**Introduction to software engineering**

**Chapter 1**

**Program versus software**

Programs are simply the source codes and object codes those fulfill some specific purpose. For example that we create in laboratories, like a program that stores records of n employees, matrix addition etc.

Many people equate the term software with computer programs. Software is not merely the chunk of codes that performs specific functions, rather software is a system that encompasses many other related materials except code.

Software= Programs+ Good user Interface + User Manuals + Documentations

Software is defined as:

According to Roger Pressman[Software Engineering: A Practitioner’s Approach],”Software encompasses programs that execute within a computer of any size and architecture, contents that are presented as the computer programs execute and documents that encompass all forms of electronic media.”

The definition of software followed at Princeton University says:

Computer software in general is used to describe a collection of computer programs, procedures and documentation that performs certain tasks on a computer system.”

|  |  |
| --- | --- |
| **Programs** | **Software** |
| * Programs are developed by individuals for their personal use. | * Software is usually developed by a group of software engineer in a team. |
| * Have limited functionality. | * More functionality. |
| * Small in size. | * Large in size. |
| * Single user. | * Large in size. |
| * Lacks proper documentation. | * Good documentation support. |
| * Adhoc development. | * Systematic development. |
| * Lack of user interface. | * Good user interface. |

**Software characteristics**

1. Software is developed or engineered, it is not manufactured.
2. Software development presents a job- shop environment.
3. Here each product is custom- built and hence unique.
4. It cannot be assembled from exiting  components
5. All the complexity of a job shop (the problem of design, estimating and scheduling) are present here.
6. human skill the most important element in a job shop is also an important element in software development.

3) Time and effort for software development are hard to estimate

4) Software does not wear out but will deteriorate.

a) Software normally does not lose its functionality with use.

5) Testing software is extremely difficult, because even a modest- size programs (< 5000

executable statements) can contain enough executable paths.

6) Software is intangible and hard to measure.

7) Software has discontinuous operational nature.

**The Software engineering Umbrella Activities**

Software engineering process framework activities are complemented by a number of

umbrella activities. Umbrella activities are applied throughout a series of software projects  and help a software team manage and control progress, quality , change and risk.

Typical umbrella activities are as follows

1. software project tracking and control
2. risk management
3. software quality assurance.
4. technical reviews
5. measurement
6. software configuration management
7. reusability management
8. work product preparation and production

**Software Myths**

It is a belief about software and the process used to build the software in the earlier days. Some of the software myths are listed below:

1. Software is easy to change.
2. Computers provide greater reliability than the devices they replace.
3. Testing software or ‘proving’ software correct can remove all the errors.
4. Reusing software increases safety.
5. Software can work right the first time.
6. Software can be designed thoroughly enough to avoid most integration problems.
7. Software with more features is better software.
8. Addition of more software engineers will make up the delay.
9. Aim is to develop working programs.

Myths were compiled (categorized) in 3 aspects.

**Management Myths**

* **Myth:**

We already have a documentation that’s full of standards and procedures for building software. Won’t that provide my people with everything they need to know?

* **Reality :**

The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable?

* **Myth:**

If we get behind schedule, we can add more programmers and catch up.

* **Reality:**

Software development is not mechanistic process like manufacturing. According to Brooks; “Adding people to late software project makes it later”

* **Myth:**

If I decide to outsource the software project to third party, I can just relax and let that firm build it.

* **Reality:**

If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsource software projects.

Good managers can manage any projects.

They think that they have latest tools.

**Customer Myths**

* **Myth:**

A general statement of objectives is sufficient to begin writing the programs, we can fill the details later.

* **Reality:**

Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is a recipe for disaster. Precise requirements are developed only through effective and continuous communication between customer and developer.

* **Myth:**

Project requirements continually change but change can be easily accommodated because software is flexible.

* **Reality:**

It is true that software requirement change, but the impact of change varies with the time at which it is introduced. When the requirements changes are requested early cost impact is relatively small. However as the time passes cost impact grows rapidly because resources and design framework has already been set.

The customer always think that the software development is an easy process.

**Developer Myths/Practitioners Myths**

* **Myth:**

Once we write the program and get it to work, our work is done.

* **Reality:**

Industry data indicate that between 60 to 80 percent of all effort expanded on software will be expanded after it is delivered to the customer for the first time.

* **Myth:**

The only deliverable work product for a successful project is the working program

* **Reality:**

A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more importantly, guidance for software support.

**Software Crisis**

The word crisis is defined as “a turning point in the course of anything, decisive or crucial time, stage or event”. Much existing software is of relatively poor quality and steadily does

deteriorate.

-Many software project are either never delivered or delivered late and over

budget since the increase complexity of applications.

