**I Software Process**

A software process (also knows 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 constrains around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conforms to it’s specification and meets the customer 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 an activity. For example, the outcome of architectural design maybe 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 have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

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 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 about 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.*

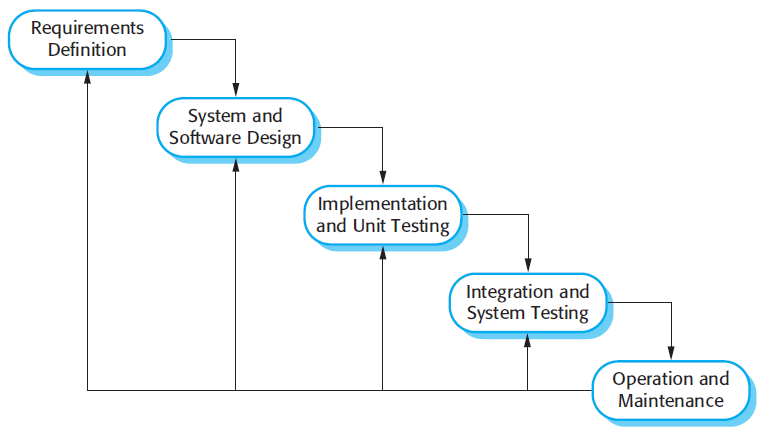
**Waterfall Model**

The waterfall model is a sequential approach, where each fundamental activity of a process 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

**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 in 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:

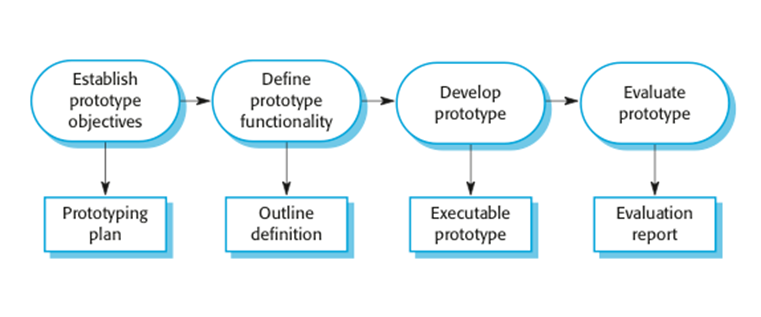
**[1]** In the **requirements engineering**, a prototype can help with the elicitation and validation of system requirements.

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 maybe then modified to reflect the changes.

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

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



The process of prototype development

The phases of a prototype are:

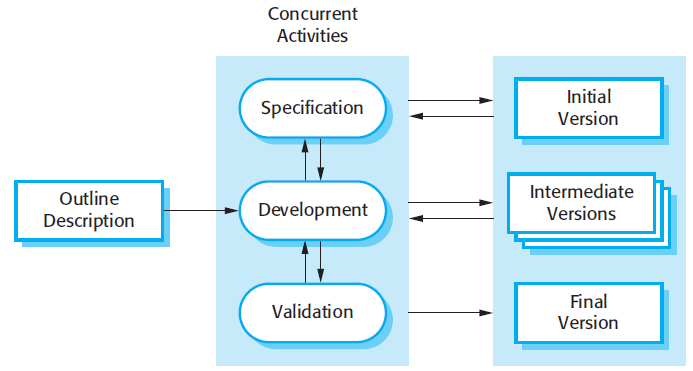
1. **Establish objectives**: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
2. **Define prototype functionality**: Decide what are the inputs and the expected output from a prototype. To reduce 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

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 business, 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 that’s required with 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 a 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 a problem-free …

