering Process

The process of initiating engineering is the subject of this section. The point of view described is having all stakeholders (including customers and developers) involved in the creation of the system requirements documents.

The following are steps required to initiate requirements engineering:

1. Identifying the Stakeholders

A stakeholder is anyone who benefits in a direct or indirect way from the system which is being developed.

In the book stakeholders are: operations managers, product managers, marketing people, internal and external customers, end-users, consultants, product engineers, software engineers, support and maintenance engineers, etc.

At inception, the requirements engineer should create a list of people who will contribute input as requirements are elicited.

The initial list will grow as stakeholders are contacted because every stakeholder will be asked”Who else do you think I should talk to?”

2. Recognizing Multiple Viewpoints

Each of the stockholders will contribute to the RE process.

As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.

The requirements engineer is to categorize all stakeholder information including inconsistencies and conflicting requirements in a way that will allow decision makers to choose an internally inconsistent set of requirements for the system.

3. Working toward Collaboration

Collaboration does not necessarily mean that requirements are defined by committee. In many cases, stakeholders collaborate by providing their view of requirements, but a strong "project champion" (business manager, or a senior technologist) may make the final decision about which requirements make the final cut.

4. Asking the First Questions

The first set of context-free questions focuses on the customer and other stakeholders, overall goals, and benefits.

The first questions:

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution that you need?

The next set of questions enables the software engineering team to gain a better understanding of the problem and allows the customer to voice his/her voice:

- How would you characterize "good" output that would be generated by a successful solution?
- What problems will this solution address?
- Can you show me the business environment in which the solution will be used?
- Will special performance issues or constraints affect the way the solution is approached?

The final set of questions focuses on the effectiveness of the communication activity itself. Gause and Weinberg call these questions "meta questions"

- Are you the right person to answer these questions? Are your answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

Eliciting Requirements

1. Collaborative Requirements Gathering

In this approach the stakeholders collaborate under the supervision of a neutral facilitator to determine the problem scope and define an initial set of requirements.

Collaborative Requirements Gathering all apply some variation on the following guidelines:

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is to identify the problem, propose elements of the solution negotiate different approaches, and specify a preliminary set of solution requirements

Read the scenario presented in the book page 153.

2. Quality Function Deployment (QFD)

In this technique the customer is asked to prioritize the requirements based on their relative value to the users of the proposed system.

QFD is a technique that translates the needs of the customer into technical requirements for software engineering. QFD identifies three types of requirements:

Normal requirements: These requirements reflect objectives and goals stated for a product or system during meetings with the customer.

Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them.

Exciting requirements: These requirements reflect features that go beyond the customer's expectations and prove to be very satisfying when present.

In meeting with the customer, *function deployment* is used to determine the value of each function that is required for the system. *Information deployment* identifies both the data objects and events that the systems must consume and produce.

Finally, *task deployment* examines the behavior of the system within the context of its environment. And *value analysis* determines the relative priority of requirements determined during each of the three deployments.

3. User Scenarios

As requirements are gathered an overall version of system functions and features begins to materialize

Developers and users create a set of scenarios that identify a thread of usage for the system to be constructed in order for the software engineering team to understand how functions and features are to be used by end-users.

4. Elicitation Work Products

For most systems, the work product includes:

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.
- A list of requirements (preferably organized by function) and the domain constraints that apply to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements.

Negotiating Requirements

The customer and the developer enter into a process of ***negotiation***, where the customer may be asked to balance functionality, performance, and other product or system characteristics against cost and time to market.

The intent of the negotiations is to develop a project plan that meets the needs of the customer while at the same time reflecting the real-world constraints (time, people, and budget) that have been imposed on the software engineering team.

Boehm defines a set of negotiation activities at the beginning of each software process iteration. The following activities are defined:

- Identify the key stakeholders. These are the people who will be involved in the negotiation.
- Determine each of the stakeholders "win conditions". Win conditions are not always obvious.
- Negotiate; work toward a set of requirements that lead to "win-win"

Validating Requirements

The purpose of requirements validation is to make sure that the customer and developer agree on details of the software requirements (or prototype) before beginning the major design work. This implies that both the customer and developer need to be present during the validation process.

As each element of the analysis model is created, it is examined for consistency, omissions, and ambiguity.

The requirements represented by the model are prioritized by the customer and grouped within requirements packages that will be implemented as software increments and delivered to the customer.