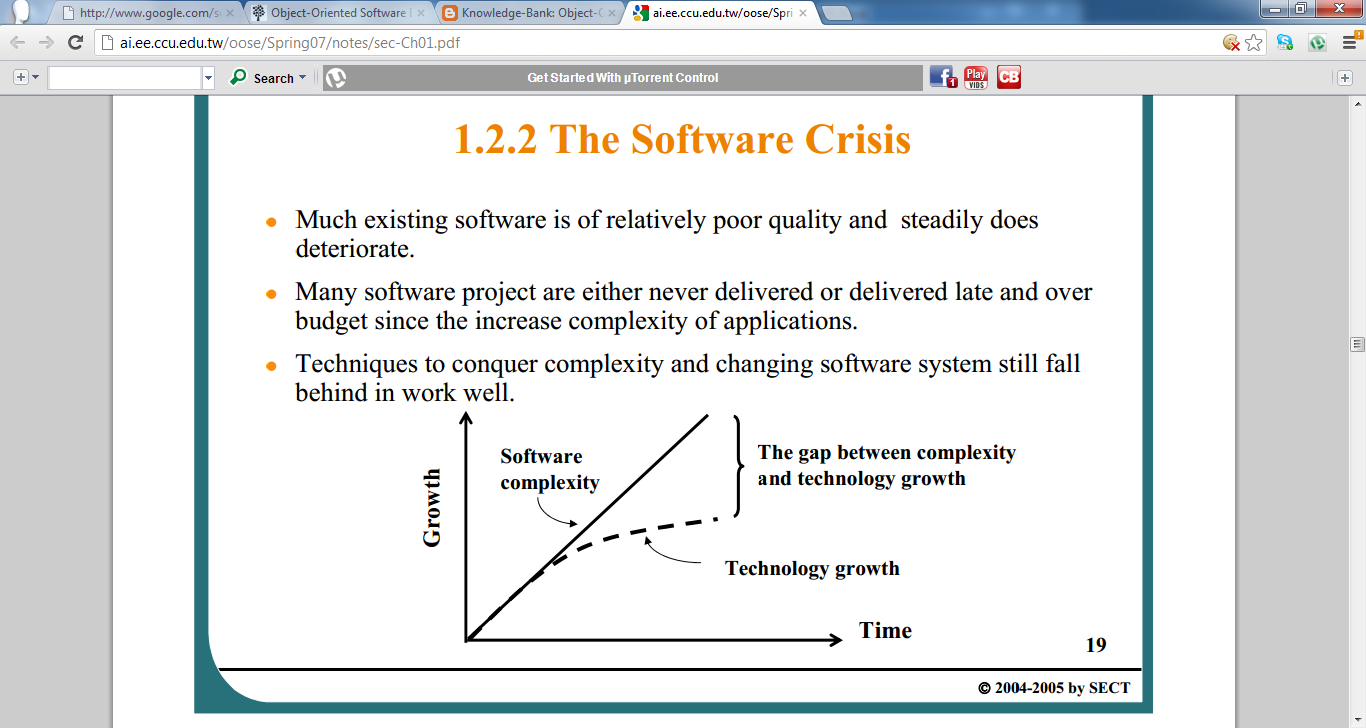
- Techniques to conquer complexity and changing software system still fall

behind in work well.

In other words, the software crisis is characterized by an inability to develop the software on time, on budget and within requirements

Reason of software crisis

1. Increases in the size and length of the software
2. Increases in cost of software compared to hardware.
3. Lack of understanding the problem and its environment
4. Lack of communication between the developer and users.
5. Increased complexity of the problems area.
6. Duplication of effort due to lack of automation in most of the software development process.



–>An example of software crisis (United States) [Mellor et al. 2004].

- At least $250 billion each year devoted to application development.

-Approximately 175,000 projects involving several million people.

- 30% projects will be cancelled before they are completed.

- More than half of the projects will cost nearly twice their original

estimates.

**Software engineering** is an engineering discipline that is concerned with all aspects of software production from the early stage of system specification to maintaining the system after it has gone into use.

It is a response to their belief that current software crisis could be solved by adapting exiting engineering practices to software development.

**The Software Engineering Problems**

->The Problems.

–Complexity. Increasing complexity of applications systems.

-Rapid platform evolution and complexity management;

- Increasing system volume in data, code, and aspects (functional and nonfunctional);

- Increasing heterogeneity in languages and paradigms, operating systems and access protocols, and technologies.

–Change. Requirements change through the life of the system.

- “The only constant in this world is change.”(Harry Palmer)

- This software characteristic is natural. So, to develop resilient software system becomes possible.

**Software Project Management (SPM)**

* Effective management of software development project is one of the vital aspect leading to the overall success.
* It’s main objective is to steer the software engineers for the completion of the software project.

**Software project management** isart of planning and leading software or refers to branch of projects management dedicated to the planning, scheduling, resource allocation, execution, tracing and delivery of software and web projects.

**Software Project Management requires**

* Managing the people, process and problem during the project work.
* Software development team to produce reliable estimates of the effort cost and project duration.
* Proper work distribution among technical and non-technical staffs.
* Adequate risk assessment.
* Software Quality Assurance(SQA) and Software Configuration Management(SCM) Formal Techincal Review.
* Decision making Capabilities.
* Good communication skills(Interpersonal, verbal and written)

**Effective management focuses on four P’s**

**Four P’s of Software Engineering (management spectrum)**

1. People
2. Product
3. Process
4. Project

**People**

*Major players* of software development process and project are:

* **Senior Managers:** Define the business issues and the strategies which has significant   influence on the project.
* **Project (Technical Manager):** To plan, motivate, organize and control the technical team (Practitioners)
* **Practitioners:** To deliver the technical skill needed to engineer the software product
* **Customers:** To specify requirements of software product
* **End users:** *The actual users* who interact with the released software
* Team leaders need to have skills to motivate the work force, organize the work so as to give the shape (product) to the concepts (requirements) and ideas and innovations *(model of leadership)*

**Product**

* The working final software is the product.
* Product defines the requirements (input from the customer) for a visible output.
* It must decide what function transforms the input to output and what performance characteristic is to be considered.

**Process**

* It is needed to establish framework activities for the project depending upon the nature of the project.
* If the process is weak, the end product will undoubtedly suffer.