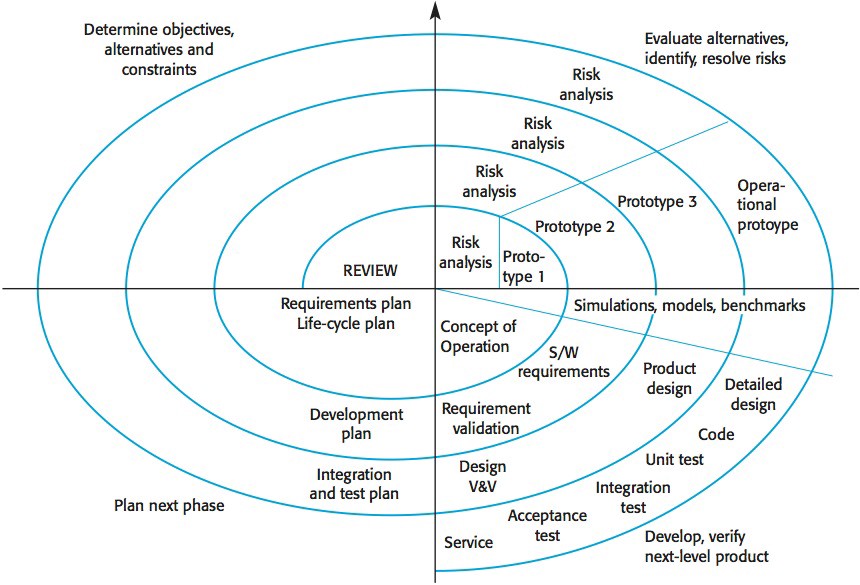
* Some organizations have procedures that have evolved over the time, and can’t follow informal iterative or agile process. For example, procedures to ensure that the software properly implements external regulations.
* System structure tends to degrades as new increments are added and get corrupted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

**Spiral Model**

The spiral model is a risk-driven where the process is represented as 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.

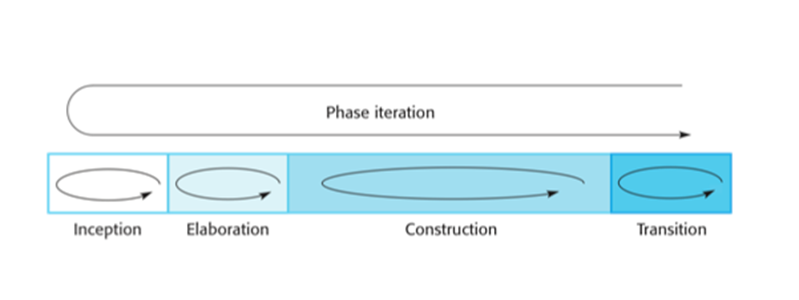
Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

**Iterative Development**

Iterative development model aims to develop a system through building small portions of all the features, across all components.

We build a product which meets the initial scope and release it quickly for customer feedback. An early version with limited features important to establish 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:

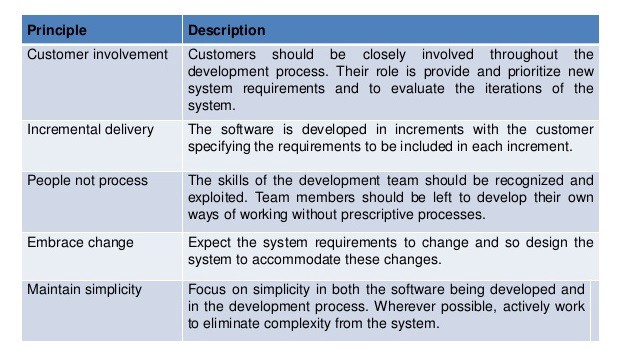
1. **Inception:**The goal is to establish a business case for the system. We should identify all the external entities that will interact with the system, and define these interactions. Then, uses this information to assess the contribution that the system makes to the business. If the contribution is minor, then the project may be cancelled.
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 methods 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 methods

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 application 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.

