UNIT- 1

Introduction to Software and Software Engineering

Software is more than just a program code. A program is an executable code, which serves some computational purpose.

Software is **collection of executable programming code**, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about **developing products**, using **well-defined**, **scientific principles and methods**.

Software engineering is an engineering branch associated with development of **software product using well-defined scientific principles**, **methods** and **procedures**.

The **outcome** of software engineering is an **efficient and reliable** software product.

Definitions

IEEE defines software engineering as:

> **The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software**; that is, the application of engineering to software.

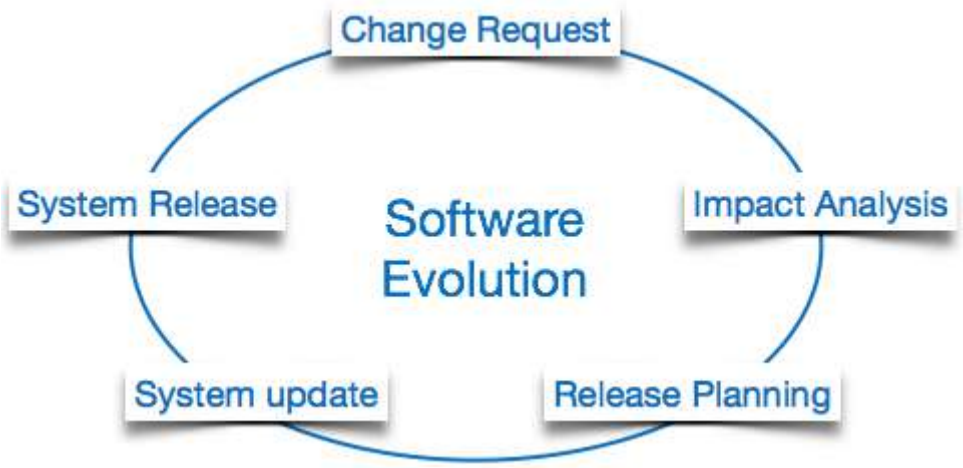
Characteristics of the Software

- It is **intangible**, meaning it cannot be seen or touched.
- It is **non-perishable**, meaning it does not degrade over time.
- It is **easy to replicate**, meaning it can be copied and distributed easily.
- It can be **complex**, meaning it can have many interrelated parts and features.
- It can be **difficult to understand and modify**, especially for large and complex systems.
- It can be **affected by changing requirements**, meaning it may need to be updated or modified as the needs of users change.
- It can be **impacted by bugs and other issues**, meaning it may need to be tested and debugged to ensure it works as intended.

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 Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. These can be combined into various categories, though each of them is contained in one another:

Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Process Models

The software development paradigm helps developer **to select a strategy to develop the software**.

A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle.

A few of software development paradigms or process models are defined as follows:

1) *Waterfall Model*

- The waterfall model is the **classic model or oldest model** and is known as mother of all the model. It is widely used in government projects and many vital projects in company.
- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, **each phase is executed completely before the beginning of the next phase**. Hence the phases do not overlap in waterfall model.
- This model is **used for small projects**.
- In this model, **feedback is taken after each phase** to ensure that the project is on the right path.
- **Testing part starts only after the development is completed**.

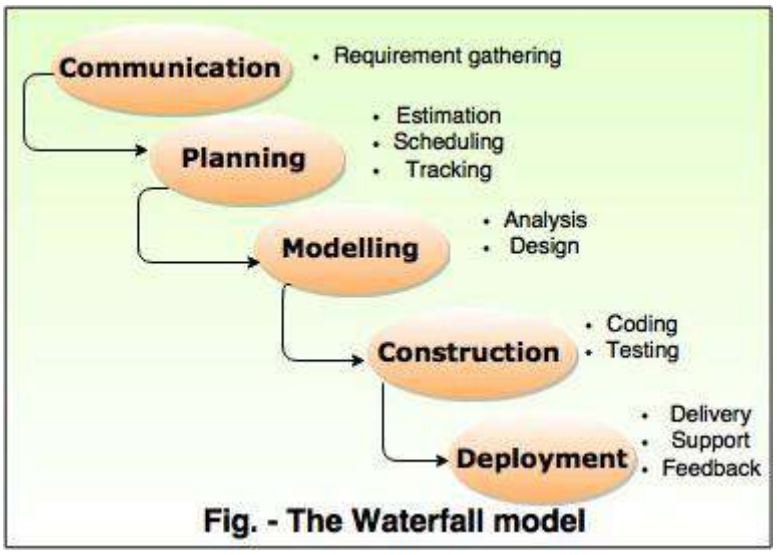


Fig. - The Waterfall model

Following are the phases in waterfall model:

i) Communication
The software development starts with the communication between customer and developer.

ii) *Planning*
It consists of **complete estimation, scheduling** for project development.

iii) *Modelling*

- Modelling consists of complete requirement analysis and the design of the project i.e algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

iv) *Construction*

- Construction **consists of code generation** and the **testing** part.
- Coding part implements the design details using an appropriate programming language.
- **Testing** is to check whether the **flow of coding is correct** or not.
- Testing also **checks** that the program provides **desired output**.