**Project**

* A project is a Planned undertaking of a piece of work.
* Needs adequate planning and control to manage the complexities

Project Planning Objectives

* To provide a framework that enables the managers to make reasonable estimation of resources, cost and schedule
* Estimations are made within limited time frame at the beginning of the software project and updated regularly as the project proceeds

**Software Process**

* When you work to build a products or system, its important to go through an series of predictable steps-a road map that helps to create a timely high-quality result. Hence the road map that you follow is called a software process.
* Software engineers and their managers adapt the process to their needs and then follow it.
* The people who have requested the software have a role to play in the process of defining, building, and testing it.
* So a software process is the set of activities and associated results that produce a software product.
* The process that deals with the technical and management of software development is called software process.

**Characteristics of software process**

1. Predictability
2. Support testability and maintainability
3. support changes
4. Early defect removal
5. Process improvement and feedback

**Software Process Models**

* Development strategy that covers the process, methods and tools
* It represents the order in which the activities of s/w development will be undertaken
* It describes the sequence in which the phases of the s/w lifecycle will be performed.
* Software development can be considered as a problem solving loop
* It is a simplified description of a software process that is presented from a particular perspective.

**Types of software process models:**

**Perspective process model:**

* **Linear Sequential Model (Water Fall Model)**
* **Rapid Application Development (RAD) Model**
* **Incremental Model**
* **The V model**
* **Evolutionary  Process Models**:
  + **Spiral Model**
  + **Prototyping Model**
  + **Concurrent Development Model**
* **Specialized Process Models:**
  + **Component Based Development**
  + **Formal Methods Models**
  + **Aspect Oriented Software Development**

**Waterfall model phases (Linear Sequential Model)**

* A systematic, sequential approach to a software engineering which starts at system information engineering level and progresses through analysis, design, coding, testing and maintenance phase.
* It is a breakdown of project activities into linear sequential phases, where each depends on the deliverables of the previous one and corresponds to a specialization of tasks
* Modeled after conventional engineering cycles.
* It is termed waterfall because the model develops systematically from one phase to another in a downward fashion.
* Theoretical Way of Developing Software??

**Steps in Waterfall Model**

* + 1. **Software Requirement Analysis**
* Requirement gathering process is intensified and focused specially on software
* Study of information domain
* Requirements for system and software must be determined and reviewed  with customer

1. **Design**
   1. It is a multi-step process to address various aspects to be implemented such as data structures, software architecture, interface representation, procedural (algorithmic details) etc It translates requirements into representation of the software which can be assessed before the code generation.
2. **Code generation**

Design translated into a machine readable form

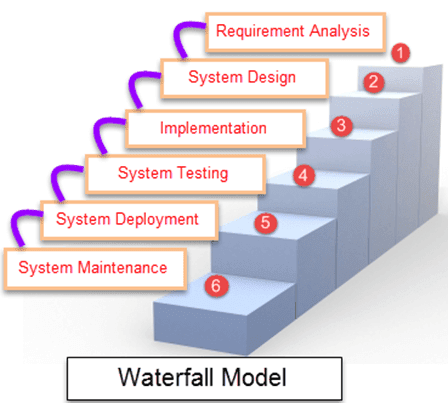
**4 Testing**

It focuses on the logical intervals of the software (all statements) and functional externals of the software, tests to uncover errors

1. **Deployment: Delivery of the final  product**

* **Basic objective:** Defined input should produce desired output.
* **Maintenance and Support**
  + **Corrective Maintenance**
  + **Perfective Maintenance**
  + **Adaptive Maintenance**

**Waterfall Model Diagram**



**Advantage**

1. **Easy to understand even by non-technical person ie customer**
2. **Each phase has well defined inputs and outputs eg Inputs to system design is requirement specification document( RSD)  and output is the design document**
3. **Easy to use as software development proceeds**
4. **Each stage has well defined deliverables or milestones**
5. **It helps the project manager in proper planning of the project.**
6. **It works well when quality requirements dominate cost and schedule requirements**
7. **It allows staff who have completed their phase activities to be freed up for other projects.**
8. **It provides structure to a technically weak or unexperienced staff.**

* **Drawback**
  + Difficulty of accommodating change after the process is underway.
  + This model is just like a one way street. Once the phase X is completed and the next phase Y has just started, then there is no provision of going back .IN other words this model is sequential in nature.
  + Appropriate when requirements are well understood.
  + It lacks overlapping and interaction among phases.
  + User have little interaction with the project team. Their feedback is not taken during development.
  + Models do not support delivery of system in pieces.
  + Model is very rigid because output of each phase is pre-requisite for successive stage
  + Customers gets opportunity to review the product very late in the life cycle because the working version of product is available very late in software development life cycle.

**When to use the water fall model?**

1. The water fall model should be limited to situation in which the requirements and the implementation of the requirements are very well understood.
2. If a company has a experience in building a certain genre system- accounting ,payroll, controller, compilers manufacturing-then a project to build another of the same type perhaps even based existing design could make efficient use of water fall model
3. The water fall model performs well for product cycle with a stable product definition and well understood technical methodologies.