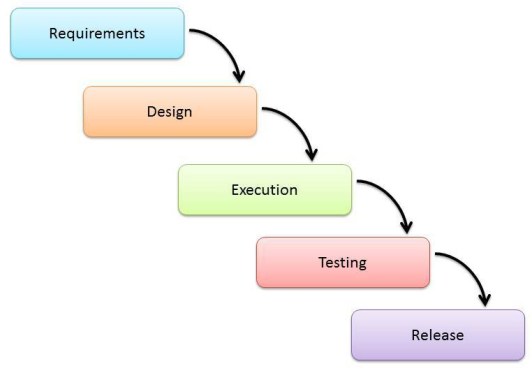
## ii copy Types of Software developing life cycles (SDLC)

* [Waterfall Model](http://melsatar.blog/2018/02/16/the-waterfall-model-a-different-perspective/)
* [V-Shaped Model](https://melsatar.blog/2018/08/27/the-validation-and-verification-model-the-v-model/)
* [Evolutionary Prototyping](http://en.wikipedia.org/wiki/Software_prototyping) Model
* [Spiral](http://en.wikipedia.org/wiki/Spiral_model) Method ([SDM](http://en.wikipedia.org/wiki/Software_development_methodology))
* [Iterative and Incremental](http://en.wikipedia.org/wiki/Iterative_and_incremental_development) Method
* [Agile development](http://en.wikipedia.org/wiki/Agile_software_development)

### Waterfall Model

#### Description

The [Waterfall Model](http://melsatar.blog/2018/02/16/the-waterfall-model-a-different-perspective/) is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.



#### The usage

Projects which not focus on changing the requirements, for example, projects initiated from a request for proposals ([RFPs](http://en.wikipedia.org/wiki/Request_for_proposal)), the customer has a very clear documented requirements

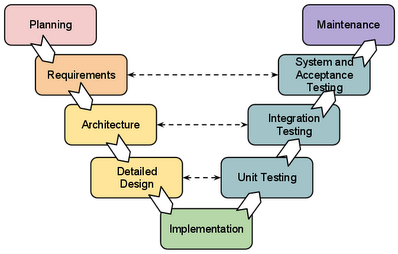
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Easy to explain to the users. * Structures approach. * Stages and activities are well defined. * Helps to plan and schedule the project. * Verification at each stage ensures early detection of errors/misunderstanding. * Each phase has specific deliverables. | * Assumes that the requirements of a system can be frozen. * Very difficult to go back to any stage after it finished. * A little flexibility and adjusting scope is difficult and expensive. * Costly and required more time, in addition to the detailed plan. |

### [V-Shaped Model](https://melsatar.blog/2018/08/27/the-validation-and-verification-model-the-v-model/)

#### Description

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.



#### The usage

* Software requirements clearly defined and known
* Software development technologies and tools are well-known

#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Simple and easy to use * Each phase has specific deliverables. * Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle. * Works well for where requirements are easily understood. * Verification and validation of the product in the early stages of product development. | * Very inflexible, like the waterfall model. * Adjusting scope is difficult and expensive. * The software is developed during the implementation phase, so no early prototypes of the software are produced. * The model doesn’t provide a clear path for problems found during testing phases. * Costly and required more time, in addition to a detailed plan |

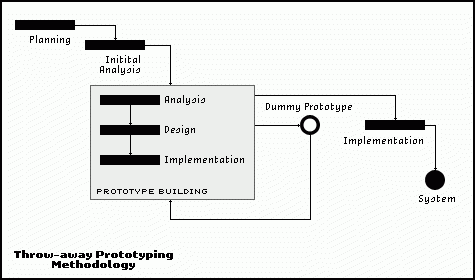
### Prototyping Model

#### Description

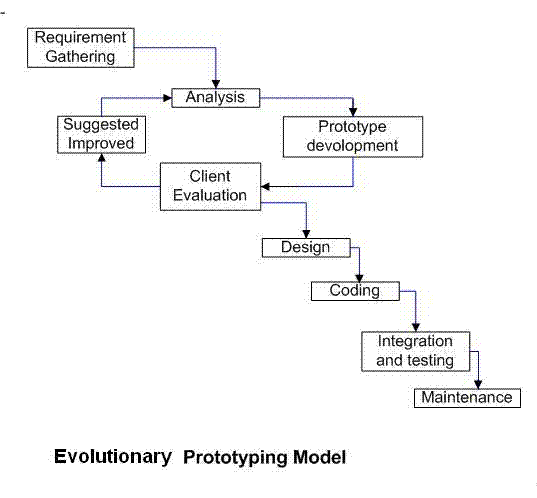
It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development and It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in the waterfall approach and hard to be implemented due to the inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