v) *Deployment*

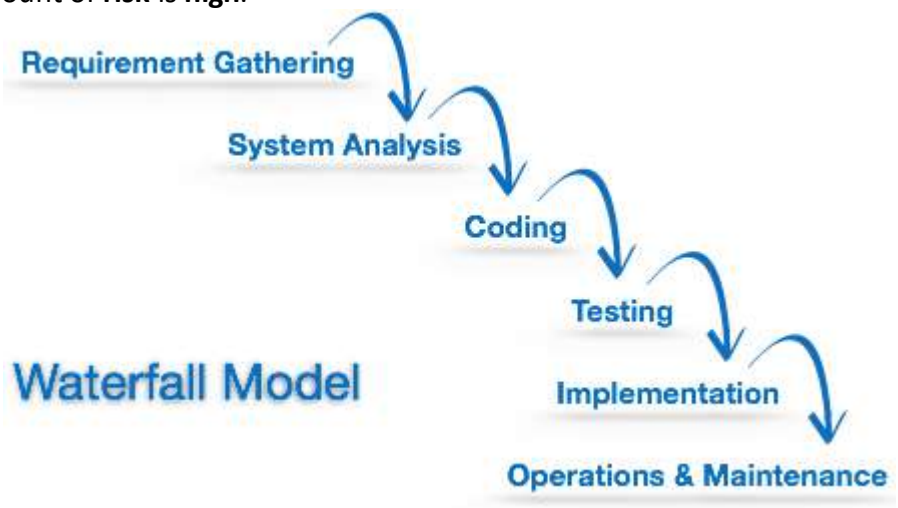
- Deployment step consists of **delivering the product** to the customer and taking **feedback** from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Advantages of Waterfall model

- The waterfall model is **simple** and **easy** to understand, to implement, and use.
- **All the requirements** are **known** at the **beginning** of the project, hence it is easy to manage.
- It **avoids overlapping** of phases because each phase is completed at once.
- This model **works for small projects** where the requirements are easily understood.
- This model is **preferred** for those projects where the **quality is more important as compared to the cost** of the project.

Disadvantages of the Waterfall model

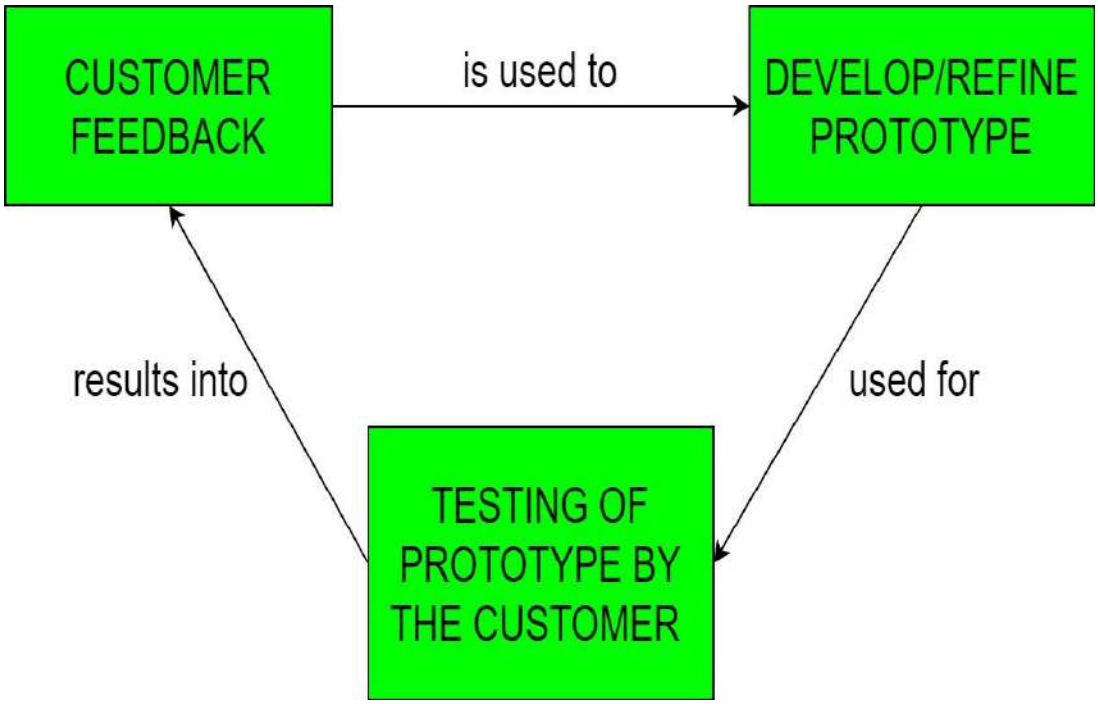
- This model is not good for **complex** and **object oriented** projects.
- In this model, the changes are not permitted so it is **not fit for moderate** to **high risk** changes in **project**.
- It is a poor model for long duration projects.
- The problems with this model are uncovered, until the software testing.
- The amount of **risk** is **high**.



2) *Prototype Model:*

Prototyping is defined **as the process of developing a working replication of a product or system that has to be engineered.**

It offers a **small-scale copy of the end product** and is used for obtaining customer feedback as described below:



The Prototyping Model is **one of the most popularly used Software Development Life Cycle Models (SDLC models).**

This model is used when the customers do not know the exact project requirements beforehand.

In this model, a prototype of the **end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.**

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software **working model** of **limited functionality**.
- In this model, **working programs** are **quickly produced**.

*The **different phases** of Prototyping model are:*

1) Communication
2) Quick design
3) Modelling and quick design
4) Construction of prototype
5) Deployment, delivery, feedback

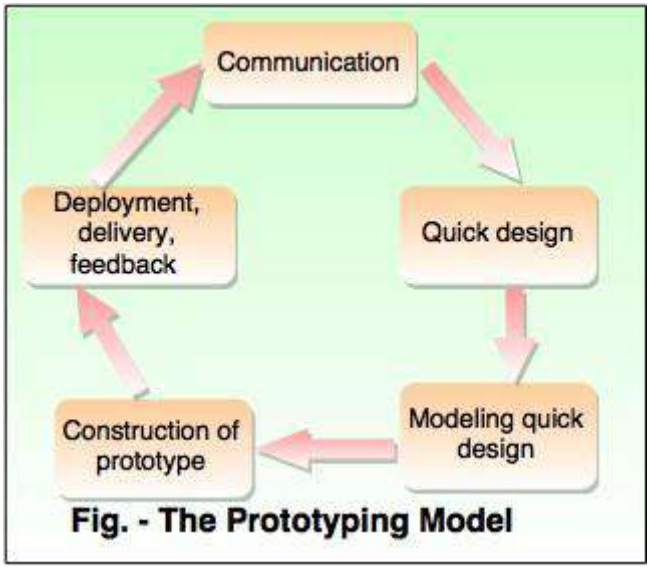


Fig. - The Prototyping Model

1. *Communication*

In this phase, developer and customer **meet** and **discuss** the overall objectives of the software.