**V Model :**

A variation in the representation of the waterfall model is called the V-model. The Vmodel provides a way of visualizing how verification and validation actions are applied to earlier engineering work. The V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities. As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side



**The RAD Model**

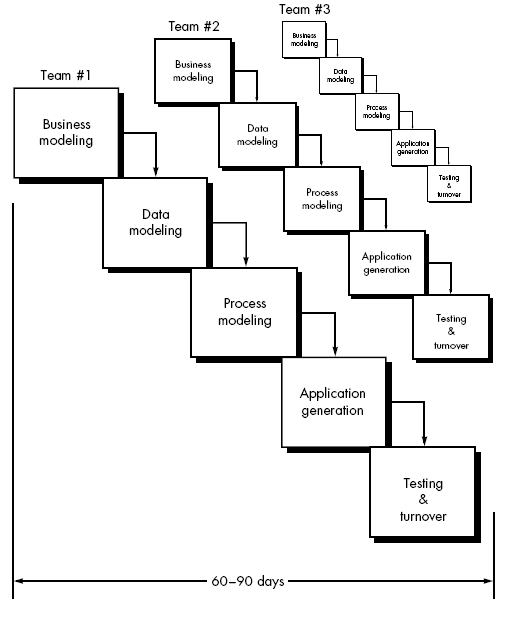
**(Rapid Application Development)**

It is a software development methodology that emphasizes on building application in a extremely short development cycle (60 to 90 days)

If requirements are well understood, and modularized (component based construction), RAD process develops fully functional system within 60 to 90 days

**When to use RAD Methodology?**

* When a system needs to be produced in a short span of time (2-3 months)
* When the requirements are known
* When the user will be involved all through the life cycle
* When technical risk(cyberattacks, password thefts’, information security incidents) is less
* When there is a necessity to create a system that can be modularized(to produce or design something in separate sections in 2-3 months of time
* When a budget is high enough to afford designers for modeling along with the cost of automated tools for code generation.



**Figure: Core elements of RAD model**

**Phases of RAD process**

1. **Communication: the customer or user provides the requirements to the developer so that developer develop the project**
2. **Planning: It includes starting to the ending (size of the project, resource  used, maintenance cost and error cost)**
3. **Modelling:**

**Core elements of RAD model**

* **Business Modeling:**  Models information flow among business functions, various input process, output aspects of information.
* **Data Modeling:**  Phase one is redefined into a set of data objects that takes part in business activity (information process).
* **Process Modeling:** The data objects are transformed through processing to achieve the information flow.
* **Application Generation:** RAD model reuses existing program components wherever possible any other automated tools to speed up the development process.
* **Testing and turn over:** Testing overhead is reduced due to reusability only the new components need testing.

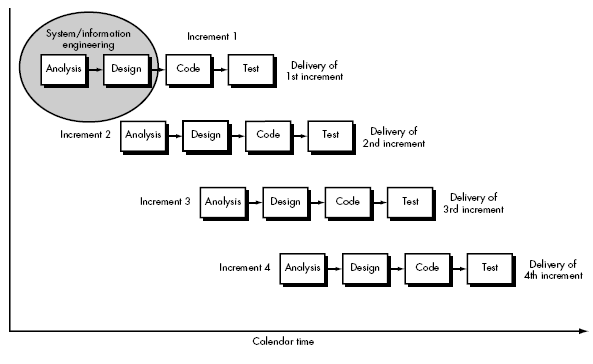
1. **Construction**
2. **Deployment**

**Rapid Application Development Advantages and Disadvantages**

|  |  |
| --- | --- |
| **Advantages of RAD Model** | **Disadvantages of RAD Model** |
| * **Flexible and adaptable to changes** | * **It can't be used for smaller projects** |
| * **It is useful when you have to reduce the overall project risk** | * **Not all application is compatible with RAD** |
| * **It is adaptable and flexible to changes** | * **When technical risk is high, it is not suitable** |
| * **It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used** | * **If developers are not committed to delivering software on time, RAD projects can fail** |
| * **Due to code generators and code reuse, there is a reduction of manual coding** | * **Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period** |
| * **Due to prototyping in nature, there is a possibility of lesser defects** | * **Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application** |
| * **Each phase in RAD delivers highest priority functionality to client** | * **Progress and problems accustomed are hard to track as such there is no documentation to demonstrate what has been done** |
| * **With less people, productivity can be increased in short time** | * **Requires highly skilled designers or developers** |

**Incremental Method**

* This model combines elements of the waterfall model in an iterative fashion.
* Each model delivers an operational product
* The features which are got incremented is always concerned by user.
* The function which were not implemented on release 1 ,they will get implemented on release 2 and after sometime release 3 will be published.
* By using incremental approach, we will get to know about features and functionality that are implemented and those yet to get implemented.
* Each linear sequence produces deliverable “increments” of the software.
* For example; Word processing software developed using this model might deliver basic file management editing, and document production functions in the first increment; and more sophisticated editing, and document production capabilities in the second increment; spelling and grammar checking in the third increment; and so on.
* When this model is used, the first increment addresses the basic requirements but many supplementary features (known and unknown) remain undelivered.
* This model focuses on the delivery of an operational product with each increment.
* It is useful when staffing is unavailable for a complete implementation by the business deadline that has been set for the project as we can release one of the version and client can keep them self-busy and provide feedback for further release.
* A need to get basic functionality to the market early.
* On projects which have lengthy development schedule.



A,B,C are modules of software product incrementally developed and delivered.