A review of the analysis model addresses the following questions:

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built?
- Has the requirements model been "partitioned" in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

These and other questions should be asked and answered to ensure that the requirements model is an accurate reflection of the customer's needs and that it provides foundation for design.

Requirements Analysis

Requirement Analysis results in the specification of software's operational characteristics; indicates software's interface with other system element; and establishes constraints that software must need. Throughout analysis modeling the software engineer's primary focus is on what not how.

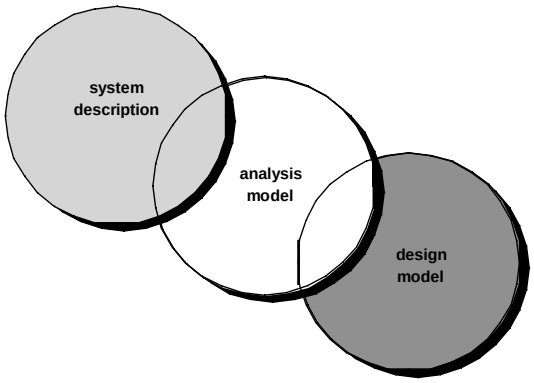
RA allows the software engineer (called an analyst or modeler in this role) to:

- E laborate on basic requirements established during earlier requirement engineeringtasks
- B uild models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

Overall Objectives

The analysis model must achieve three primary objectives:

1. To describe what the customer requires
2. To establish a basis for the creation of a software design, and
3. To define a set of requirements that can be validated once the software is built.



The analysis model bridges the gap between a system-level description that describes overall functionality as it is achieved by ap plying software, hardware, data, human, and other system elements and a software design that describes the software application architecture .

Analysi s Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non -functional models until design.
    - For example, a database may be required, but the classes necessary to implement it, the functions required to access it, and the behavior that will be exhibited as it is used should be considered only after problem domain analysis has been completed.
- Minimize coupling throughout the system.
    - The level of interconnectedness betwee n classes and functions should be reduced to a minimum.
- Be certain that the analysis model prov ides value to all stakeholders.
    - Each constituency has its own use for the model.
- Keep the model as simple as it can be.
    - Ex: Don't add additional diagrams when th ey provide no new information.
    - Only modeling elements that have values should be implemented.

Domain Analysis

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typ ically for reuse on multiple projects within that application domain . . . [Object -oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common object s, classes, subassemblies, and frameworks

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.

- Analyze each application in the sample.

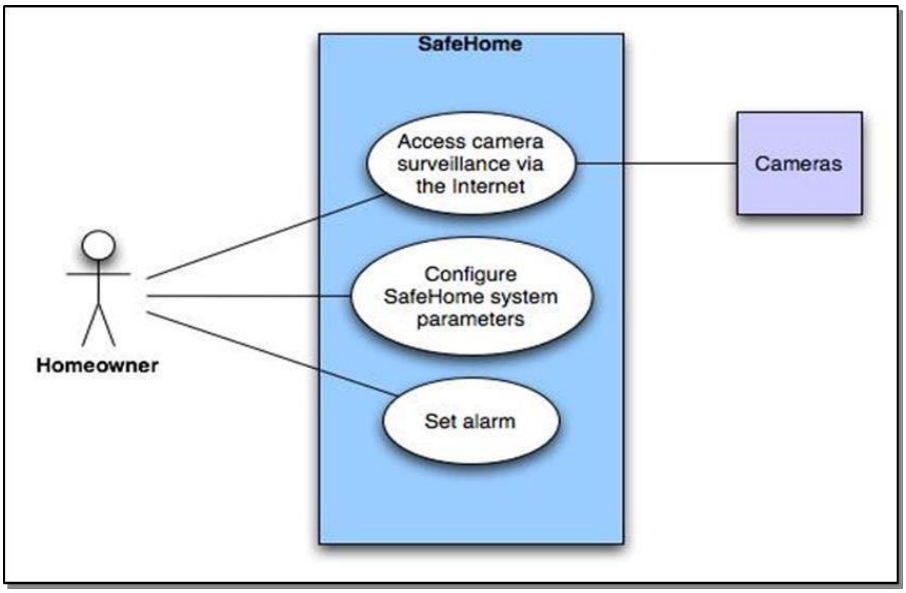
Develop an analysis model for the objects

SCENARIO BASED MODELING

- If the software engineer understand how end users want to interact with a system,the software team will be able to properly charectrize requirements and build meaningful analysis and design models.
- Analysis modeling with UML begins with the creation of scenarios in the form of uses-cases,activity diagram and swimlane diagrams.

