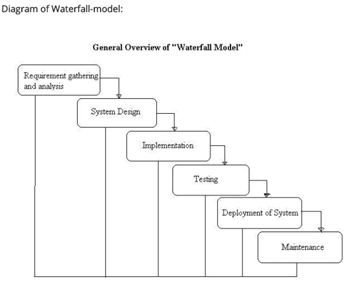
**Kenji Leong**

**Software Development Life Cycle (SDLC) + 3 development models**

Software Development Life Cycle (SDLC) + 3 development models

SDLC also known as application development life-cycle is a process to ensure good software is build and typically it includes 5 phases starting with the analysis and requirements gathering and ending with the implementation. Firstly, requirement gathering is a phase that is critical to the success of the project as expectation is set by the sponsor or your team. Secondly, analysis and design phase is when the team choose the software that they want to use. Thirdly, Coding is the longest phase due to enhancement needed from the sponsor. Fourthly, testing phase includes user acceptance testing to ensure the system meets their expectation. Lastly, deployment phase is when the team roll out the system to the public.

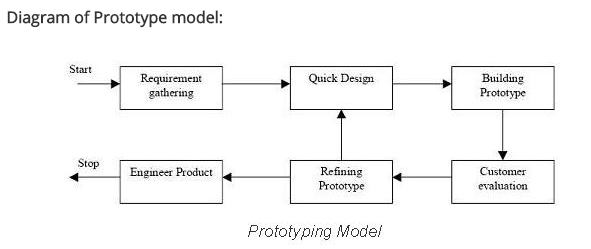
Besides, there are many different models that can be used to achieve different required objectives. Firstly, Waterfall model also known as linear-sequential life cycle model is a traditional model that is well structured. It uses a sequential model as it flows downwards where each phase must be completed fully before the next phase can begin. This model is usually used for small project that have no uncertain requirements.



As shown above there are 6 steps required for a waterfall model. For the advantages, it is easy to manage because of the rigidity of the model as each phase has a specific deliverable and a review process. Also, since the steps are completed one at a time. As a result, phases does not overlap with each other. However, the disadvantages include where once it reaches the testing stage, it is very troublesome and difficult to change something that is in the concept stage and it not a suitable for long term project due to many changes that need to be made during the process.

Thus, a waterfall model is best used only when the requirements are very clear and fixed where no changes will be made and also the size of the project is small.

Besides, Prototype model is where instead of freezing the requirements before any coding can be done. A throwaway prototype is built so that the developers can better understand the requirements that are needed for the required systems. By doing so, this can help verify requirement that are unclear.



As shown above, a throwaway model is always being used to get feedback from the customer or sponsor and the team will then refine the product based on the feedback that was given. The advantage of using a prototype model includes the users are actively involved in the development since they have to provide feedback and errors such as bug can be discovered earlier so that the development team will have more time to solve these problems.

On the other hand, the disadvantages is that it can increase the complexity of the system. For example, as more feedback is given, there will be more requirements needed to satisfy the demand. As a result, the scope may expand beyond the original plan.

Thus, a prototype model should only be used when the system has a very high amount of interaction with end users such as online system where reviews and feedback will be consistently done by customers.

Lastly, agile model is a type of incremental model where for each phases a software is being built and tested. Overtime, more functionality will then slowly be added into the software until the system is completed. For example, SCRUM is one of the most popular agile framework.



There are three roles in this framework. Firstly, the product owner determine what need to be done by creating a wish list called a product backlog. Next, the development team then choose some of things that need to be done on the wish list and decides on how to implement these things. After that, the scrum master will ensure to keep the development team focused on their goal. Lastly, at the end of each sprint which is one round of work, the team will have to complete the task they chosen from the wish list and repeat the cycle again.

The advantage of using an agile model include is that the working software is developed frequently and the team is flexible to any late changes at any time of time period of the project. However, one of the disadvantage is that only senior programmers can handle the development process because this would require many experience as decision has to made fast. As a result, it is not recommended for newbie to be in charge of this model.