Merit of Incremental

1. Develop high risk or major functions first.
2. Each release delivers on operational product.
3. Customers can respond to each build
4. Initial product delivery is fast

Demerit of Incremental

1. Requires early definition of a complete and fully functional system to allow for the definition of increments so as high risk and major functional requirements can be implemented at some earlier release.
2. Total cost of the complete software is high because we are going through multiple time of release and gone through phases again and again,

**Evolutionary Model**

* Evolutionary models are inherently iterative in nature which is much suitable for new systems where no clear idea of the requirements, inputs and output parameters.
* It helps to develop increasingly more complete versions of the target software with each iteration.
* Good communication between customers and developers.
* Business and product requirement changes as the development proceeds, Straight path to end product becomes unrealistic.
* Tight market deadline      Limited version to be introduced.
* System requirements are well understood but details of product extensions or system extensions are not known.

**Types of Evolutionary Models:**

* **Prototyping Model**
* **Spiral Model**
* **Concurrent development Model**

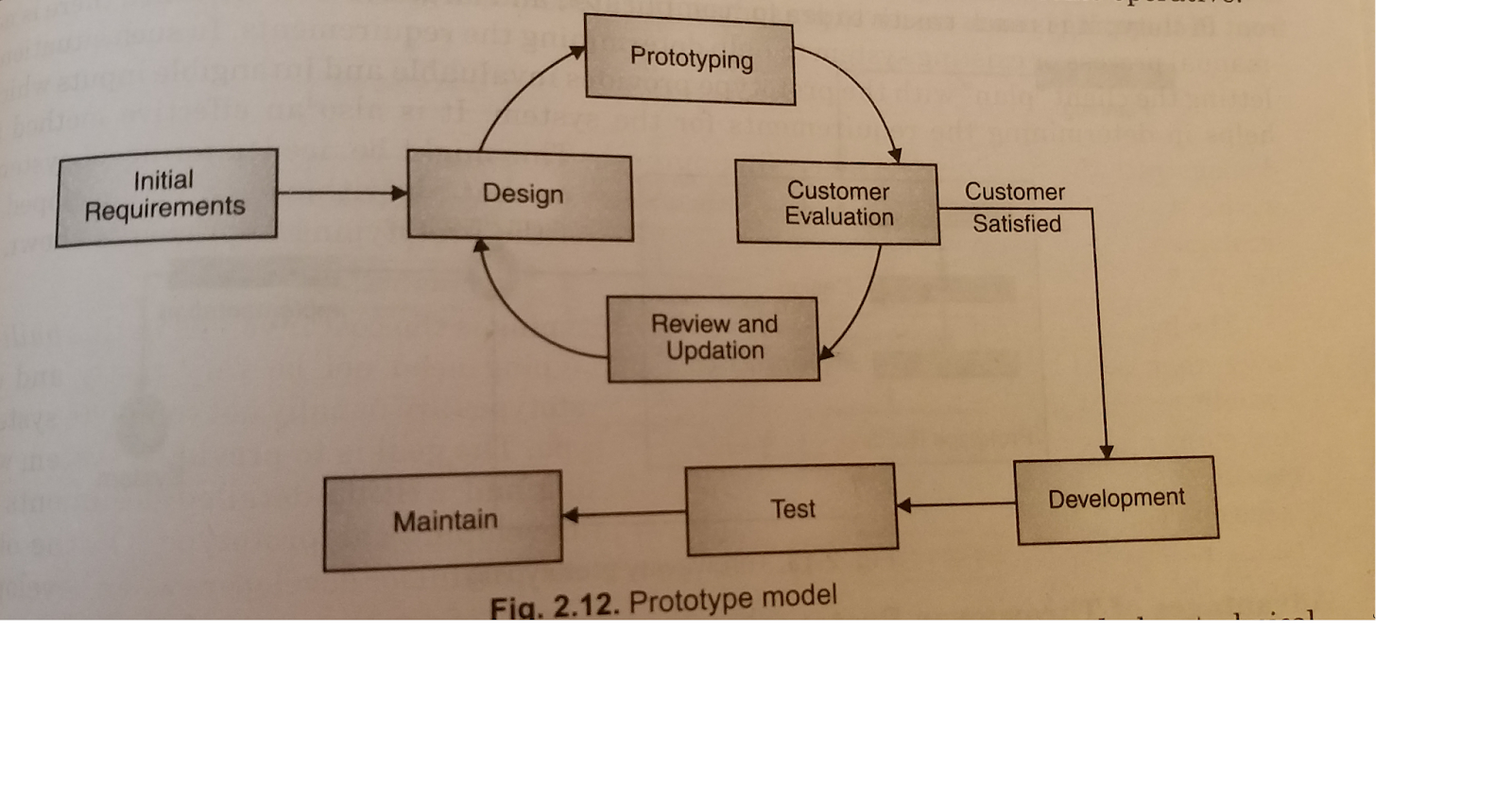
**What is Prototyping Model?**

**Prototyping Model is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved. It also creates base to produce the final system or software. It works best in scenarios where the project's requirements are not known in detail. It is an iterative, trial and error method which takes place between developer and client.**

* In this model a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements. And the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system.
* If this is the case, then prototyping paradigm may offer the best approach.
* A partial product is built in the initial stage. Therefore customers get a chance to see the product early in the life cycle.
* Requirements becomes more clear resulting into an accurate product.
* New requirements can be easily accommodate as there is scope for refinement.

