**ST.** XAVIER’S **COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Software Engineering**

**Theory Assignment #3**

**Submitted By**

**Aishwarya Rai**

**014BIM002**

**Submitted To**

|  |  |
| --- | --- |
| **Signature** | **Remarks** |
| **Mr. BalKrishna Subedi**  **Lecturer**  **Dept. of Computer Science** |  |  |

**Software process model**

Software process implies the phases involved for the development of the software.

The software process model represents various perspective of software.

Software process model often known as the product life cycle describes the sequence of phases for the entire lifetime of a product covering everything from the initial commercial idea until the final de-installation or disassembling of the product after its use. (unknown, 2013)

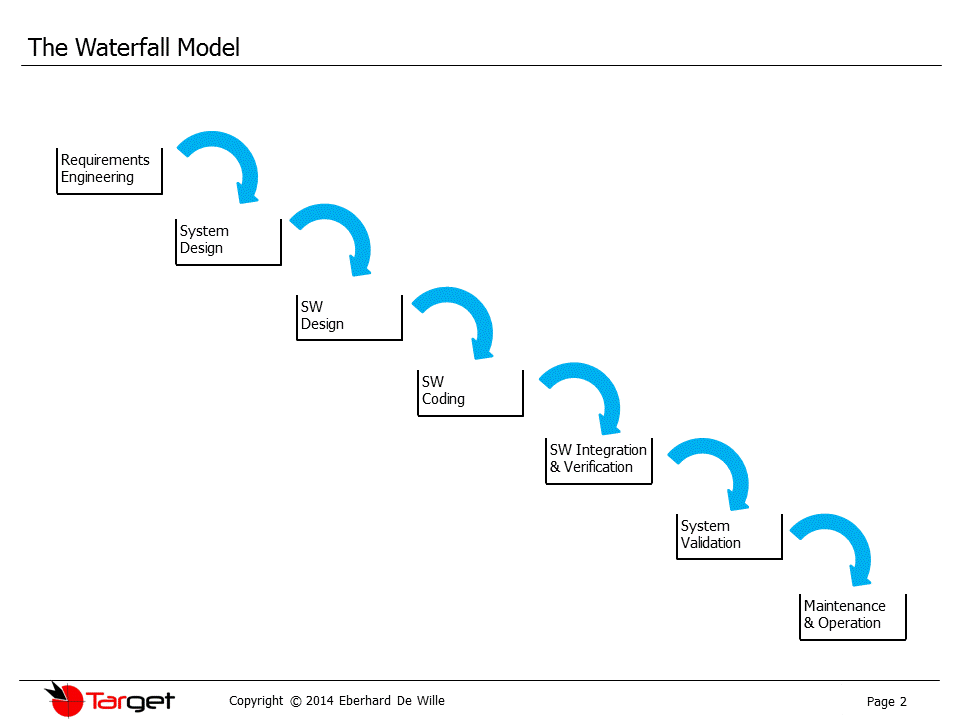
Furthermore, a development strategy that encompasses the process, methods, and tools layers and the generic is often referred to as software process model. The model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverable that are required. (unknown, http://www.the-software-experts.com/e\_dta-sw-process-model.php)

**PROCESS MODELS:**

**The Waterfall Model**

The waterfall model is believed to have been the first process model which was introduced and widely followed in software engineering. The innovation was that the first time software engineering was divided into separate phases.

The different phases of software engineering were identified and simply cascaded in each other, allowing for loops in case it was found in a subsequent phase that the previous phase was not done properly, emphasized to be the waterfall model.

  
  
The phases of "The Waterfall Model" are:

1. Requirement Analysis & Definition:

All the requirement model studies, analyses and collects the valid requirements of the end users of the software.

Like in other process models requirements are split up in functional requirements and constraints which the system has to fulfil.

The aim is to generate a Requirements Specification Document which is used as an input for the next phase of the model.

1. System Design:

The system has to be properly designed before any implementation is started.

This involves an architectural design which defines and describes the main blocks and components of the system, their interfaces and interactions.

By this the needed hardware is defined and the software is split up in its components.E.g. this involves the definition or selection of a computer platform, an operating system, other peripheral hardware, etc

The software components have to be defined to meet the end user requirements and to meet the need of possible scalability of the system.