Thus, an agile model should only be used when the sponsor or clients are always changing their requirement based on the feedback given from the customer and also changes can be easily made since it is flexible due to the frequency of new functionality being produced.

**Jia Cheng**

**Software Development Life Cycle (SDLC) + 3 development models**

Software Development Life Cycle is basically the process from start to finish of the software, starting from an idea to actual software deployment. This plan consist of details regarding the maintenance, enhancement and the likes for the project. Which also could be used to further improve the quality of the software as well as the developmental process. With this said, there are different SDLC models that are used and followed in industries. All of them shares fairly similar phases. Like

* Waterfall Model
  + As the name implies, the software development processes’ progress are seen flowing downwards like a cascading waterfall through different phases like Requirement Analysis, System Design, Implementation, Testing, Deployment and Maintenance. The phases does not overlap with one another. Usually one phase paves the way to the subsequent phase. As the name Waterfall implies, it’s very hard to reverse the phase since it only goes in one direction.
  + It’s best used for short projects with clear requirements. However, it does bring about many uncertainty as no working software is produced until late stage of the cycle.
  + Here’s an image for better visual: 

**Requirement Analysis -**

Here is where you gather all the different requirements of the system that is wanted by the client to be developed in this phase. This includes coming up with the Software Requirement Specification to be documented

**System Design -**

This phase is where the system design is to be prepared. This can be the architecture of the design. Things include like hardware requirements and the likes.

**Implementation -**

As the name says, it means to code out the system as a prototype first. This prototype, usually called a unit is to be integrated in the next phase to be tested for its performance

**Integration with Testing -**

Integrating into the current system and also tested for its robustness and reliability. As well as finding any potential bugs and errors.

**Deployment -**

This is not to confuse with Implementation. It means to deploy the system to the client or market.

**Maintenance -**

This means further enhancement to the product after release. As such, based on feedbacks and reports, updates could be implemented for the market and client for enhancements, or maintenance. Things like new operating system platform doesn’t run well with the software in the market etc

* Iterative Model
  + Basically, this model starts with a very small set of software requirements and through many repeated enhancements, it will then be deployed until the complete system is ready. There can be many iteration of the software development cycles at any time.
  + It’s best used when there is time to the market constraint. However, it does also allow results and progress to be tracked periodically. Though it may need more resources and that the Endpoint of the iteration may not be known.
  + Here’s an image for better visual: 
  + As you can see, in this model, the whole requirement is developed into various builds and all the subsequent stages of the SDLC. With this in mind, there will need to be vigorous validation of the requirements and verification of all the different version of build. This process could also be known as an “Evolutionary Acquisition”. During every subsequent release of the module, there will be functions approved and tested, added from the previous build
* Big Bang Model
  + To very simply put, this model is where there is not set of instruction and guidelines. The development just start with the resources and effort as the input, with the end product not be as per the customer’s requirement. Very little planning is required in this approach.
  + It’s best used for small development teams or small projects. It allows easier management and flexibility. Albeit with no planning required, it has its own uncertainty. It can turn out wrong if the requirements are misunderstood.
  + There are actually no model from this as they do not follow any specific progress. So it can start with any requirement. However, this would mean that, the end product might not be completed or ideal.

In a typical SDLC environment, there are different phases. They are Planning, Defining, Designing, Building, Testing, Deployment in the cycle. For a better visual, see the pic:

Also take note that, for different industry and culture, the stages of the process of the SDLC can and might be termed differently. For example, some might put the Planning and Defining of this SDLC example together, making just 5 phases or perhaps, maybe say the “Designing” phase is termed as “Analysis and SRS Design” phase.

**Planning (Requirement Analysis) -**

It is the most basic steps in all of the SDLC, the different model of SDLCs like waterfall, Big-Bang model might place little to high emphasis on this phase. As the stage imply, it is where we take in input from our client and make sure our objectives and their are on the same page. Per se, this is very important in a Waterfall model as it is very uncertain of the following events as there are no working products/builds until the very late stages of it’s cycle. For this, it is also planning up and scheduling what to do next for the next upcoming stages.