**When to use prototyping model**

1. When Requirements are unstable or have to be clarified as the requirements clarification stage of a waterfall model
2. Develop user interface
3. Short-lived demonstration
4. New original development with analysis and design portions of objects -oriented development
5. You should use Prototyping when the requirements are unclear
6. It is important to perform planned and controlled Prototyping.
7. Regular meetings are vital to keep the project on time and avoid costly delays.
8. The users and the designers should be aware of the prototyping issues and pitfalls.
9. At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.
10. In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.
11. You should select the appropriate step size for each version.
12. Implement important features early on so that if you run out of the time, you still have a worthwhile system



**Prototyping Model has following six SDLC phases as follow:**

**Step 1: Requirements gathering and analysis/Identify the user’s basic requirements.**

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

**Step 2: Quick design**

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

**Step 3: Build a Prototype/Develop the initial /working prototype**

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

**Step 4: Initial user evaluation/use the prototype for further refinements:**

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

**Step 5: Refining prototype/Revise and enhance the prototype:**

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

**Step 6: Implement Product and Maintain**

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

**Types of Prototyping Models**

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

**Rapid Throwaway Prototype**

Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.

In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

**Evolutionary Prototyping**

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.

This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

**Incremental Prototyping**

In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

**Extreme Prototyping:**

Extreme prototyping method is mostly used for web development. It is consists of three sequential phases.

1. Basic prototype with all the existing page is present in the HTML format.
2. You can simulate data process using a prototype services layer.
3. The services are implemented and integrated into the final prototype.

**Advantages of the Prototyping Model**

Here, are important pros/benefits of using Prototyping models:

* Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
* Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
* Helps team member to communicate effectively
* Customer satisfaction exists because the customer can feel the product at a very early stage.
* There will be hardly any chance of software rejection.
* Quicker user feedback helps you to achieve better software development solutions.
* Allows the client to compare if the software code matches the software specification.
* It helps you to find out the missing functionality in the system.
* It also identifies the complex or difficult functions.
* Encourages innovation and flexible designing.
* It is a straightforward model, so it is easy to understand.
* No need for specialized experts to build the model
* The prototype serves as a basis for deriving a system specification.
* The prototype helps to gain a better understanding of the customer's needs.
* Prototypes can be changed and even discarded.
* A prototype also serves as the basis for operational specifications.
* Prototypes may offer early training for future users of the software system.

**Disadvantages of the Prototyping Model**

Here, are important cons/drawbacks of prototyping model:

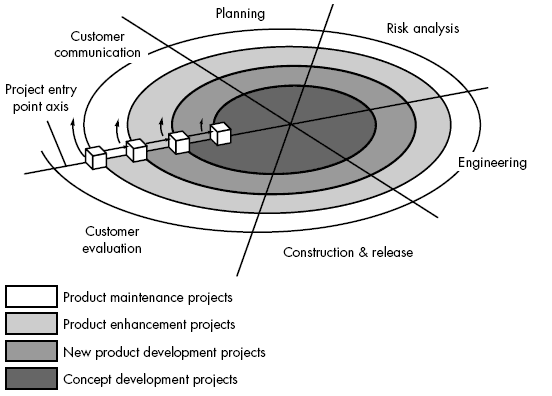
* Prototyping is a slow and time taking process.
* The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
* Prototyping may encourage excessive change requests.
* Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
* There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
* Poor documentation because the requirements of the customers are changing.
* It is very difficult for software developers to accommodate all the changes demanded by the clients.
* After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.
* The client may lose interest in the final product when he or she is not happy with the initial prototype.
* Developers who want to build prototypes quickly may end up building sub-standard development solutions.

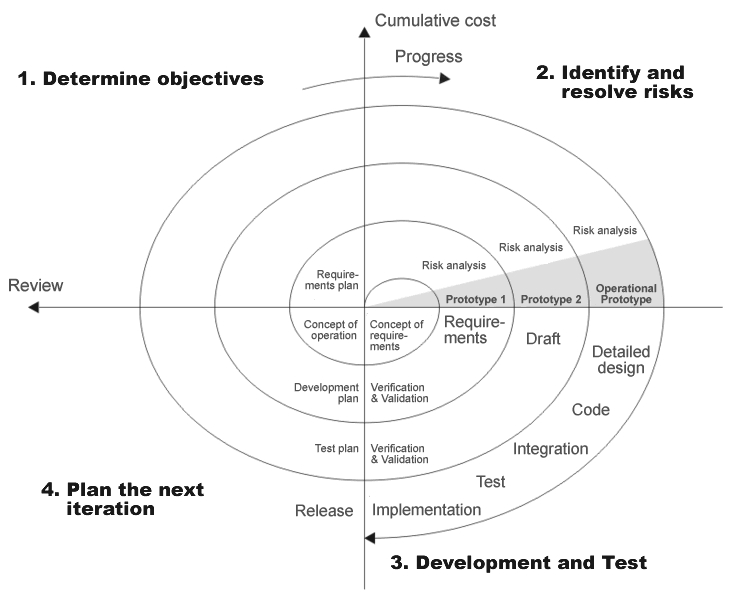
**What is the difference between waterfall and prototype model?**

1. **Budget: high: Used for big project and customer is known so requirements are well cleared at early stage but in prototype it is low. As you are free to design according to your time and customer is unknown.**
2. **Time: High as if big budget but for prototype time according to own and can make minor correction**
3. **Risk Low: Everything is done according to customer but in prototype it is high because no guarantee of requirements fulfilled by customer and it is customized.**
4. **Software life: High because team work, all requirements are fulfilled and well planned but in prototype is it low because limited team work, not so much planning required .**
5. **Maintenance: Low because small upgrade is required but in prototype it is high because frequently some corrections may be required.**

**Spiral Model**

* This model is originally proposed by Boehm.
* It is a meta model as it subsumes all other SDLC model .
* A single loop representation as iterative waterfall model
* It is a combination of both iterative and one of the sdlc model
* At start of each phase as s risk handling technique , it uses the approach of prototyping model to build the prototype.
* It is an evolutionary software process model.
* Risk driven process model generator.
* This model handles software development process in phase manner, each phase being treated as a project work.