It has some types, such as:

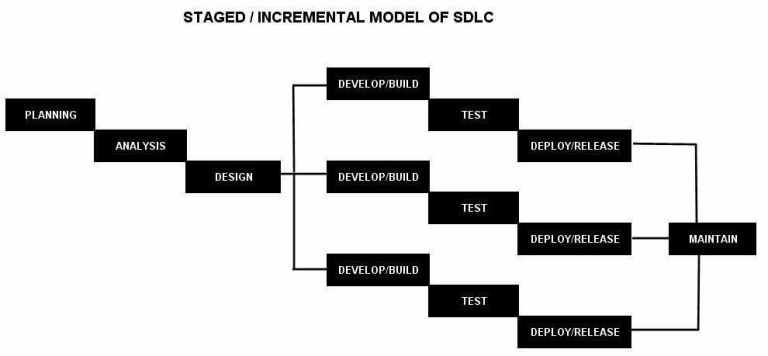
* Throwaway prototyping: Prototypes that are eventually discarded rather than becoming a part of the finally delivered software



* Evolutionary prototyping: prototypes that evolve into the final system through an iterative incorporation of user feedback.



* Incremental prototyping: The final product is built as separate prototypes. In the end, the separate prototypes are merged in an overall design.



* Extreme prototyping: used in web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented

#### The usage

* This process can be used with any software developing life cycle model. While this shall be chosen when you are developing a system has user interactions. So, if the system does not have user interactions, such as a system does some calculations shall not have prototypes.

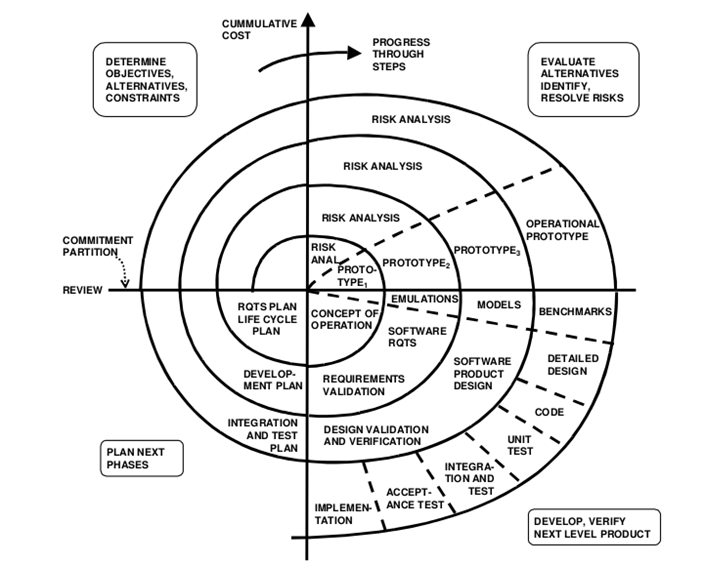
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Reduced time and costs, but this can be a disadvantage if the developer loses time in developing the prototypes. * Improved and increased user involvement. | * Insufficient analysis. User confusion of prototype and finished system. * Developer misunderstanding of user objectives. * Excessive development time of the prototype. * It is costly to implement the prototypes |

### Spiral Model (SDM)

#### Description

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.



#### The usage

It is used in the large applications and systems which built-in small phases or segments.