**Defining Requirements -**

For this stage, it is when we come up with different documents to ensure and make sure that the developers and the client are on the same page. This stage defines all the different product requirements and all to get them approved by the customer. The documents created in his stage is usually what the developers with design and develop for this entire progress. This document is also known as Software Requirement Specification (SRS). There are also many more like Terms of Reference and Project Plan (also known as Project Charter).

**Designing -**

This stage is where you design the software/product’s architecture. This design also shows how the data flow from one place to another. Basically, it shows the entire design approach for the architecture. Adding on, with the previous stage’s Software Requirement Specification, this stage will come up with the structure of the software documented in Design Document Specification. This Design Document Specification is reviewed by many of the important people in the project. They will review the specification based on risk, reliability, modularity, constraints, usability and the likes.

**Building -**

As the name imply, it’s the stage where the actual coding starts and the product is built. As much so, this building of the code’s architecture would be smooth if the Design Requirement Specification are well informed and organized. Furthermore, the Design Requirement Specification have to be followed closely like a guideline as it is defined by their organization. The programming language could be such as C, C++, Pascal, Java, PHP etc. The programming language used here is dependent on the architecture the software is based off.

**Testing -**

This always comes after building. It’s testing the software built to see its performance. This stage will test the prototype built through many series of reports and test. Many of which will be tracked and fixed until it reaches the standard defined in the Software Requirement Specification

**Deployment -**

Simply enough, this means that the product will be released to the market formally. Usually, with the base solid software created and deployed, there can be further enhancements made to the software. This is also known as updates, which can be done to enhance different functionality as well as maintenance for the existing users through feedbacks.

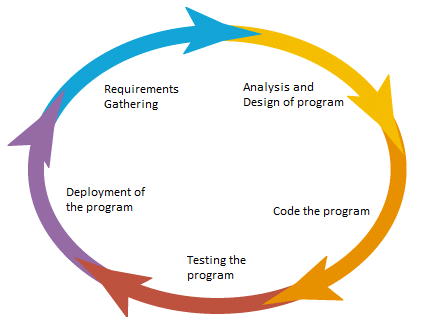
**Goh Mao Cheng**

Software Development Life Cycle

## Description of Software Development Life Cycle

The Software Development Life Cycle (SDLC) is the term used in software engineering which includes a well-defined, structured sequential phases in software engineering to build a good, working software. What goes on from start of project to deployment of software system. That is, from inception of idea to deployment. The different phases of SDLC are Requirement engineering, Analysis & Design, Implementation (coding), Testing and Deployment

## Phases in Software Development Life Cycle



*Figure 1: Software Development Life Cycle Phases*

The sequence of stages that begins from the creation of a project and builds up to a finished product are:

1. Requirements Gathering/ Requirements Engineering
   * Finds out what the user (client or customer) wants the software to do
   * This phase will be covered in detail in other lectures
   * Pitfalls:

-Fuzzy, difficult to specify precisely

-Incomplete requirements

-Unstable requirements

-Misunderstanding user goals

1. Analysis and Design
   * Analysis: Uses a combination of text and diagrammatic forms to depict requirements
   * Design: Produce a representation of an entity that will later be built. Include architectural design, user interface design, database design etc
   * Pitfalls:

-Not meeting the requirements

-Compromised design due to cost / time

-Different designs – which is better?

-Explosion of *derived* requirements

-Poor, ad-hoc design

1. Implementation (Coding)
   * Detailed designs converted into instructions written in the programming language
   * Pitfalls:

-Tied down to certain technology

-Future enhancements not considered

-Different developers / teams

-Bad compromises made due to deadline

-Conflicting choices

-Cost vs Time

-Speed vs Memory Usage

1. Testing
   * Ensure software is reliable
   * Ensure software meets users’ needs
   * Covered in detail in other lectures
   * Pitfalls:

-Testing done by persons who are not developers

-Testing an after-thought

-Testing done late in project

-Insufficient testing   
-(test cases / test data)

1. Deployment
   * Application distributed among a group of selected customers prior to official release
   * Application delivered to customer (may involve training)
   * Pitfalls:

