Software Processes

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.

A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

The Software Process Model

A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described. Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering. Some examples of the types of software process models that may be produced are:

1. **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
2. **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
3. **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

There are several various general models or paradigms of software development:

1. **The waterfall approach:** This takes the above activities and produces them as separate process phases such as requirements specification, software design, implementation, testing, and so on. After each stage is defined, it is "signed off" and development goes onto the following stage.
2. **Evolutionary development:** This method interleaves the activities of specification, development, and validation. An initial system is rapidly developed from a very abstract specification.
3. **Formal transformation:** This method is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving.' This means that you can be sure that the developed programs meet its specification.
4. **System assembly from reusable components:** This method assumes the parts of the system already exist. The system development process target on integrating these parts rather than developing them from scratch.

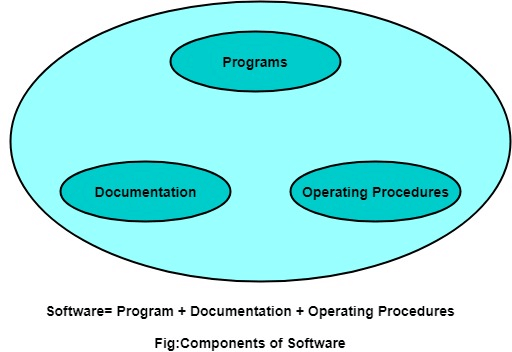
Software Crisis

1. **Size:** Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.
2. **Quality:** Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.
3. **Cost:** Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.
4. **Delayed Delivery:** Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.

Program vs. Software

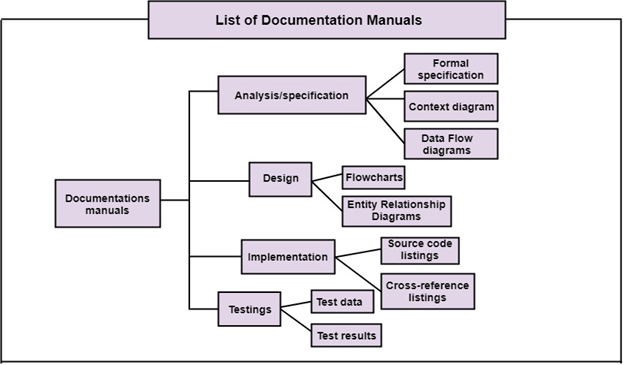
Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.

There are three components of the software as shown in fig:

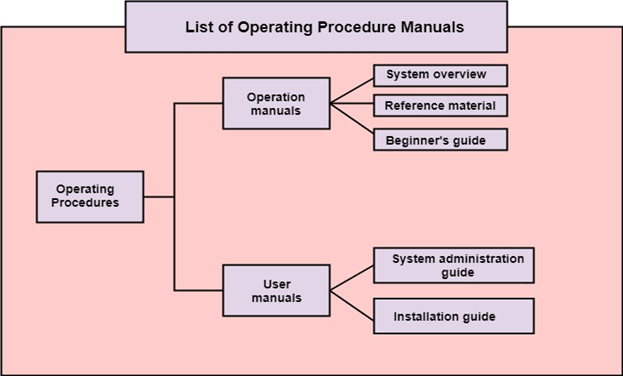


**1. Program:** Program is a combination of source code & object code.

**2. Documentation:** Documentation consists of different types of manuals. Examples of documentation manuals are: Data Flow Diagram, Flow Charts, ER diagrams, etc.



1. **Operating Procedures:** Operating Procedures consist of instructions to set up and use the software system and instructions on how react to the system failure. Example of operating system procedures manuals is: installation guide, Beginner's guide, reference guide, system administration guide, etc.



**Software Process**

A software process (also known as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): Define the main functionalities of the software and the constraints around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conform to its specification and meet the customer's needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.

There are also **supporting activities**such as configuration and change management, quality assurance, project management, user experience.