Usually in this phase various documents are generated, one for each discipline, so that the software usually will receive a software architecture document.

1. Software Design:

Based on the system architecture which defines the main software blocks the software design will break them further down into code modules.

The interfaces and interactions of the modules are described, as well as their functional contents.

All necessary system states like startup, shutdown, error conditions and diagnostic modes have to be considered and the activity and behaviour of the software has to be defined.

The output of this phase is a Software Design Document which is the base of the following implementation work.

1. Coding:

Based on the software design document the work is aiming to set up the defined modules or units and actual coding is started.

The system is first developed in smaller portions called units. They are able to stand alone from an functional aspect and are integrated later on to form the complete software package.

1. Software Integration & Verification:

Each unit is developed independently and can be tested for its functionality. This is the so called Unit Testing.

It simply verifies if the modules or units to check if they meet their specifications. This involves functional tests at the interfaces of the modules, but also more detailed tests which consider the inner structure of the software modules.

During integration the units which are developed and tested for their functionalities are brought together. The modules are integrated into a complete system and tested to check if all modules cooperate as expected.

1. System Validation:

After successfully integration including the related tests the complete system has to be tested against its initial requirements.

This will include the original hardware and environment, whereas the previous integration and testing phase may still be performed in a different environment or on a test bench.

1. Operation & Maintenance:

The system is handed over to the customer and will be used the first time by him. Naturally the customer will check if his requirements were implemented as expected but he will also validate if the correct requirements have been set up in the beginning.

In case there are changes necessary it has to be fixed to make the system usable or to make it comply to the customer wishes.

In most of the "Waterfall Model" descriptions this phase is extended to a never ending phase of "Operations & Maintenance". All the problems which did not arise during the previous phases will be solved in this last phase.  
  
**The weakness of the Waterfall Model is at hand:**

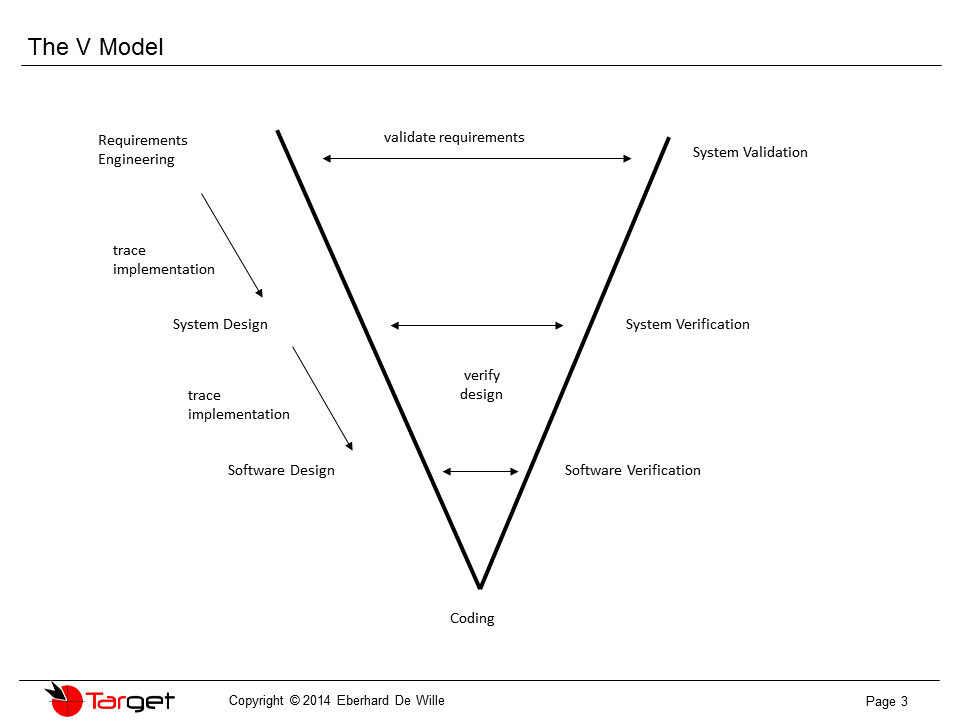
It is very important to gather all possible requirements during the first phase of requirements collection and analysis. If not all requirements are obtained at once the subsequent phases will suffer from it. Reality is that only a part of the requirements is known at the beginning and a certain percentage will be gathered during the complete development time.