The model consider risk , which often goes un-noticed by most other models.The models starts with determining objectives and contraints of the software at the start of one iteration..Next phase is of prototyping the software which includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase planning for the next iteration is done.

Each cycle of the spiral uses the same processes that are as given below:

1. Determine Objective setting /identified alternative and constarints: requirement gathering is done from customer. Objectives are identified, elaborated at start of every phase.
2. Evaluate alternatives ,Identify and resolve risk: the risk assessment and reduction is done. All possible solutions are evaluated and find the best possible solution. We do Risk assessment for best possible solution and prototype is built for the best solution
3. Product development and validation: Identified feature are developed and verified through testing
4. Planning next phase : plan for the next phase development (Requirement plan ,development plan and Integration and test plan)

When to use Spiral model?

1. When project is large
2. When release are required to be frequent
3. When risk and cost evaluation is important
4. When requirements are unclear and complex
5. When changes may require at any time
6. For medium to high risk projects.
7. When creation of a  prototype is appropriate

**Advantage of spiral model**

1. In second quadrant  risk handling is done
2. It is used in large and complex models For Eg if a university is to be developed ,requirements are not known earlier ,slowly the requirements are known so when going through the phase we go on making a plan so it is flexible.
3. Changes can be easily incorporated in later phases with customer satisfaction.After each loop customer can see the product development and give the requirements and make changes.
4. Customers can see the development of the product.

**Disadvantages of spiral model**

* Not suitable for small projects as cost increases because we do not do testing and planning  for small projects as cost increases.
* It is much more complex than other SDLC models because each phase has to iterated many items and any other SDLC model has to be used to built the software.
* Number of phases is unknown at the start of time so estimation is very difficult.All the requirements are not known at starting, as we go through phase requirements changes.
* **Customer Communication:**
  + Task for effective communication between customer and developer
* **Planning :**
  + Task is to define resources, timeline and other project related information
* **Risk analysis:**
  + Task is to assess technical and management risks
* **Engineering:**
  + Task required to build one or more representations of the application
* **Construction and release:**
  + Task to construct, test, install and provide support(documentation, training etc)
* **Customer evaluation:**
  + Task is to obtain customer feedback or evaluation (engineering stage versus implementation stage)

**Difference between spiral and waterfall model**

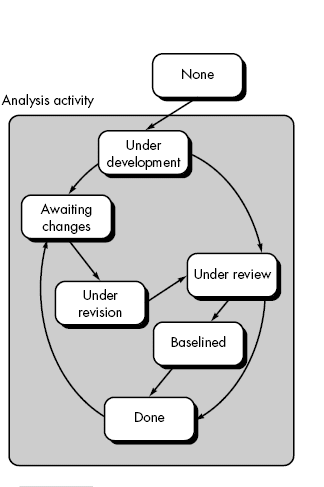
1. **Spiral model is not suitable for small projects but waterfall model is suitable for small projects**
2. **S.piral model is suitable for long and ongoing projects but waterfall model is not suitable for long and ongoing projects**
3. **In spiral process is complex but waterfall model easy to understand**
4. **in spiral model , one can revisited the different phase of software development as many times as one wants during development process but it is not possible in water fall model**
5. **No documentation is required but waterfall model has clear documentation of the entire process.**
6. **Better risk management in spiral model but in waterfall high amount of risk and uncertainty**
7. **Phases are repeated itself but in waterfall phases are processed and completed one at a time.**
8. **Spiral model is flexible with user requirements but in waterfall model requirements once fixed cannot be modified**

**Difference between prototype and spiral model**

|  |  |  |
| --- | --- | --- |
|  | Prototype Model | Spiral Model |
| **1** | **Prototype model is a software development model in which a prototype is built, tested and then refined as per customer needs.** | **Spiral model is a risk-driven software development model and is made with features of incremental, waterfall or evolutionary prototyping models.** |
| **2** | **It is also referred to as rapid or closed ended prototype.** | **It is also referred to as meta model.** |
| **3** | **It does not give emphasis on risk analysis.** | **It takes special care about risk analysis and alternative solution is undertaken.** |
| **4** | **In prototype model, customer interaction is continuous until the final prototype is approved.** | **In spiral model, there is no continuous customer interaction.** |
| **5** | **It is best suited when the requirement of the client is not clear and supposed to be changed.** | **It is best suited when the customer requirements are clear.** |
| **6** | **Cost effective quality improvement is very much possible.** | **Cost effective quality improvement is not possible.** |
| **7** | **In Prototype model, improvement of quality does not increases the cost of product.** | **In Spiral model, improvement of quality can increase the cost of product.** |
|  |  |  |

**Concurrent Development Model**

* Proposed by Davis and Sitaram.
* Defines a series of events that will trigger transition from state to state for each software engineering activity .
* Helps to figure out the actual picture of the state of the project.
* Instead of showing software engineering activities as a sequence of tasks it defines a network of activities existing simultaneous with other activities.
* Events generated in one activity may trigger a state transition of an activity.
* This model is applicable to all types of software development and provides an accurate picture of the current state of a project.