-Incorrect expectation of production environment

-Not testing out production environment

-Mismatch between development and   
 production environment

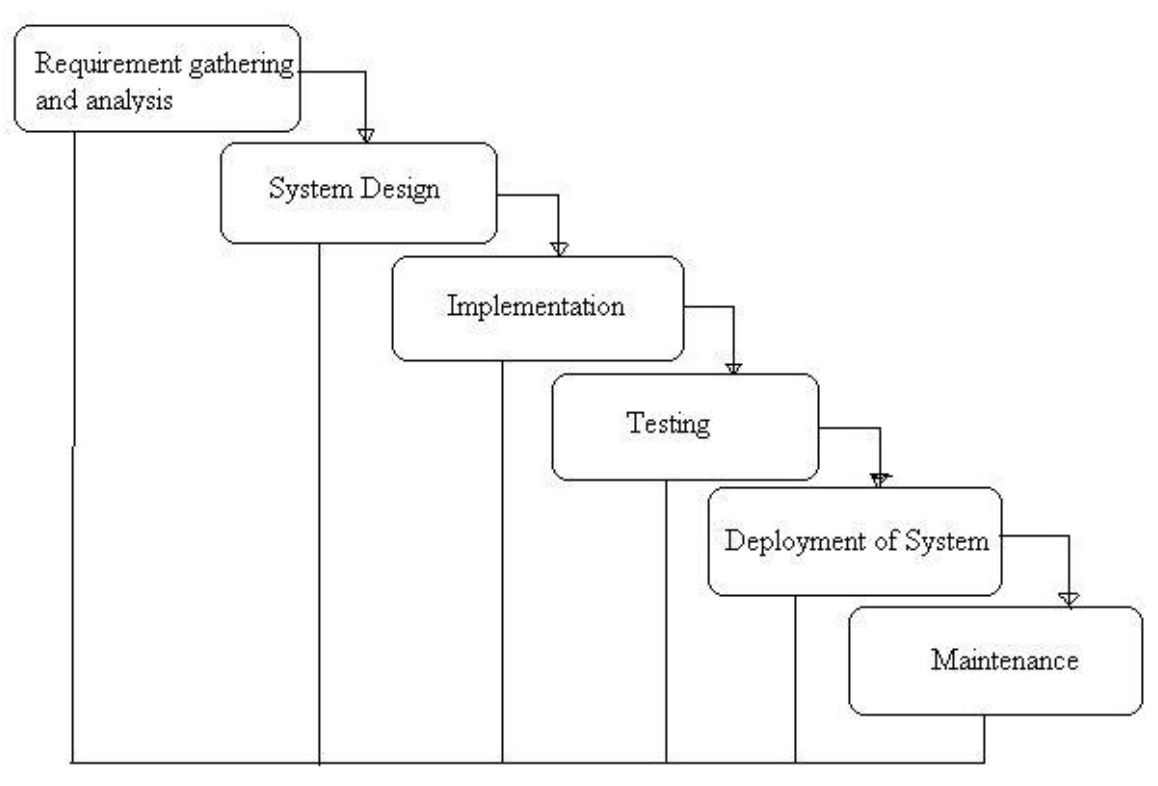
-Version control

# Software Development Models

The software development models are the different processes or steps that are being used for the development of the project depending on the project’s aims, final objective and goals. There are many development life cycle models that have been developed to fulfil different requirements. The models specify the various stages of the process and the order in which they are carried out.

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

## Waterfall model

The Waterfall Model was first Process Model to be presented. It is additionally referred to as a linear-sequential life cycle model. It is extremely easy to comprehend and utilize. In a waterfall model, each stage must be finished completely before the following stage can start. This kind of software development model is utilized for the venture which is small and there are no uncertain requirements. When finishing each stage, a review happens to decide whether the project is on the right track and whether to proceed or dispose of the project. In this model software testing begins only after the development is finished. In waterfall model, phases do not overlap with one another

*Figure 2: Waterfall Model Flow of Phases*

Advantages:

* Simple and easy to understand and use.
* Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
* Phases are processed and completed one at a time. Phases do not overlap.
* Works well for smaller projects where requirements are very well understood.