2. *Quick design*

- Quick design is implemented when requirements are known.
- It **includes** only the **important aspects** i.e input and output format of the software.
- It **focuses** on those **aspects** which are **visible to the user** rather than the detailed plan.
- It helps to construct a **prototype**.

3. *Modelling quick design*

- This phase gives the clear idea about the **development of software** as the software is now constructed.
- It allows the developer to better understand the exact requirements.

4. *Construction of prototype*

The prototype is **evaluated** by the **customer** itself.

5. *Deployment, delivery, feedback*

- If the user is not satisfied with current **prototype** then it is **refined** according to the requirements of the user.
- The process of refining the prototype is **repeated till all the requirements of users are met.**
- When the users are satisfied with the developed prototype then the **system is developed on the basis of final prototype**.

Advantages of Prototyping Model

- In the development process of this model **users** are **actively involved**.
- The development process is the best platform to understand the system by the user.
- **Earlier error detection** takes place in this model.
- It gives **quick user feedback** for better solutions.
- It identifies the **missing functionality easily**. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model

- The **client involvement is more** and it is not always considered by the developer.
- It is a **slow process** because it takes more time for development.
- Many **changes can disturb the rhythm of the development** team.
- It is a **throw away prototype** when the users are confused with it.

3) *RAD Model*

- RAD is a **Rapid Application Development** model.
- RAD is a linear sequential software development process.
- Using the RAD model, **software** product is **developed** in a **short period of time**.
- The **initial activity** starts with the **communication** between customer and developer.
- Planning depends upon the initial requirements and then the **requirements are divided into groups.**
- Planning is more important to work together on different modules.

The RAD model consist of following phases:

1) Business Modelling
2) Data modelling
3) Process modelling
4) Application generation

5) Testing and turnover

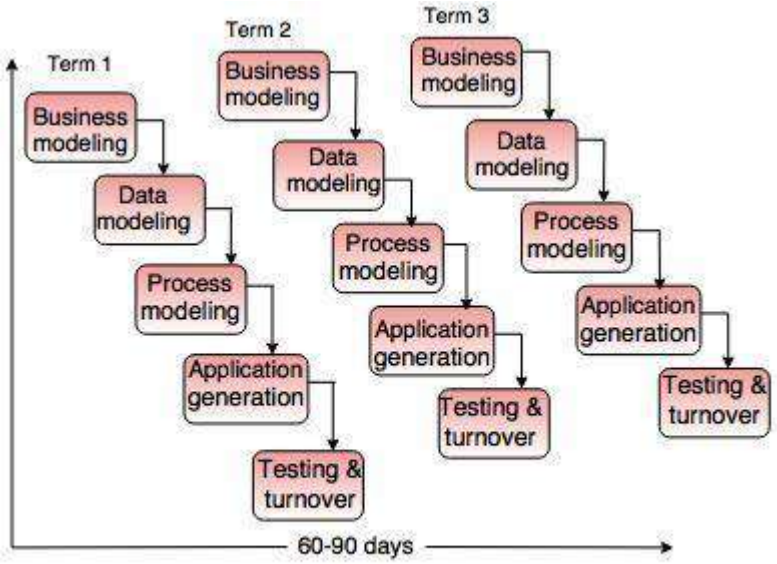


Fig. - RAD Model

1) *Business Modelling*

- Business modelling consists of the **flow of information between various functions in the project.**
  For example, what **type of information** is produced by every **function** and which are the **functions to handle that information**.
- It is necessary to perform **complete business analysis** to get the essential business information.

2) *Data modelling*

- The information in the business modelling phase is refined into the **set of objects** and it is essential for the business.
- The **attributes of each object** are identified and defined the **relationship** between objects.

3) *Process modelling*

- The data objects defined in the data modelling phase are changed to fulfil the **information flow to implement the business model**.
- The process description is created for **adding**, **modifying**, **deleting** or **retrieving** a data object.

4) *Application generation*

- In the application generation phase, the **actual system is built**.
- To construct the software the **automated tools are used**.

5) *Testing and turnover*

- The **prototypes are independently tested** after each iteration so that the overall testing time is reduced.
- The **data flow** and the **interfaces** between all the components are fully tested. Hence, most of the programming components are already tested.

***Advantages** of RAD Model*

- The process of **application development and delivery** are **fast**.
- This model is **flexible**, if any changes are required.
- **Reviews** are taken from the clients at the starting of the development hence there are **lesser chances to miss the requirements**.

***Disadvantages** of RAD Model*

- The **feedback from the user** is required **at every development phase**.
- This **model is not a good choice for long term and large projects**.