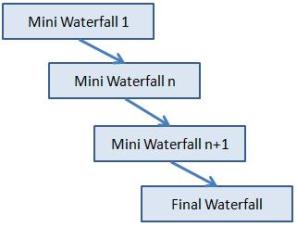
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Estimates (i.e. budget, schedule, etc.) become more realistic as work progressed because important issues are discovered earlier. * Early involvement of developers. * Manages risks and develops the system into phases. | * High cost and time to reach the final product. * Needs special skills to evaluate the risks and assumptions. * Highly customized limiting re-usability |

### Iterative and Incremental Model

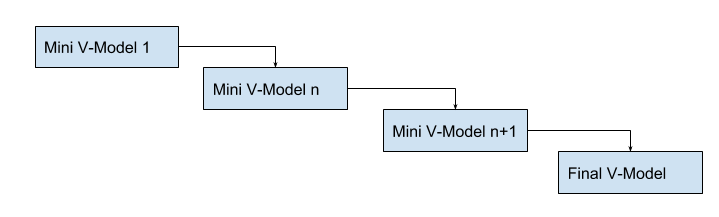
#### Description

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during the development of earlier parts or versions of the system. It can consist of mini waterfalls or mini V-Shaped model



#### The usage

It is used in shrink-wrap application and large system which built-in small phases or segments. Also, can be used in a system has separated components, for example, ERP system. Which we can start with the budget module as a first iteration and then we can start with the inventory module and so forth.



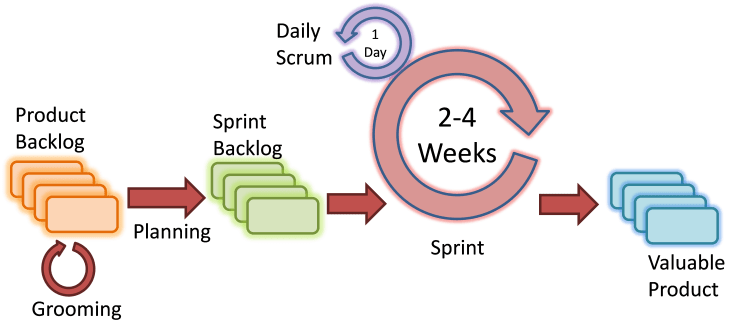
#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Produces business value early in the development lifecycle. * Better use of scarce resources through proper increment definition. * Can accommodate some change requests between increments. * More focused on customer value than the linear approaches. * We can detect project issues and changes earlier. | * Requires heavy documentation. * Follows a defined set of processes. * Defines increments based on function and feature dependencies. * Requires more customer involvement than the linear approaches. * Partitioning the functions and features might be problematic. * Integration between the iterations can be an issue if it is not considered during the development and project planning. |

### Agile Model

#### Description

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.



Scrum Agile Model

#### The usage

It can be used with any type of the project, but it needs more engagement from the customer and to be interactive. Also, we can use it when the customer needs to have some functional requirement ready in less than three weeks and the requirements are not clear enough. This will enable more valuable and workable piece for software early which also increase the customer satisfaction.

#### Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Decrease the time required to avail some system features. * Face to face communication and continuous inputs from customer representative leaves no space for guesswork. * The end result is the high-quality software in the least possible time duration and satisfied customer. | * Scalability. * The ability and collaboration of the customer to express user needs. * Documentation is done at later stages. * Reduce the usability of components. * Needs special skills for the team. |

III COPY

A software process is a structured set of activities required to develop a software system. Note that we are talking about a "software process" -- not a "software *development* process."

There are many different kinds of software processes, but each and every one of them involve these four types of fundamental activities:

* Software **specification** - defining what the system should do;
* Software **design and implementation** - defining the organization of the system and implementing the system;
* Software **validation** - checking that it does what the customer wants;
* Software **evolution** - changing the system in response to changing customer needs.

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. When we describe and discuss software processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities. Process descriptions may also include:

* **Products**, which are the outcomes of a process activity;
* **Roles**, which reflect the responsibilities of the people involved in the process;
* Pre- and post-**conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.

**Plan-driven** processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In **agile** processes, planning is incremental and it is easier to change the process to reflect changing customer requirements. In practice, most practical processes include elements of both plan-driven and agile approaches.

**Software process models**

**The waterfall model**

Plan-driven model. Separate and distinct phases of specification, software design, implementation, testing, and maintenance.