**Component-Based Development**

* This model incorporates many of the characteristics of the spiral mode.
* Commercial-off-the Shelf (COTS) software components, developed by vendors who offer them as product, can be used when software is to be built.
* These targeted functionality with well-defined interfaces that enable the component to be integrated into the software.
* It is evolutionary in nature, demanding an iterative approach to the creation of software.
* Component based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefit.

**The Formal Methods Model**

* When formal methods are used during development, they provide a mechanism for eliminating many problems that are difficult to overcome using other software engineering paradigms.
* Ambiguity, incompleteness and inconsistency can be discovered and corrected more easily through the application of mathematical analysis.
* For the time being it is time consuming and expensive.
* Extensive training is required for necessary background of this model. Difficult to use for the technically unsophisticated customers.

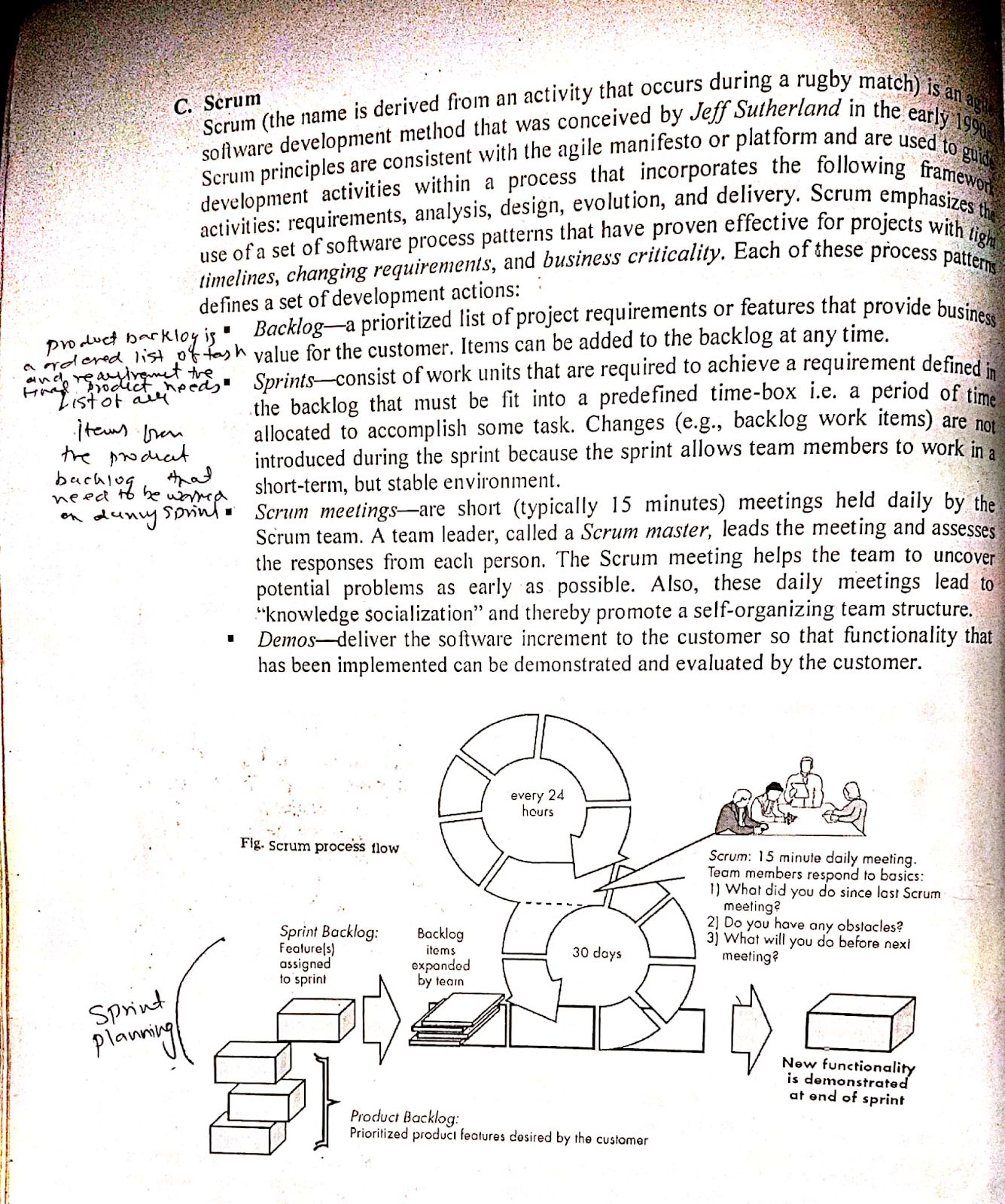
**Aspect-Oriented Software Development**

* Aspect oriented software e development, often referred to as aspect-oriented programming(AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
* A distinct aspect- oriented process has not yet matured. However, it is likely that such a process will adopt characteristics of both the spiral and concurrent process models. The evolutionary nature of spiral is appropriate as aspects are identified and then constructed.

**SCRUM PROCESS**

***Scrum* is an iterative software development and product management method that applies** [**Agile principles**](http://agilemanifesto.org/)

**The basic Scrum principle is iterative development. The entire project timeline is broken down into short iterations called sprints. During each sprint – each usually lasting two weeks – a team commits to deliver on a set of** [**user stories**](https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/)**, concise product feature descriptions. Short iterations ensure that the team prioritizes the features that are actually needed and can be adjusted after receiving feedback. In the best-case scenario, each sprint delivers a small product increment that functions, is QA-approved, and can be handed to stakeholders or end users to get their opinions.**

****