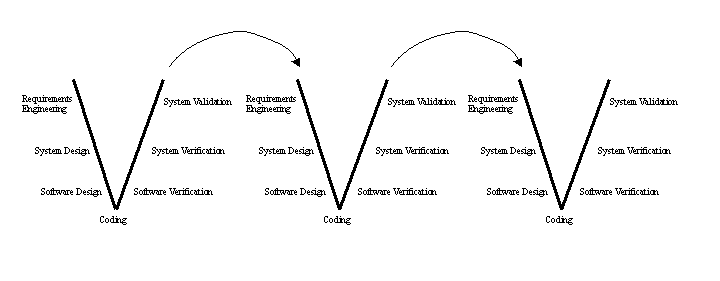
Iterations are only meant to happen within the same phase or at best from the start of the subsequent phase back to the previous phase. If the process is kept according to the school book this tends to shift the solution of problems into later phases which eventually results in a bad system design. Instead of solving the root causes the tendency is to patch problems with inadequate measures.

There may be a very big "Maintenance" phase at the end. The process only allows for a single run through the waterfall. Eventually this could be only a first sample phase which means that the further development is squeezed into the last never ending maintenance phase and virtually run without a proper process.

**The V model:**

A further development of the waterfall model led to the so called "V-Model". If you look at it closely the individual steps of the process are almost the same as in the waterfall model. However, there is one big difference. Instead of going down the waterfall in a linear way the process steps are bent upwards at the coding phase, to form the typical V shape. The reason for this is that for each of the design phases it was found that there is a counterpart in the testing phases which correlate to each other.

  
The time in which the V-model evolved was also the time in which software testing techniques were defined and various kinds of testing were clearly separated from each other. This new empasis on software testing (of course along with improvements and new techniques in requirements engineering and design) led to the evolution of the waterfall model into the V-model. The tests are derived directly from their design or requirements counterparts. This made it possible to verify each of the design steps individually due to this correlation.   
  
Another idea evolved which was the traceability down the left side of the V. This means that the requirements have to be traced into the design of the system, thus verifying that they are implemented completely and correctly. Another feature can be observed when you compare the waterfall model to the V-model. The "Operation & Maintenance" phase was replaced in later versions of the V-model with the validation of requirements. This means that not only the correct implementation of requirements has to be checked but also if the requirements are correct. In case there is the need of an update of the requirements and subsequently the design and coding, etc. there are two options. Either this has to be treated like in the waterfall model in a never ending maintenance phase, or in going over to another V-cycle. The earlier versions of V-models used the first option. For later versions a series of subsequent V-cycles was defined, as shown in the below diagram:

  
  
This idea also correlated with the established sample phases for products as it is present in many industries. One of the cascaded V-cycles became the V-cycle of a sample phase. In addition to this the V-cycles were tailored. This means that in earlier sample phases not all the intermediate work products and process step were established to their full extend but it was simply reduced to what makes sense. By these measures the V-model became a usable process model. It does not consider every detail and possibility but evaluated over a multitude of projects in various industries it proved its usability.