Use-case Diagram

- In general, use cases are written first in an informal narrative fashion
- Use case: Access camera surveillance via the Internet—display camera views
- (ACS-DCV)
- Actor: homeowner
- The homeowner logs onto the SafeHome Products website.
- The homeowner enters his or her user ID.
- The homeowner enters two passwords (each at least eight characters in length).
- The system displays all major function buttons.
- The homeowner selects the “surveillance” from the major function buttons.
- The homeowner selects “pick a camera.”
- The system displays the floor plan of the house.
- The homeowner selects a camera icon from the floor plan.
- The homeowner selects the “view” button.
- The system displays a viewing window that is identified by the camera ID.
- The system displays video output within the viewing window at one frame persecond.

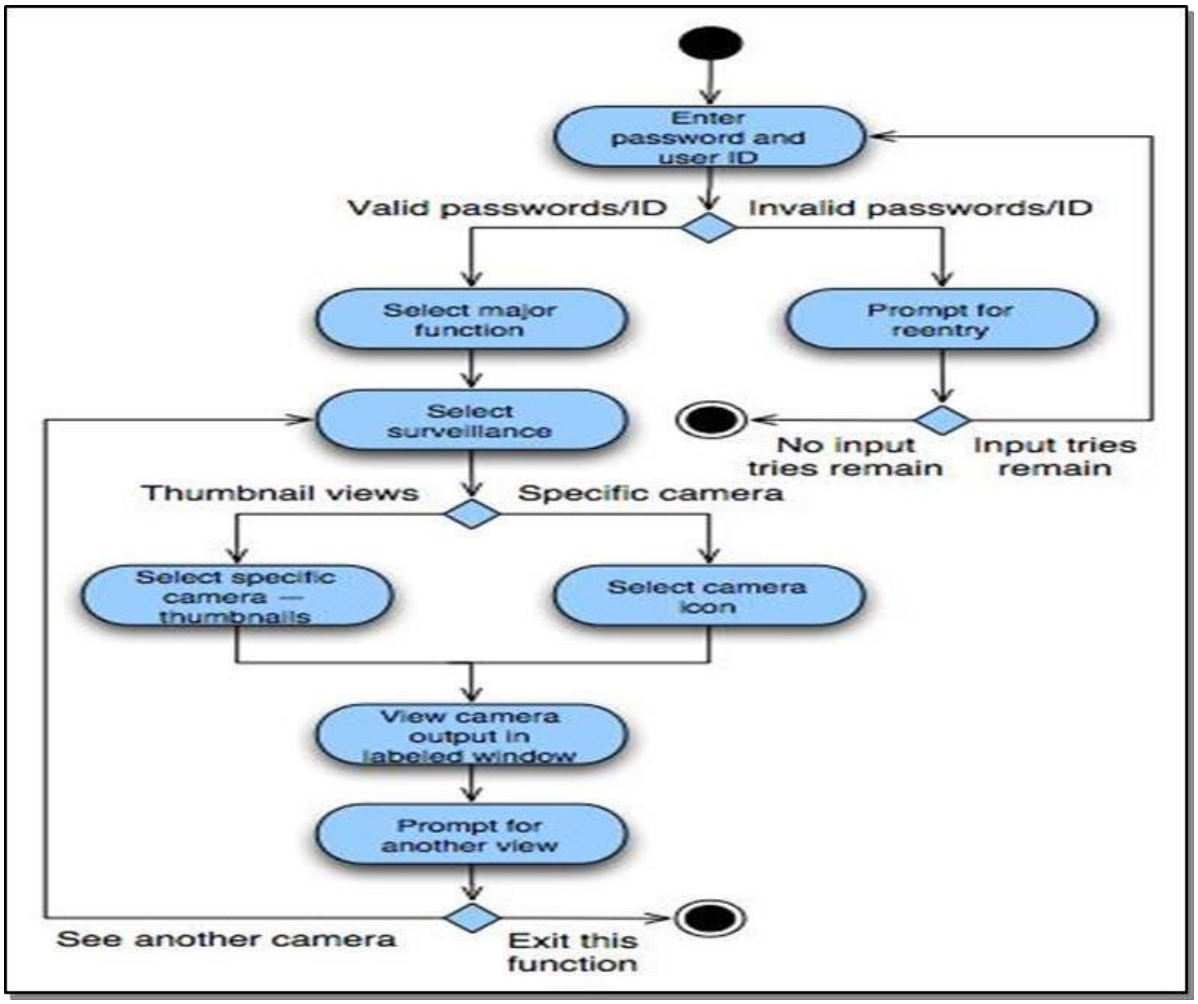


Alternative Actions

- Can the actor take some other action at this point?
- Is it possible that the actor will encounter some error condition at this point?
- Is it possible that the actor will encounter behavior invoked by someevent outside theactor's control?

Activity diagram

- The UML activity diagram supplements the use-case by providing a graphical representation of the flow of interaction within a specific scenario
- Similar to flowchart an activity diagram uses rounded rectangles to imply a specify system functions,arrows to represent flow through the system,decision diamonds to depict a branching decision and solid horizontal lines to indicate that parallel activities are occurring



FLOW ORIENTED MODELING

It shows how data objects are transformed by processing the function.

The Flow oriented elements are:

i. Data flow model

- It is a graphical technique. It is used to represent information flow.