Disadvantages:

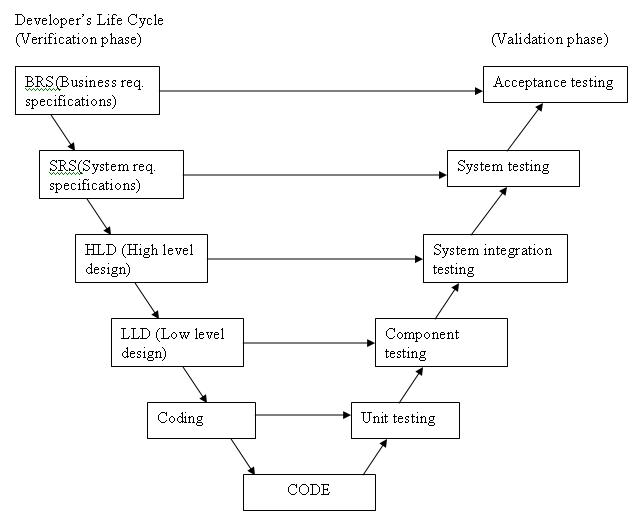
* Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.

When to use the Waterfall model:

* This model is used only when the requirements are very well known, clear and fixed.
* Product definition is stable.
* Technology is understood.
* There are no ambiguous requirements
* Ample resources with required expertise are available freely
* The project is short

## V-Model

The V-Model refers to the Verification and Validation model. Similar to the Waterfall Model, the V-Model is a sequential flow of processes, where each phase has to be completed before the team is able to move on to the following phase. However, the development phase in the V-Model is segregated into smaller and specific areas, whilst ensuring that the testing phase are properly correspondent to the related segregated development phase.



*Figure 3: V-Model Flow of Phases*

V- model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. V-Model is one of the many software development models. Testing of the product is planned in parallel with a corresponding phase of development in V-model. The Model is split into two phases, the high-level design phase (HLD) and the low-level design phase (LLD).  
The high-level design (HLD) phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.   
The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well  
  
Advantages:

* Simple and easy to use.
* Testing activities like planning, testdesigning happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
* Proactive defect tracking – that is defects are found at early stage.
* Avoids the downward flow of the defects.
* Works well for small projects where requirements are easily understood.

Disadvantages:

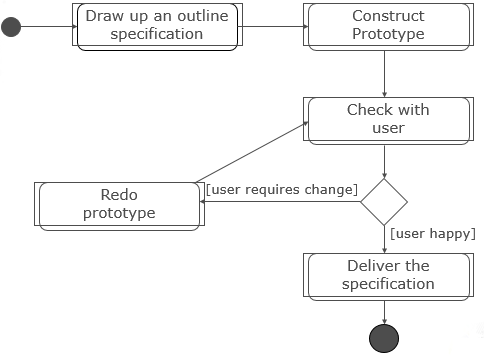
* Very rigid and least flexible.
* Software is developed during the implementation phase, so no early prototypes of the software are produced.
* If any changes happen in midway, then the test documents along with requirement documents has to be updated.