**Incremental development**

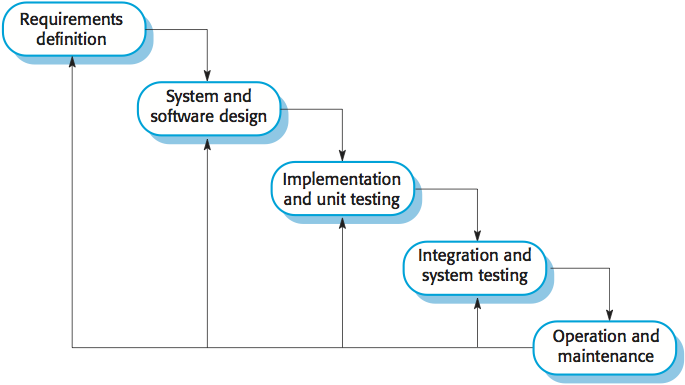
Specification, development and validation are interleaved. The system is developed as a series of versions (increments), with each version adding functionality to the previous version. May be plan-driven or agile.

**Integration and configuration**

Based on the existence of a significant number of reusable components/systems. The system development process focuses on integrating these components into a system rather than developing them from scratch. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

**The waterfall model**



There are separate identified **phases** in the waterfall model:

**Requirements analysis and definition**

The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

**System and software design**

The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

**Implementation and unit testing**

During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

**Integration and system testing**

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

**Operation and maintenance**

Normally (although not necessarily), this is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase. Waterfall model **problems** include:

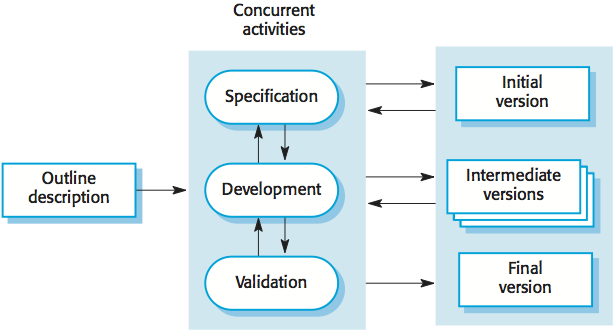
**Difficult to address change**

Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. Few business systems have stable requirements.

**Very few real-world applications**

The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

**Incremental development model**



**Benefits** of incremental development:

**Lower cost of changes**

The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

**Frequent feedback**

It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.

**Faster delivery**

More rapid delivery and deployment of useful software to the customer is possible. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

**Problems** with incremental development (from the management perspective):

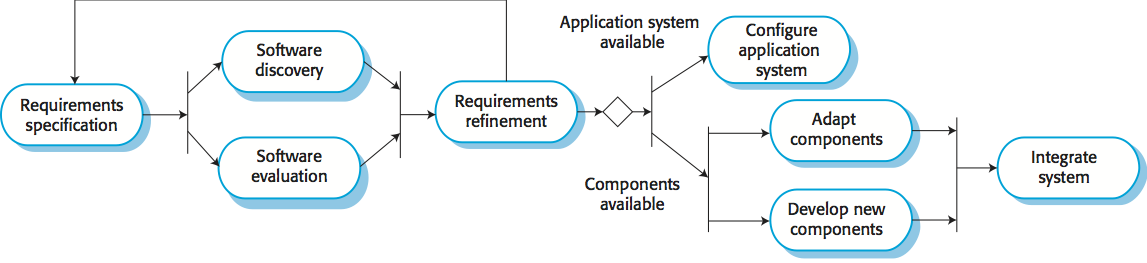
**The process is not visible**

Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

**System structure tends to degrade as new increments are added**

Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

**Integration and configuration**



This approach is based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems. Process stages include:

* Component analysis;
* Requirements modification;
* System design with reuse;
* Development and integration.

Reuse is now the standard approach for building many types of business system.

Types of software components:

* **Web services** that are developed according to service standards and which are available for remote invocation.
* Collections of objects that are developed as a **package** to be integrated with a component framework such as .NET or J2EE.
* Stand-alone **commercial-off-the-shelf** systems (COTS) that are configured for use in a particular environment.