- The data objects are flowing within the software and transformed by processing the elements.
- The data objects are represented by labeled arrows. Transformation are represented by circles called as bubbles.
- DFD shown in a hierarchical fashion. The DFD is split into different levels. It also called as 'context level diagram'.

ii. Control flow model

- Large class applications require a control flow modeling.
- The application creates control information instated of reports or displays.
- The applications process the information in specified time.
- An event is implemented as a boolean value.

For example, the boolean values are true or false, on or off, 1 or 0.

iii. Control Specification

- A short term for control specification is CSPEC.
- It represents the behaviour of the system.
- The state diagram in CSPEC is a sequential specification of the behaviour.
- The state diagram includes states, transitions, events and activities.
- State diagram shows the transition from one state to another state if a particular event has occurred.

iv. Process Specification

- A short term for process specification is PSPEC.
- The process specification is used to describe all flow model processes.
- The content of process specification consists narrative text, Program Design Language(PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.

BEHAVIOURAL MODELS:

Behavioural models are used to describe the overall behaviour of a system. Two types of behavioural model are:

- Data processing models that show how data is processed as it moves through the system;
- State machine models that show the systems response to events.

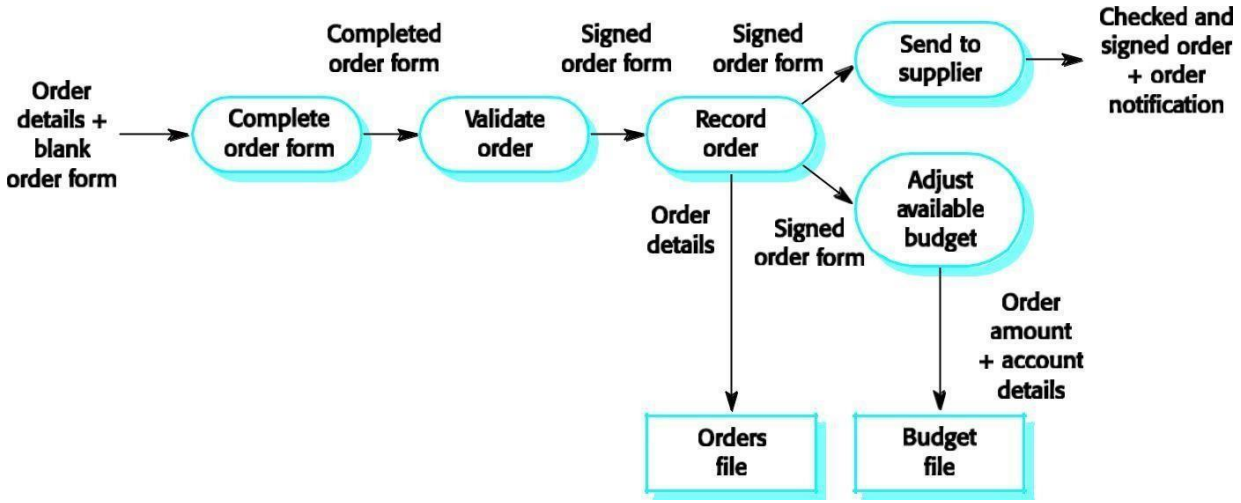
These models show different perspectives so both of them are required to describe thesystem‘s behaviour.