Along with **other activities**aim to **improve**the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. **Products**: The outcomes of the activity. For example, the outcome of architectural design may be a model for the software architecture.
2. **Roles**: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.
3. **Pre and post conditions**: The conditions that must be true before and after an activity. For example, the pre-condition of the architectural design is the requirements that have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

The software process is complex, it relies on making decisions. There’s no ideal process and most organizations have developed their own software process.

For example, an organization that works on critical systems has a very structured process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective.

**Software Process Models**

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

We’re going to take a quick glance at very general process models. These generic models are abstractions of the process that can be used to explain different approaches to the software development. They can be adapted and extended to create more specific processes.

*Some methodologies are sometimes known as****software development life cycle****(SDLC) methodologies, though this term could also be used more generally to refer to any methodology.*

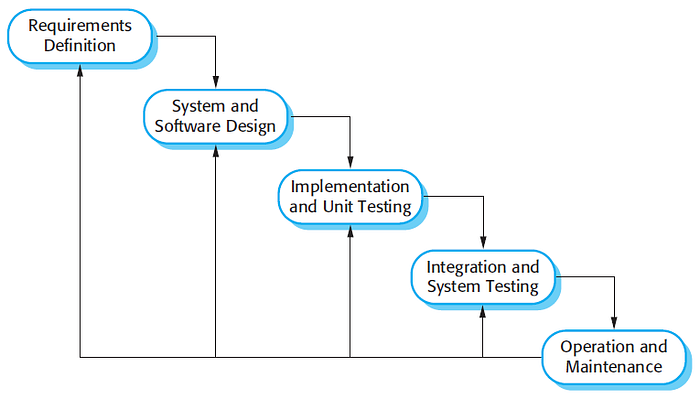
**Waterfall Model**

The waterfall model is a sequential approach, where each fundamental activity of a process is represented as a separate phase, arranged in linear order.

In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process).

*Plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan. While the agile process, planning is incremental and it’s easier to change the process to reflect requirement changes.*

The phases of the waterfall model are: **Requirements, Design, Implementation, Testing, and Maintenance.**



The Waterfall Model — From [Software Engineering, 9th edition, Chapter 2](https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html), by [Ian Sommerville](https://sites.google.com/site/iansommerville/home)

**The Nature of Waterfall Phases**

In principle, the result of each phase is one or more documents that should be approved and the next phase shouldn’t be started until the previous phase has completely been finished.

In practice, however, these phases overlap and feed information to each other. For example, during design, problems with requirements can be identified, and during coding, some of the design problems can be found, etc.

The software process, therefore, is not a simple linear but involves feedback from one phase to another. So, documents produced in each phase may then have to be modified to reflect the changes made.

**When To Use?**

In principle, the waterfall model should only be applied when requirements are well understood and unlikely to change radically during development as this model has a relatively rigid structure which makes it relatively hard to accommodate change when the process is underway.

**Prototyping**

A prototype is a version of a system or part of the system that’s developed quickly to check the customer’s requirements or feasibility of some design decisions.

So, a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.

In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.

While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A software prototype can be used:

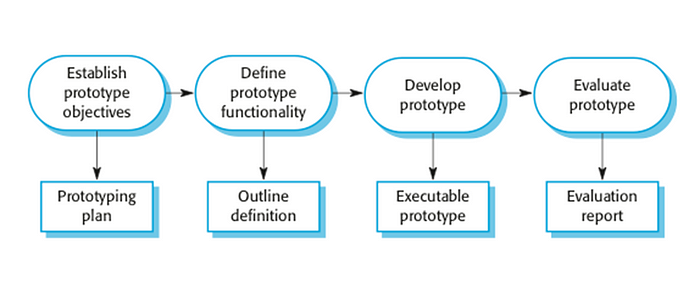
**[1]** In the **requirements engineering**, a prototype can help with the elicitation and validation of system requirements.

It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, and find areas of strength and weakness in the software.

Furthermore, as the prototype is developed, it may reveal errors and in the requirements. The specification may be then modified to reflect the changes.

**[2]** In the **system design**, a prototype can help to carry out design experiments to check the feasibility of a proposed design.

For example, a database design may be prototyped and tested to check it supports efficient data access for the most common user queries.