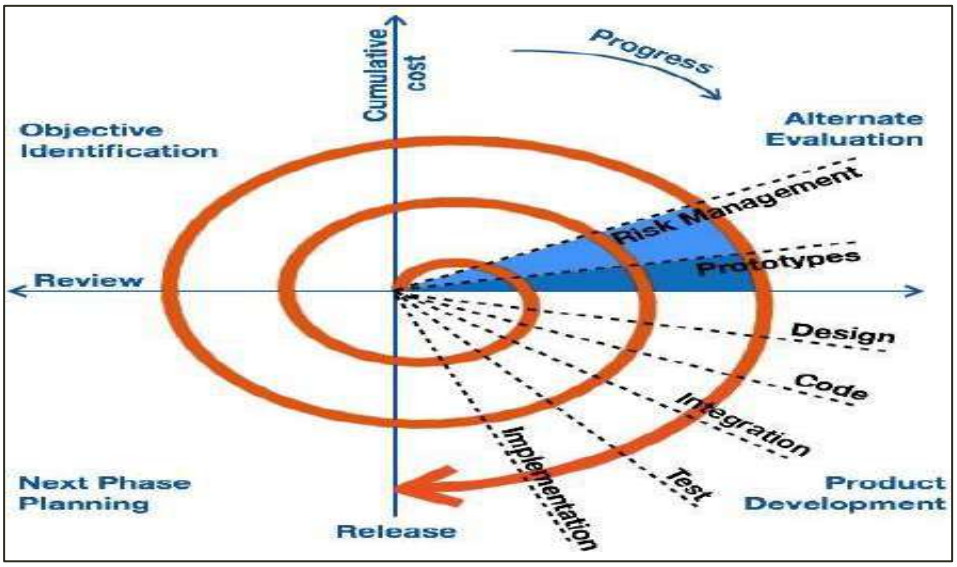
SPIRAL MODEL

- The spiral model, initially proposed by Boehm, is an evolutionary software process model that **couples the iterative feature of prototyping** with the **controlled** and **systematic** aspects of the **linear sequential model**.
- Using the spiral model, the software is **developed in a series of incremental releases.**

- The Spiral model of software development is shown in bellow figure.
- The diagrammatic representation of this **model appears like a spiral with many loops**.

- The **exact number of loops in the spiral is not fixed**. **Each loop of the spiral represents a phase of the software process.**
- For example, the innermost loop might be concerned with feasibility study, the next loop with requirements specification, the next one with design, and so on.

- **Each phase in this model is split into four sectors (or quadrants)** as shown in bellow figure. The following activities are carried out during each phase of a spiral model.



Spiral Model

First quadrant (Objective Setting)

- During the first quadrant, it is needed **to identify the objectives** of the phase.
- **Examine the risks** associated with these objectives.

Second Quadrant (Risk Assessment and Reduction)

- A **detailed analysis** is carried out **for each identified project risk**.
- **Steps are taken to reduce the risks**. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

- **Develop and validate the next level of the product** after resolving the identified risks.

Fourth Quadrant (Review and Planning)

- **Review** the results achieved so far with the customer and **plan the next iteration** around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

Circumstances to use spiral model

The spiral model is **called a meta model** since **it encompasses all other life cycle models**.

Risk handling is inherently built into this model.

The **spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks**.

However, this **model is much more complex** than the other models – this is probably a factor deterring its use in ordinary projects.

Advantages

- High amount of **risk analysis**
- Useful for **large** and **mission-critical projects**.

Disadvantages

- Can be a **costly** model to use.
- Risk analysis **needed** highly particular **expertise**
- **Doesn't work** well for **smaller projects**.

4) *Agile Model*

- Agile model is a **combination of incremental and iterative process** models.
- This model **focuses on the user's satisfaction** which can be achieved with **quick delivery of the working software** product.

Fig. Agile Model

Phases of Agile Model

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

- Requirements gathering
- Design the requirements
- Construction/ iteration

- Testing/ Quality assurance
- Deployment
- Feedback

- **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**.

- **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.

- **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.**

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

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

- **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 model **breaks the product into individual iterations**.
- **Every iteration includes cross functional teams working on different areas such as planning, requirements, analysis, design, coding, unit testing and acceptance testing.**
- At the end of an iteration working product shows to the users.

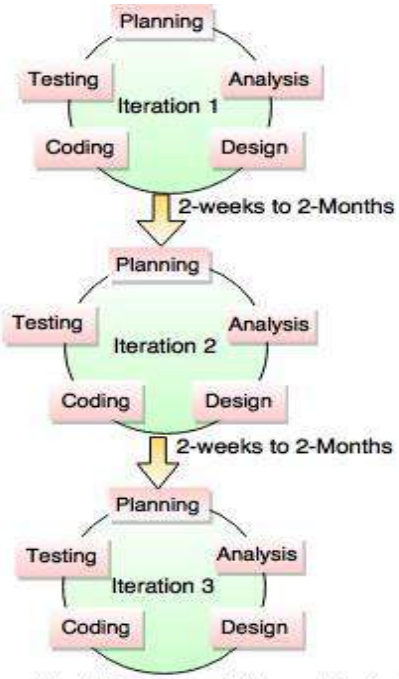


Fig. - Graphical Representation of Agile Model

- **With every increment**, **features are incremented** and the final increment hold all the features needed by the customers.
- The **iterations** in agile process are **shorter in duration** which can vary from **2 weeks to 2 months.**

Advantages of Agile model

- Customers are satisfied because of **quick and continuous delivery** of useful software.
- Regular delivery of working software.
- Face to face **interaction between the customers**, **developers** and **testers** and it is best form of communication.

  Even the **late changes in the requirement can be incorporated** in the software.

Disadvantages of Agile model

- It is totally depends on customer interaction. If the customer is not clear with their requirements, the development team can go in the wrong direction.
- Documentation is less, so the **transfer of technology** to the new team members is **challenging**.

SSAD and OOAD

In software engineering, **SSAD and OOAD represent two distinct methodologies for analyzing and designing software systems.**

Structured Systems Analysis and Design (SSAD)

SSAD is a **traditional**, **process-oriented approach** that focuses on the flow of data and the functions performed within a system.

It emphasizes **breaking down a system into smaller, manageable components** through functional decomposition.

Key techniques and tools in SSAD include:

- **Data Flow Diagrams (DFDs):** Illustrate how data moves through a system and the processes that transform it.
- **Entity-Relationship Diagrams (ERDs):** Model the data requirements by **identifying entities** (data objects) and the **relationships** between them.
- **Process Specifications:** Detail the logic and steps involved in each process.

SSAD is often considered suitable for smaller, well-defined systems where the emphasis is on the sequential execution of processes and data transformation.

Object-Oriented Analysis and Design(OOAD)

Object-Oriented Analysis and Design (OOAD) is a **way to design software by thinking of everything as objects similar to real-life things**.