1) Data-processing models:

Data flow diagrams (DFDs) may be used to model the system‘s data processing. These show the processing steps as data flows through a system.

DFDs are an intrinsic part of many analysis methods. Simple and intuitive notation that customers can understand. Show end-to-end processing of data.

Order processing DFD:



Data flow diagrams:

DFDs model the system from a functional perspective.

Tracking and documenting how the data associated with a process is helpful to develop anoverall understanding of the system.

Data flow diagrams may also be used in showing the data exchange between a system andother systems in its environment.

2) State machine models:

These model the behaviour of the system in response to external and internal events.

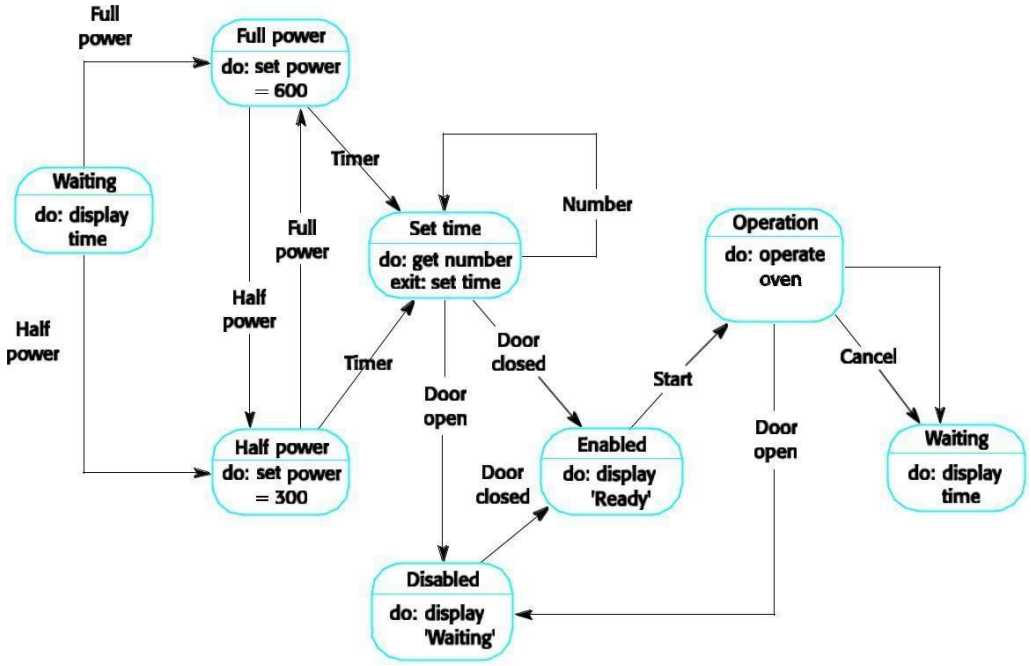
They show the system‘s responses to stimuli so are often used for modelling real-time systems. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.

Statecharts are an integral part of the UML and are used to represent state machine models.

Statecharts:

Allow the decomposition of a model into sub-models (see following slide). A brief description of the actions is included following the _do‘ in each state. Can be complemented by tables describing the states and the stimuli.

Microwave oven model:



Microwave oven state description:

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows _Half power‘.
Full power	The oven power is set to 600 watts. The display shows _Full power‘.
Set time	The cooking time is set to the user‘s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows _Not ready‘.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows _Ready to cook

Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows _Cooking complete' while buzzer is sounding.

Microwave oven stimuli:

Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Number	The user has pressed a numeric key
Door open	The oven door switch is not closed
Door closed	The oven door switch is closed
Start	The user has pressed the start button
Cancel	The user has pressed the cancel button

SEMANTIC DATA MODELS:

Used to describe the logical structure of data processed by the system. An entity-relation-attribute model sets out the entities in the system, the relationshipsbetween these entities and the entity attributes

Widely used in database design. Can readily be implemented using relational databases.No specific notation provided in the UML but objects and associations can be used.

Data dictionaries

Data dictionaries are lists of all of the names used in the system models. Descriptions ofthe entities, relationships and attributes are also included.

Advantages

- Support name management and avoid duplication;

Store of organisational knowledge linking analysis, design andimplementation;Many CASE workbenches support data dictionaries.