**The Spiral model:**

**Process:**

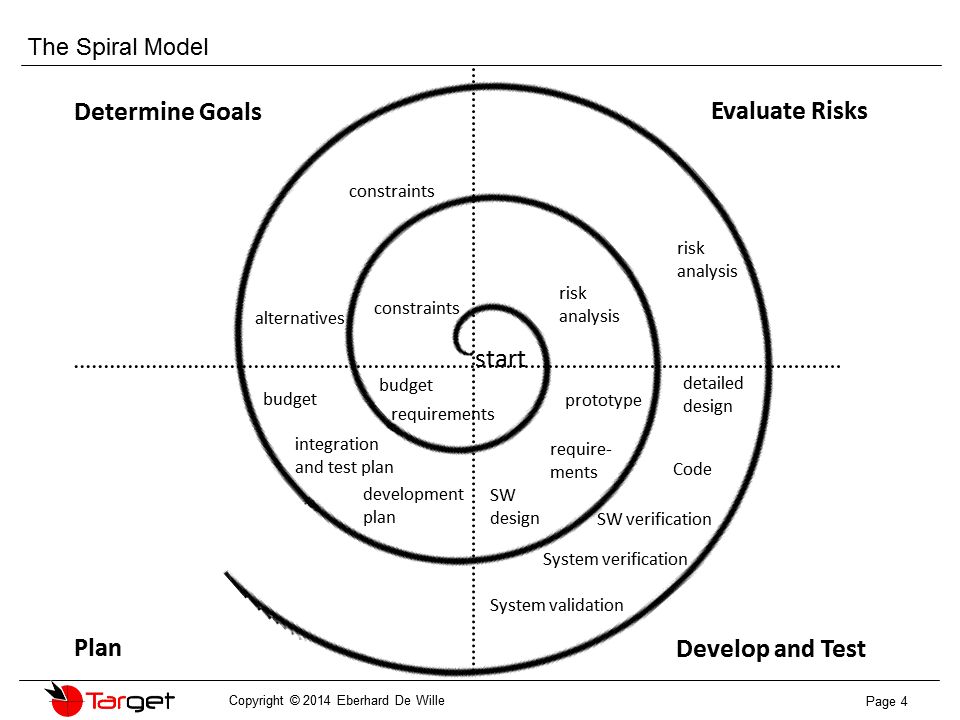
The process begins at the center position.

From there it moves clockwise in traversals.

Each traversal of the spiral usually results in a deliverable.

It is not clearly defined what this deliverable is.

This changes from traversal to traversal.

For example, the first traversals may result in a requirement specification. The second will result in a prototype, and the next one will result in another prototype or sample of a product, until the last traversal leads to a product which is suitable to be sold. Consequently the related activities and their documentation will also mature towards the outer traversals. E.g. a formal design and testing session would be placed into the last traversal.  
  
  
  
There are different instances of this model and you can find them with 3 to 6 task regions. The above picture shows 4 task regions.   
These regions are:

* The planning task - to define resources, responsibilities, milestones and schedules.
* The goal determination task - to define the requirements and constraints for the product and define possible alternatives.
* The risk analysis task - to assess both technical and management risks.
* The engineering task - to design and implement one or more prototypes or samples of the application.

The most outstanding distinction between the spiral model and other software models is the explicit risk evaluation task. Although risk management is part of the other processes as well, it does not have an own representation in the process model. For other models the risk assessment is a sub-task e.g. of the overall planning and management. Further there are no fixed phases for requirements specification, design or testing in the spiral model. Prototyping may be used to find and define requirements. This may then be followed by "normal" phases as they can be found in other process models to handle design and testing.   
  
The advantages of t**h**e spiral model are

* it reflects the development approach in many industries much better than the other process models do. It uses a stepwise approach which e.g. goes hand in hand with the habit of maintaining a number of hardware sample phases in cases where the product to be produced is not only software for a given environment, but also contains the development of hardware.
* This way the developers and the customer can understand and react much better to risks in the evolutionary process.
* By having an iterative process which reduces formalisms and activities in the earlier phases the use of resources is optimized.
* Further, any risks should be detected much earlier than in other process models and measures can be taken to handle them.

The disadvantages of the spiral model are

* the risk assessment is rigidly anchored in the process. First of all it demands risk-assessment expertise to perform this task and secondly in some cases the risk assessment may not be necessary in this detail. For completely new products the risk assessment makes sense.
* Also if you think of the multitude of carry over projects in many industries i.e. applying an already developed product to the needs of a new customer by small changes, the risks are not a subject generating big headaches. Generally speaking the spiral model is not much esteemed and not much used, although it has many advantaged and could have even more if the risk assessment phases would be tailored to the necessary amount.

**References:**

(unknown, http://www.onlineclassnotes.com/2013/01/what-is-software-process-model.html, 2013)

(unknown, http://www.the-software-experts.com/e\_dta-sw-process-model.php)

(http://www.onlineclassnotes.com/2013/01/what-is-software-process-model.html, 2013)