**Software process activities**

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

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, whereas in incremental development they are interleaved.

**Software specification**

The process of establishing what services are required and the constraints on the systemâ€™s operation and development.

Requirements engineering process:

* **Feasibility study**: is it technically and financially feasible to build the system?
* Requirements **elicitation and analysis**: what do the system stakeholders require or expect from the system?
* Requirements **specification**: defining the requirements in detail
* Requirements **validation**: checking the validity of the requirements

**Software design and implementation**

The process of converting the system specification into an executable system.

* **Software design**: design a software structure that realizes the specification;
* **Implementation**: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

* **Architectural design**: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
* **Interface design**: define the interfaces between system components.
* **Component design**: take each system component and design how it will operate.
* **Database design**: design the system data structures and how these are to be represented in a database.

**Software validation**

Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

* **Validation**: are we building the right product (what the customer wants)?
* **Verification**: are we building the product right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity and includes the following stages:

* **Development or component testing**: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.
* **System testing**: testing of the system as a whole, testing of emergent properties is particularly important.
* **Acceptance testing**: testing with customer data to check that the system meets the customer's needs.

**Software evolution**

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

**Coping with change**

Change is inevitable in all large software projects. Business changes lead to new and changed system requirements New technologies open up new possibilities for improving implementations Changing platforms require application changes Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality.

Two strategies to **reduce the costs** of rework:

**Change avoidance**

The software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers.

**Change tolerance**

The process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

**Software prototyping**

A prototype is an initial version of a system used to demonstrate concepts and try out design options. A prototype can be used in:

* The requirements engineering process to help with requirements elicitation and validation;
* In design processes to explore options and develop a UI design;
* In the testing process to run back-to-back tests.

**Benefits** of prototyping:

* Improved system usability.
* A closer match to users' real needs.
* Improved design quality.
* Improved maintainability.
* Reduced development effort.

Prototypes may be based on rapid prototyping languages or tools. They may involve **leaving out functionality**:

* Prototype should focus on areas of the product that are not well-understood;
* Error checking and recovery may not be included in the prototype;
* Focus on functional rather than non-functional requirements such as reliability and security.

Prototypes should be **discarded** after development as they are not a good basis for a production system:

* It may be impossible to tune the system to meet non-functional requirements;
* Prototypes are normally undocumented;
* The prototype structure is usually degraded through rapid change;
* The prototype probably will not meet normal organizational quality standards.

**Incremental development/delivery**

Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality. User requirements are prioritized and the highest priority requirements are included in early increments. Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

**Advantages** of incremental delivery:

* Customer value can be delivered with each increment so system functionality is available earlier.
* Early increments act as a prototype to help elicit requirements for later increments.
* Lower risk of overall project failure.
* The highest priority system services tend to receive the most testing.

Incremental delivery **problems**:

* Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
* The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

**Process improvement**

Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

**Process maturity approach**

Focuses on improving process and project management and introducing good software engineering practice. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.

**Agile approach**

Focuses on iterative development and the reduction of overheads in the software process. The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

**Process improvement activities** form a continuous cycle with a feedback loop:

* **Measure** one or more attributes of the software process or product. These measurements forms a baseline that help decide if process improvements have been effective.
* **Analyze** the current process and identify any bottlenecks.
* **Change** the process to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

**Process measurement**

* Wherever possible, quantitative process data should be collected.
* Process measurements should be used to assess process improvements.
* Metrics may include:
  + Time taken for process activities to be completed, e.g. calendar time or effort to complete an activity or process.
  + Resources required for processes or activities, e.g. total effort in person-days.
  + Number of occurrences of a particular event, e.g. number of defects discovered.

**The SEI capability maturity model**

* **Initial:** Essentially uncontrolled
* **Repeatable:** Product management procedures defined and used
* **Defined:** Process management procedures and strategies defined and used
* **Managed:** Quality management strategies defined and used
* **Optimizing:** Process improvement strategies defined and used