In OOAD, we first understand what the system needs to do, then identify key objects, and finally decide how these objects will work together. This approach helps make software easier to manage, reuse, and grow.

What is Object-Oriented Analysis and Design(OOAD)?

OOAD is **based on the concepts of object-oriented programming (OOP)** and is an **organized** and **systematic approach to designing** and **developing software** systems.

It is a software engineering paradigm that **integrates** two distinct but closely related processes: **Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD).**

Important Aspects of OOAD

Below are some important aspects of OOAD:

- **Object-Oriented Programming:** In this the real-world items are represented/mapped as software objects with attributes and methods that relate to their actions.
- **Design Patterns:** Design patterns are used by OOAD to help developers in building software systems that are more efficient and maintainable.
- **UML Diagrams:** UML diagrams are used in OOAD to **represent the different components and interactions** of a software system.
- **Use Cases:** OOAD uses **use cases to help developers understand the requirements of a system** and to design software systems that meet those requirements.

Object-Oriented Analysis

Object-Oriented Analysis (OOA) is the process **of understanding and analyzing the system requirements by looking at the problem scenario in terms of objects**.

- These objects **represent real-world entities** or **concepts** that are relevant to the system being developed.
- During OOA, the **goal is to identify the objects, their attributes, behaviors, and relationships**, without focusing on how the system will be implemented.

For example: Lets say you're building a game:

- OOA helps you figure out all the things you need to know about the game world - the **characters**, their **features**, and how they **interact**.
- It's like making a map of everything important.
- OOA also helps you understand **what your game characters will do**. It's like writing down a script for each character.
- **Every program has specific tasks or jobs it needs to do**. OOA helps you list and describe these jobs.

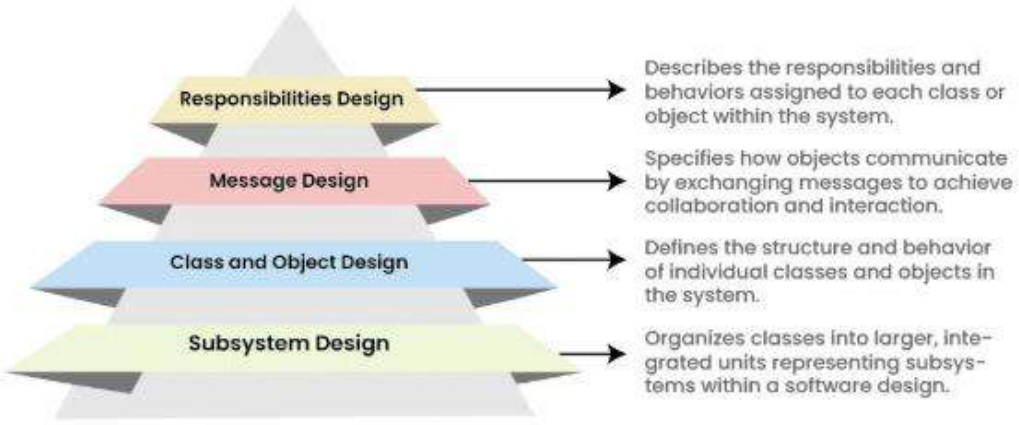
- In our game, it could be **tasks like moving characters or keeping score**. It's like making a to-do list for your software.
- OOA is smart about breaking things into different parts. It **splits the job into three categories:** things your game knows, things your game does, and how things in your game behave.

Object-Oriented Design
In the object-oriented software development process, **the analysis model, which is initially formed through object-oriented analysis (OOA), undergoes a transformation during object-oriented design (OOD**) i.e implementation of the conceptual model developed in OOA. This evolution is crucial because it shapes the analysis model into a detailed design model.

Furthermore, **as part of the object-oriented design process**, it is essential to define specific aspects:

- **Data Organization of Attributes:**
    - OOD involves specifying how data attributes are organized within the objects.
    - This includes determining the types of data each object will hold and how they relate to one another.
- **Procedural Description of Operations:**
    - OOD **requires a description for each operation** that an object can perform.
    - This involves **detailing the steps or processes involved** in carrying out specific tasks.

Below diagram shows a design pyramid for object-oriented systems. It is having the following four layers.



The Object Oriented Design Pyramid

1. **The Subsystem Layer:** It represents the subsystem that enables software to achieve user requirements and implement technical frameworks that meet user needs.
2. **The Class and Object Layer:** It represents the class hierarchies that enable the system to develop using generalization and specialization. This layer also represents each object.

3. **The Message Layer:** This layer deals with how objects interact with each other. It includes messages sent between objects, method calls, and the flow of control within the system.
4. **The Responsibilities Layer:** It focuses on the responsibilities of individual objects. This includes defining the behavior of each class, specifying what each object is responsible for, and how it responds to messages.

Benefits of Object-Oriented Analysis and Design(OOAD)

- It **increases** the **modularity and maintainability** of software by encouraging the creation of tiny, **reusable parts** that can be **combined to create more complex systems.**
- It provides a high-level, **abstract representation** of a software system, making understanding and maintenance easier.
- It promotes object-oriented design principles and the **reuse of objects**, which **lowers the amount of code** that must be produced and raises the quality of the program.
- Software engineers can use the same language and method that OOAD provides to communicate and work together more successfully in groups.
- It can assist developers in creating **scalable software systems** that can **adapt to changing user needs and business demands** over time.

Challenges of Object-Oriented Analysis and Design(OOAD)

- Because **objects and their interactions need to be carefully explained** and handled, it might complicate a software system.
- Because objects must be instantiated, managed, and interacted with, this may **result in additional overhead** and reduce the software's speed.
- **For beginner** software engineers, OOAD **might have a challenging learning curve** since it requires a solid grasp of OOP principles and methods.
- It can be a **time-consuming process** that involves significant upfront planning and documentation. This can lead to longer development times and higher costs.
- OOAD can be **more expensive** than other software engineering methodologies due to the upfront planning and documentation required.