The process of prototype development — From [Software Engineering, 9th edition, Chapter 2](https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html), by [Ian Sommerville](https://sites.google.com/site/iansommerville/home)

The phases of a prototype are:

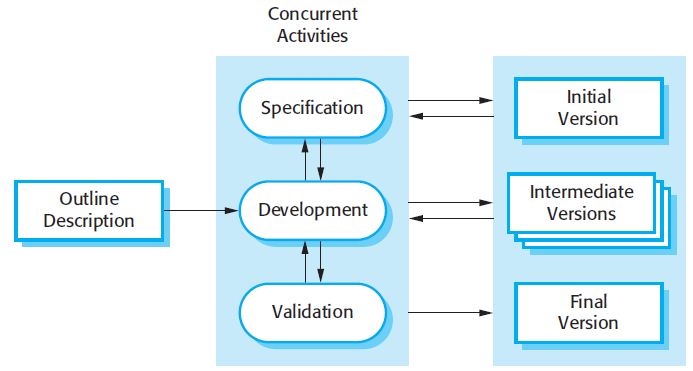
1. **Establish objectives**: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
2. **Define prototype functionality**: Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.
3. **Develop the prototype**: The initial prototype is developed that includes only user interfaces.
4. **Evaluate the** **prototype**: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.

Prototyping is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc).

**Incremental Development**

Incremental development is based on the idea of developing an initial implementation, exposing this to user feedback, and evolving it through several versions until an acceptable system has been developed.

The activities of a process are not separated but interleaved with feedback involved across those activities.



The Incremental Development Model — From [Software Engineering, 9th edition, Chapter 2](https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html), by [Ian Sommerville](https://sites.google.com/site/iansommerville/home)

Each system increment reflects a piece of the functionality that is needed by the customer. Generally, the early increments of the system should include the most important or most urgently required functionality.

This means that the customer can evaluate the system at early stage in the development to see if it delivers what’s required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

**Incremental Vs Waterfall Model**

Incremental software development is better than a waterfall approach for most businesses, e-commerce, and personal systems.

By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.

Compared to the waterfall model, incremental development has three important benefits:

1. The **cost of accommodating changing** customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than what's required with the waterfall model.
2. It’s easier to get **customer feedback** on the work done during development than when the system is fully developed, tested, and delivered.
3. More **rapid delivery** of *useful*software is possible even if all the functionality hasn’t been included. Customers are able to use and gain value from the software earlier than it’s possible with the waterfall model.

**It can be a plan-driven or agile, or both**

Incremental development is one of the most common approaches. This approach can be either plan-driven or agile, or both.

In a plan-driven approach, the system increments are identified in advance, but, in the agile approach, only the early increments are identified and the development of later increments depends on the progress and customer priorities.

**It’s not a problem-free**

But, it’s not problem-free …

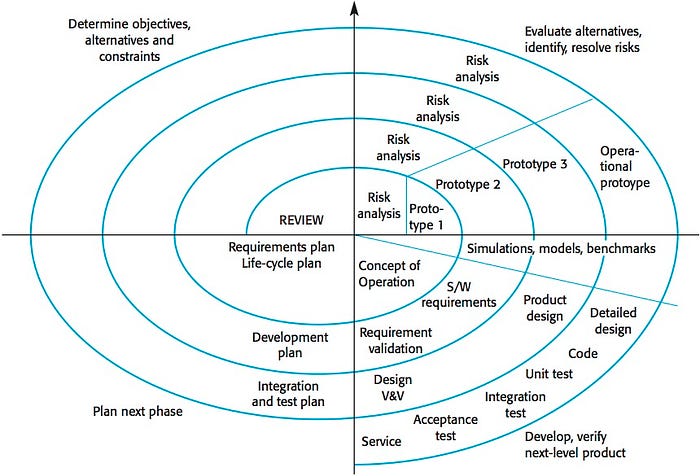
* Some organizations have procedures that have evolved over time, and can’t follow informal iterative or agile processes. For example, procedures to ensure that the software properly implements external regulations.
* System structure tends to degrade as new increments are added and get corrupted as regular changes are incorporated. Even if time and money are spent on refactoring to improve the software, further changes become more difficult and costly.