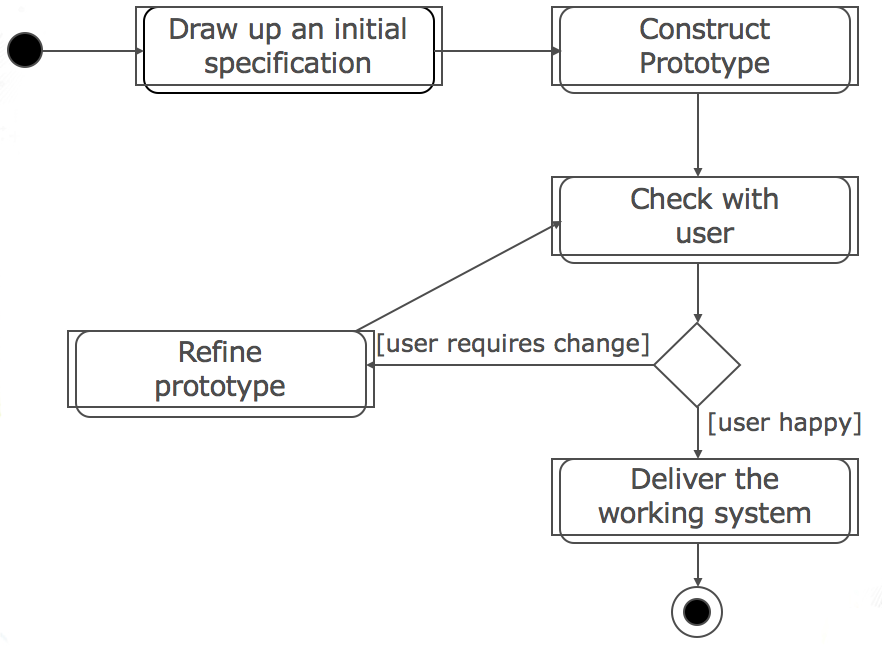
When to use the V-Model:

* The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
* The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise

## Prototype Model

A prototype is a working product with limited functionalities. The implementation of the Prototype Model in the Software Development Life Cycle results in the building of software product prototypes which displays the functions and design of the product that under development during the building process of the product prototype. This model is commonly applied to complicated and large projects which do not have a general existing system to act as a basis for determining the requirements of the product. The prototype, however, does not include all of the functionalities and logic that are required in the finalised product. Prototyping allows the team and the client to have a tangible prototype to test out and use. This allows better perspective and view of the product and its capabilities, as well as its functionalities, so as to allow for additional suggestions and solutions to improve on the prototype. The client will also be able to fully understand the requirements of the desired product that they have provided to the team. Moreover, the team will be able to clearly know if the development of the product meets their requirements.



*Figure 4: Throwaway Prototyping Model Flow of Phases  
  
  
*

*Figure 5: Evolutionary Prototyping Model Flow of Phases*

Types of Prototype Model:

Throwaway Prototyping (Figure 4)

* + Use a high-level language e.g. Visual Basic
  + Example: drag and drop to create GUI
  + Reuse components
  + Take from existing systems
  + Ignore error handling
  + Examples: no validation of input, no exception handling
  + Omit features e.g. security, logging
  + Ignore functions e.g. doing a mock-up

Evolutionary Prototyping (Figure 5)

* + Develop an initial system implementing parts of the system that are well understood
  + Show the developed system to user for comments (explore requirements)
  + Refine the system and repeat the process to implement more parts of the system until the full system is completed

Advantages:

* Clarify user requirements
* Specifications can be developed incrementally, giving users opportunity to change their mind

Disadvantages:

* Estimating, planning and managing a prototype project can be difficult because there is no regular deliverables e.g. how to predict how many iterations?
* Continual changes tend to corrupt software structure. Changes become more costly and difficult

When to use Throwaway or Evolutionary Prototype Model:

Throwaway

* Verify requirements that are unclear
* Useful when you have uncertain requirements

Evolutionary

* Uncover unknown requirements and then evolve
* Useful when requirements are hard to specify

**References**

**Kenji Leong**

[**http://istqbexamcertification.com/what-are-the-software-development-models/**](http://istqbexamcertification.com/what-are-the-software-development-models/)

[**http://istqbexamcertification.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/**](http://istqbexamcertification.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/)

[**http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/**](http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/)

[**https://www.scrumalliance.org/why-scrum#why-its-called-scrum**](https://www.scrumalliance.org/why-scrum#why-its-called-scrum)

[https://www.scrumalliance.org/why-scrum - why-its-called-scrum](https://www.scrumalliance.org/why-scrum#why-its-called-scrum)

**Jia Cheng**

<https://www.tutorialspoint.com/sdlc/sdlc_overview.htm>

^and all the subsequently pages

**Goh Mao Cheng**

[**http://istqbexamcertification.com/what-are-the-software-development-models/**](http://istqbexamcertification.com/what-are-the-software-development-models/)