Real world applications of Object-Oriented Analysis and Design(OOAD)

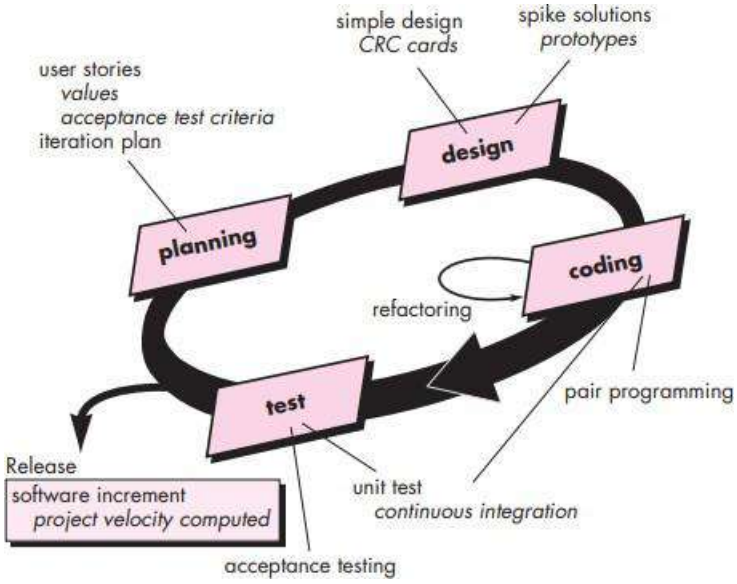
Some examples of OOAD's practical uses are listed below:

- **Banking Software:** In banking systems, OOAD is frequently used to simulate complex financial transactions, structures, and customer interactions. Designing adaptable and reliable financial apps is made easier by OOAD's modular and scalable architecture.
- **Electronic Health Record (EHR) Systems:** Patient data, medical records, and healthcare workflows are all modeled using OOAD. Modular and flexible healthcare apps that may change to meet emerging requirements can be made through object-oriented principles.
- **Flight Control Systems:** OOAD is crucial in designing flight control systems for aircraft. It helps model the interactions between different components such as navigation systems, sensors, and control surfaces, ensuring safety and reliability.
- **Telecom Billing Systems:** In the telecom sector, OOAD is used to model and build billing systems. It enables the modular and scalable modeling of complex subscription plans, invoicing rules, and client data.
- **Online Shopping Platforms:** E-commerce system development frequently makes use of OOAD. Product catalogs, user profiles, shopping carts, and payment procedures are all modeled, which facilitates platform maintenance and functionality expansion.

Agile Process models

1. Extreme Programming (XP)
2. Adaptive Software development (ASD)
3. Dynamic software Development Method (DSDM)
4. Scrum
5. Crystal
6. Lean software Development (LSD)
7. Feature Driven development (FDD)
8. Agile Modeling (AM)
9. Agile unified Process(AUP)

1. Extreme Programming(XP)



Extreme Programming (XP) Overview

- **Object-oriented** development approach.
- Organized around **four main activities**: Planning, Design, Coding, Testing.

---

1. Planning

- Begins with **listening to user requirements**, written by customers to describe features.
- Requirements are prioritized by **business value** and estimated by developers in **development weeks**.
- Requirements longer than 3 weeks are **split**.
- Requirements grouped into releases; release scheduling can prioritize **all, high-value, or high-risk** requirements first.
- After the first release, **project velocity** (requirements completed) is used to guide future planning.
- Stories can be **added, changed, or removed** throughout the project.

---

2. Design

- Follows **KIS – Keep It Simple** principle.
- Uses **CRC cards** (Class-Responsibility-Collaborator) for object-oriented design.
- **Spike solutions** (prototypes) are created for complex design problems to reduce risk.

---

3. Coding

- Begins with writing **unit tests** before coding.
- Emphasizes **simplicity** and **instant feedback** through testing.
- Uses **pair programming** – two developers work together at one workstation.
- Promotes **continuous integration** to detect issues early and maintain code quality.

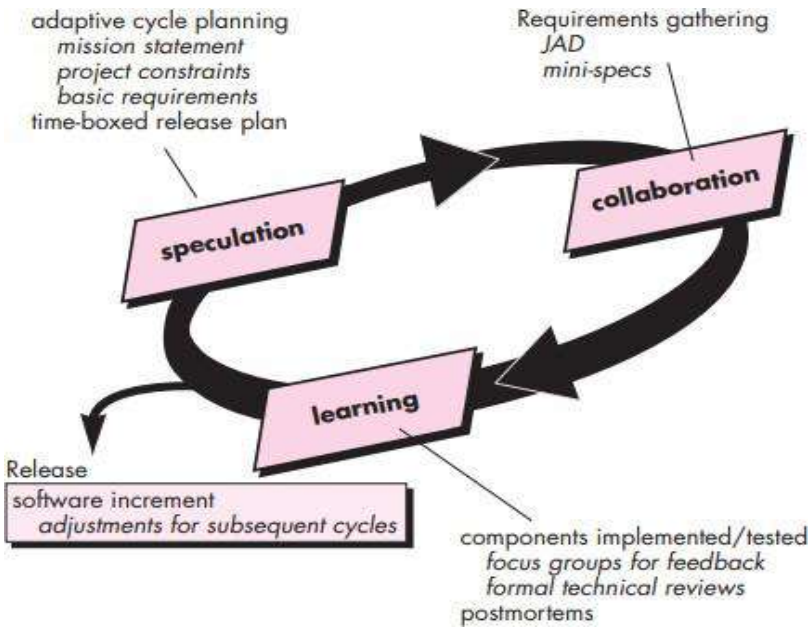
---

4. Testing

- Unit tests are **automated** and support **regression testing**.
- Daily testing ensures **progress tracking** and early bug detection.
- **Acceptance tests** are defined by the customer to validate overall system functionality based on user stories.