**Spiral Model**

The spiral model is risk-driven where the process is represented as a spiral rather than a sequence of activities.

It was designed to include the best features from the waterfall and prototyping models, and introduces a new component; risk assessment.

Each loop (from *review*till *service —*see figure below) in the spiral represents a phase. Thus the first loop might be concerned with system feasibility, the next loop might be concerned with the requirements definition, the next loop with system design, and so on.



The spiral model

Each loop in the spiral is split into four sectors:

1. **Objective setting**: The objectives and risks for that phase of the project are defined.
2. **Risk assessment and reduction:** For each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. For example, if there’s a risk that the requirements are inappropriate, a prototype may be developed.
3. **Development and validation:** After risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface then we must prototype the user interface. If the risk is in the development process itself then use the waterfall model.
4. **Planning:**The project is reviewed and a decision is made whether to continue with a further loop or not.

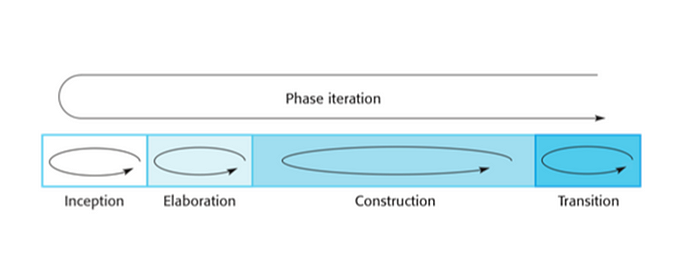
The spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

**Iterative Development**

The iterative development model aims to develop a system by building small portions of all the features, across all components.

We build a product that meets the initial scope and releases it quickly for customer feedback. An early version with limited features is important to establish a market and get customer feedback.

In each increment, a slice of system features is delivered, passing through the requirements till the deployment.



The phases of iterative development

The phases of iterative development are:

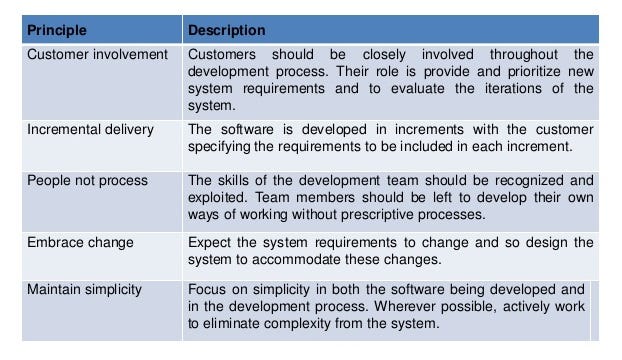
1. **Inception:**The goal is to establish a business case for the system. We should identify all the external entities that will interact with the system, and define these interactions. Then, uses this information to assess the contribution that the system makes to the business. If the contribution is minor, then the project may be canceled.
2. **Elaboration:**We develop an understanding of the problem domain and architecture framework, develop the project plan, and identify risks.
3. **Construction:***Incrementally*fills in the architecture with production-ready code produced from analysis, design, implementation, and testing of the requirements. The components of the system are dependent on each other and they’re developed in parallel and integrated during this phase. On the completion of this phase, you should have a complete working software.
4. **Transition:**We deliver the system into the production operating environment.

*All the phases will be done once, while the construction phase will be incrementally visited for each increment; for each slice of system features.*

**Agile**

Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances.

The agile method refers to a group of software development models based on the incremental and iterative approach, in which the increments are small and typically, new releases of the system are created and made available to customers every few weeks.



The principles of agile

They involve customers in the development process to propose requirements changes. They minimize documentation by using informal communications rather than formal meetings with written documents.

They are best suited for applications where the requirements change rapidly during the development process.

There are a number of different agile methods available such as: Scrum, Crystal, Agile Modeling (AM), Extreme Programming (XP), etc.

**Increment Vs Iterative Vs Agile**

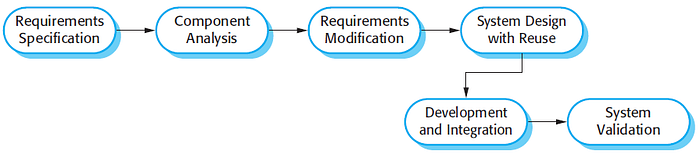
You might be asking about the difference between incremental, iterative, and agile models.

Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.

An agile approach combines the incremental and iterative approach by building a small portion of each feature, one by one, and then both gradually adding features and increasing their completeness.

**Reuse-oriented Software Engineering**

It’s attempting to reuse an existing design or code (probably also tested) that’s similar to what’s required. It’s then modified, and incorporated to the new system.



The Reuse-oriented software engineering model —

Although the initial “requirements specification” phase and the “validation ” phase are comparable with other software processes, the intermediate phases in a reuse-oriented process are different. These phases are:

1. **Component analysis:**A search is made for the components to implement the given requirements specification. Usually, there’s no exact match, and components may only provide some of the functionality required.
2. **Requirements modification**: During this phase, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. If the modifications are impossible, the *component analysis* activity may be re-entered to search for alternative solutions.
3. **System design with reuse:** During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and they will organize the framework accordingly. Some new software has to be designed if some reusable components are not available.
4. **Development and integration:** The components are integrated to create the new system. System integration, in this model, maybe part of the development process rather than a separate activity.

There are basically three types of software components that can be used in a reuse-oriented process:

1. **Web services** that are developed according to well-known service standards and which will become available for remote invocation.
2. **Collections of objects** that are developed as a **package**to be integrated with a component framework such as .NET or Java EE.
3. **Standalone software systems** that are configured for use in a particular environment.

**It has an obvious advantage, But!**

Reuse-oriented software engineering has an obvious advantage of reducing the amount of software to be developed and therefore reducing cost and risks and usually leads to faster delivery.

However, requirements compromises can’t be avoided, which may lead to a system that does not meet the real needs of users.

Furthermore, some control over the system evolution might also be lost as new versions of the reusable components are not under the control of the organization using them.

**Summary**

**Waterfall**

It’s useful when the requirements are clear, or following a very structured process as in critical systems which needs detailed, precise, and accurate documents to describe the system to be produced.

Not good when requirements are ambiguous and don’t support frequent interaction with the customers for feedback and proposing changes. It’s not suitable for large projects that might take a long time to be developed and delivered.

**Prototype**

Again, it’s an early sample or release of a product built to test a concept or to act as a thing to be replicated or learned from.

This is very useful when requirements aren’t clear, and the interactions with the customer and experimenting with an initial version of the software results in high satisfaction and clearance of what is to be implemented.

Its downsides are, good tools need to be acquired for quick development (like coding) in order to complete a prototype. In addition, the costs for training the development team on prototyping may be high.

**Incremental & Iterative**

They’re suited for large projects, less expensive to the change of requirements as they support customer interactions with each increment.

Initial versions of the software are produced early, which facilitates customer evaluation and feedback.

They don’t fit into small projects, or projects that waterfall is best suited for; A structured process with a detailed, and accurate description of the system.

**Spiral**

It’s good for high risky or large projects where the requirements are ambiguous. The risks might be due to cost, schedule, performance, user interfaces, etc.

Risk analysis requires highly specific expertise, and the project’s success is highly dependent on the risk analysis phase. It doesn’t work well for smaller projects.

**Agile**

It suits small-medium size projects, with rapid changes in the requirements as the customer is involved during each phase.

Very limited planning is required to get started with the project. It helps the company in saving time and money (as a result of customer physical interaction in each phase). The daily meetings make it possible to measure productivity.

Difficult to scale up to large projects where documentation is essential. A highly skilled team is also needed.

If team members aren’t committed, the project will either never be completed or fail. And there’s always a limitation in time, like in increments, meetings, etc.

**Process Activities**

The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes.

In the waterfall model, they are organized in sequence, while in incremental development they are interleaved. How these activities are performed might depend on the type of software, people involved in the development, etc.