---

2. Adaptive Software development (ASD):



Adaptive Software Development (ASD)

Designed for **building complex, changing systems**.

- Emphasizes **team collaboration**, **self-organization**, and **continuous learning**.
- Applicable across **any process model**—not limited to one method.

ASD Lifecycle Phases:

1. **Speculation**
    - Project is initiated using the **customer's mission**, **constraints**, and **basic requirements**.
    - **Adaptive cycle planning** defines release cycles.
2. **Collaboration**
    - Focuses on **teamwork**, **communication**, and **individual creativity**.
    - Built on **trust**, where team members:
        - Criticize constructively
        - Help without resentment
        - Contribute equally
        - Are skilled and communicative
3. **Learning**
    - Continuous learning is as important as task completion.
    - Teams **learn in three ways**:
        - **Focus groups**
        - **Technical reviews**
        - Project De brief
    - Helps avoid overestimating understanding of tech, process, or project.

- ASD supports **self-organizing teams**, **interpersonal collaboration**, and **both individual and team learning**.
- Promotes **higher success rates** in software projects, especially in complex or uncertain environments.

3. Dynamic software Development Method (DSDM):

- **DSDM** is an **Agile methodology** focused on **delivering systems quickly** using **incremental prototyping** in a controlled environment.
- It follows the **80% rule**: only **enough work is done in each iteration** to move forward; **remaining details are handled in later** iterations as requirements evolve.

DSDM Life Cycle Phases

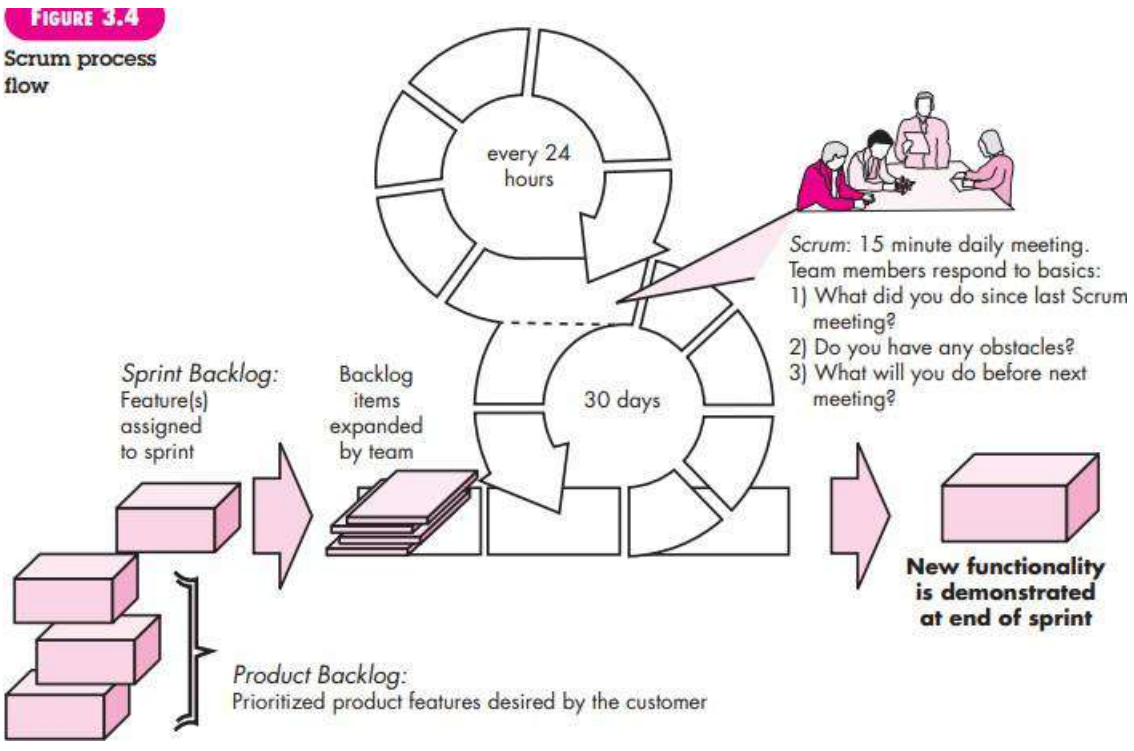
1. **Feasibility Study**
   - Assesses basic business needs, constraints, and viability for using DSDM.
2. **Business Study**
   - Defines functional and information requirements, system architecture, and maintainability needs.
3. **Functional Model Iteration**
   - Develops prototypes to show functionality and gather user feedback for refining requirements.
4. **Design and Build Iteration**
   - Engineers prototypes for real-world use. Often overlaps with functional modeling.
5. **Implementation**
   - Deploys software to the live environment. May not be fully complete; changes can be made in the next cycle.

Adaptability

- DSDM can be **combined with XP** (for development practices) and **ASD** (for team collaboration and self-organization).
- Managed by the **DSDM Consortium**, a global group responsible for maintaining and evolving the method.

4. Scrum :

FIGURE 3.4
Scrum process flow



- It follows the **Agile Manifesto** and supports **fast, adaptive development** using short, focused iterations.

Scrum Framework Activities

- Includes **requirements, analysis, design, evolution, and delivery**.
- Work is done in **iterations called sprints**, which are **time-boxed** (usually 30 days).
- **Sprints** provide a stable environment — **no changes** allowed mid-sprint.

Key Scrum Components

1. **Product Backlog**
    - A **prioritized list of features or requirements** with business value.
    - Continuously updated by the **product manager**.
2. **Sprint**
    - A set of tasks selected from the backlog to be completed within a time-box.
    - **Each sprint aims to deliver a potentially shippable product increment.**
3. **Scrum Meetings (Daily Stand-ups)**
    - **15-minute daily check-ins** where team members answer:
        - What tasks did you complete yesterday?
        - What tasks are you working on today?
        - What's blocking your progress?
    - Led by the **Scrum Master**, who facilitates and removes obstacles.
4. **Demo (Sprint Review)**
    - Shows the completed functionality to the customer for feedback.
    - May not include all features — only what was achievable during the sprint.

Scrum Principles

- Supports **changing requirements**, **tight timelines**, and **business-critical projects**.
- Encourages **self-organizing teams** and **early problem identification**.
- Aims to **deliver value incrementally** and adapt quickly to feedback.

---

5. Crystal :

- **Developed to support flexible, lightweight processes** that **adapt to different team sizes and project needs.**
- Views development as a **resource-limited, cooperative game** aimed at delivering **working software** and preparing for future work.
- **Crystal** is a **family of agile methodologies**, each tailored to different project sizes and criticalities.
- All Crystal methods share **core principles** but vary in:
  - **Roles**
  - **Processes**
  - **Work products**
  - **Practices**
- Teams choose the **most suitable Crystal method** (e.g., Crystal Clear, Crystal Orange) based on **project type and environment**.

6. **Lean Software Development (LSD):** Lean Software Development (LSD) has adapted the principles of lean manufacturing to the world of software engineering. The lean principles can be summarized as
   1) eliminate waste,
   2) build quality ,
   3) create knowledge,
   4) defer commitment,
   5) deliver fast,
   6) respect people,
   7) optimize the whole.

Each of these principles can be adapted to the software process.

For example, eliminate waste within the context of an agile software project can be interpreted to mean (1) **adding no extraneous features or functions**,

(2) assessing the cost and schedule impact of any newly requested requirement,

(3) removing any superfluous process steps,

(4) establishing mechanisms to improve the way team members find information,

(5) ensuring the testing finds as many errors as possible

(6) reducing the time required to request and get a decision that affects the software or the process that is applied to create it, and

(7) streamlining the manner in which information is transmitted to all stakeholders involved in the process.

7. Feature Driven Development (FDD):

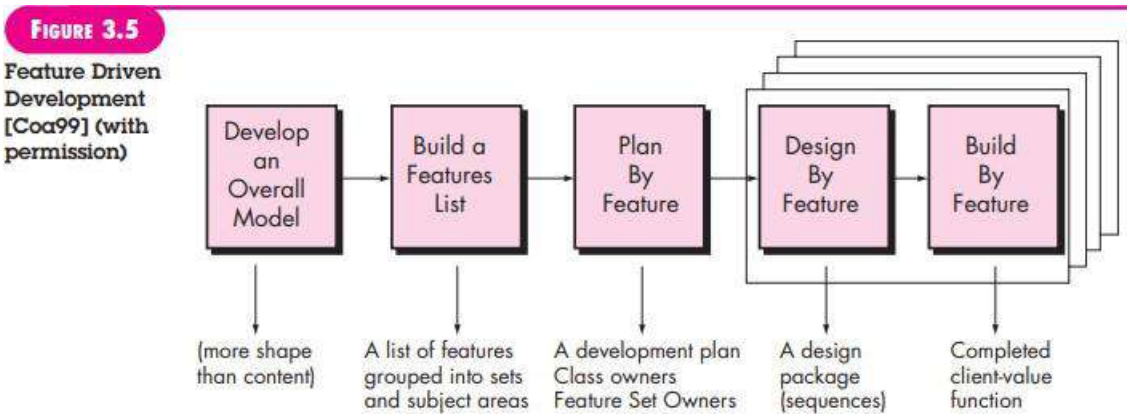


FIGURE 3.5
Feature Driven Development [Coa99] (with permission)

- **FDD** is an **agile, object-oriented** process model.
- Best suited for **moderate to large projects**.

Core Concepts:

- Focuses on **developing features** (small, client-valued functions).
- Uses **feature-based decomposition** to manage complexity.
- Encourages **team collaboration** and **clear communication** (verbal, text, graphical).

Six Milestones for Each Feature:

1. **Design Walkthrough**
2. **Design**
3. **Design Inspection**
4. **Code**
5. **Code Inspection**
6. **Promote to Build**

Benefits of FDD:

- **Features are small**, easy to describe, review, and inspect.
- **Incremental development** — operational features delivered **every 2 weeks**.
- Features are grouped **hierarchically by business domain**.
- **Planning and tracking** are based on feature progress, not arbitrary task lists.
- Emphasizes **quality** via inspections, audits, metrics, and pattern use.

8. Agile Modeling: (AM):

- **Agile Modeling** is a **lightweight, practice-based approach** to modeling and documenting software systems.
- It helps manage **complex, business-critical systems** by improving understanding, enabling effective partitioning, and ensuring quality.
- Models are created to be **"just good enough"**, not perfect.

Core Values & Principles:

- Aligned with the **Agile Manifesto**
- Encourages **courage** to make tough design decisions and **humility** to value input from business experts

---

Unique Features of Agile Modeling:

1. **Model with a purpose**
2. **Use multiple models**
3. **Travel light** (avoid unnecessary documentation)
4. **Focus on content over representation**
5. **Know your models and tools**
6. **Adapt locally** (tailor practices to your team/project)

Agile Unified Process(AUP) :

- **AUP** combines the **Unified Process** with **Agile principles**, following a **"serial in the large, iterative in the small"** approach.
- Uses four classic UP phases: **Inception, Elaboration, Construction, Transition** (linear at high level).
- Within each phase, **iterations** are used to deliver **working software quickly and adaptively**.

Core Iteration Activities:

1. **Modeling** – Create UML models that are **"just good enough"**; avoid unnecessary detail.
2. **Implementation** – Translate models into working code.
3. **Testing** – Apply Agile-style testing to validate functionality and catch defects early.
4. **Deployment** – Deliver software increments and gather user feedback.

5. **Configuration & Project Management** – Handle changes, risks, and team coordination.
6. **Environment Management** – Maintain tools, standards, and infrastructure.

AUP aims to keep processes **lightweight and value-driven**, while still benefiting from the structure of the Unified Process.