***WHAT IS SOFTWARE DEVELOPMENT?***

Software development (also known as application development, software design, designing software, software application development, enterprise application development, or platform development) is the development of a [software](http://en.wikipedia.org/wiki/Software) product. The term "software development" may be used to refer to the activity of [computer programming](http://en.wikipedia.org/wiki/Computer_programming), which is the process of writing and maintaining the source code, but in a broader sense of the term it includes all that is involved between the conception of the desired software through to the final manifestation of the software, ideally in a planned and structured process. Therefore, [software development](http://www.binarysemantics.com/software_development_company.html) may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

Software can be developed for a variety of purposes, the three most common being to meet specific needs of a specific client/business (the case with [custom software](http://en.wikipedia.org/wiki/Custom_software)), to meet a perceived need of some set of potential users (the case with [commercial](http://en.wikipedia.org/wiki/Commercial_software) and [open source software](http://en.wikipedia.org/wiki/Open_source_software)), or for personal use (e.g. a scientist may write software to automate a mundane task). Embedded software development, that is, the development of [embedded software](http://en.wikipedia.org/wiki/Embedded_software) such as used for controlling consumer products, requires the development process to be integrated with the development of the controlled physical product.

The need for better [quality control](http://en.wikipedia.org/wiki/Quality_control) of the software development process has given rise to the discipline of [software engineering](http://en.wikipedia.org/wiki/Software_engineering), which aims to apply the systematic approach exemplified in the [engineering](http://en.wikipedia.org/wiki/Engineering) paradigm to the process of software development

**Software Development Process**

A software development process, also known as a *software development life cycle* (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a lifecycle model a more general term and a software development process a more specific term. For example, there are many specific software development processes that 'fit' the spiral lifecycle model. ISO 12207 is an ISO standard for software lifecycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO 12207.

**Software development activities**

The activities of the software development process are represented in the waterfall model. There are several other models to represent this process.

***Planning***

The important task in creating a software product is extracting the requirements or requirements analysis. Customers typically have an abstract idea of what they want as an end result, but not what software should do. Incomplete, ambiguous, or even contradictory requirements are recognized by skilled and experienced software engineers at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

***Implementation, testing and documenting***

Implementation is the part of the process where software engineers actually program the code for the project.

Software testing is an integral and important phase of the software development process. This part of the process ensures that defects are recognized as soon as possible.

Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an API, be it external or internal. It is very important to document everything in the project.

***Deployment and maintenance***

Deployment starts after the code is appropriately tested, is approved for release and sold or otherwise distributed into a production environment.

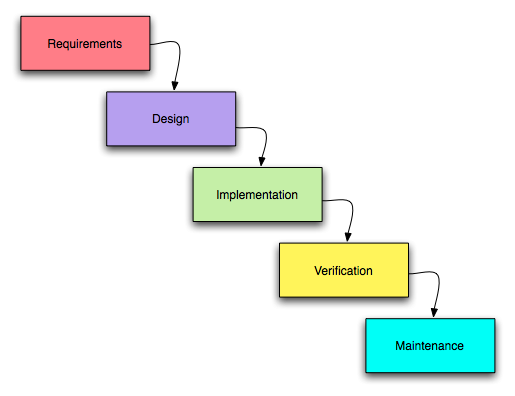
Software Training and Support is important and a lot of developers fail to realize that. It would not matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are often resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, it is very important to have training classes for new clients of your software.

Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. It may be necessary to add code that does not fit the original design to correct an unforeseen problem or it may be that a customer is requesting more functionality and code can be added to accommodate their requests. If the labor cost of the maintenance phase exceeds 25% of the prior-phases' labor cost, then it is likely that the overall quality of at least one prior phase is poor.

***Software development models***

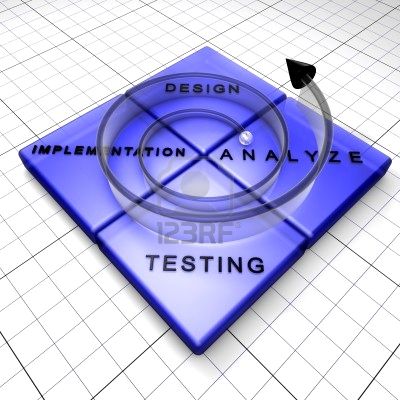
Several models exist to streamline the development process. Each one has its pros and cons, and it's up to the development team to adopt the most appropriate one for the project. Sometimes a combination of the models may be more suitable.

***Waterfall model***

The waterfall model shows a process, where developers are to follow these phases in order:

In a strict Waterfall model, after each phase is finished, it proceeds to the next one. Reviews may occur before moving to the next phase which allows for the possibility of changes (which may involve a formal change control process). Reviews may also be employed to ensure that the phase is indeed complete; the phase completion criteria are often referred to as a "gate" that the project must pass through to move to the next phase. Waterfall discourages revisiting and revising any prior phase once it's complete. This "inflexibility" in a pure Waterfall model has been a source of criticism by supporters of other more "flexible" models.

***Spiral mode***

******The key characteristic of a Spiral model is risk management at regular stages in the development cycle. In 1988, Barry Boehm published a formal software system development "spiral model", which combines some key aspect of the waterfall model and rapid prototyping methodologies, but provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems.

The Spiral is visualized as a process passing through some number of iterations, with the four quadrant diagram representative of the following activities:

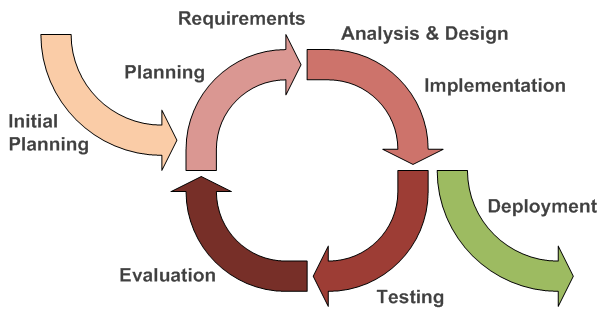
* identify software targets, selected to implement the program, clarify the project development restrictions;
* Risk analysis: an analytical assessment of selected programs, to consider how to identify and eliminate risk;the implementation of the project: the implementation of software development and verification;Risk-driven spiral model, emphasizing the conditions of options and constraints in order to support software reuse, software quality can help as a special goal of integration into the product development. However, the spiral model has some restrictive conditions, as follows:

The spiral model emphasizes risk analysis, and thus requires customers to accept this analysis and act on it. This requires both trust in the developer as well as the willingness to spend more to fix the issues, which is the reason why this model is often used for large-scale internal software development.

🡪If the implementation of risk analysis will greatly affect the profits of the project, the spiral model should not be used.

Software developers have to actively look for possible risks, and analyze it accurately for the spiral model to work. The first stage is to formulate a plan to achieve the objectives with these constraints, and then strive to find and remove all potential risks through careful analysis and, if necessary, by constructing a prototype.

***Iterative and incremental development***

******Iterative development prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes can assist with revealing design goals of a client who does not know how to define what they want.

***Agile development***

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

There are many variations of agile processes:

In Extreme Programming (XP), the phases are carried out in extremely small (or "continuous") steps compared to the older, "batch" processes. The (intentionally incomplete) first pass through the steps might take a day or a week, rather than the months or years of each complete step in the Waterfall model. First, one writes automated tests, to provide concrete goals for development. Next is coding (by a pair of programmers), which is complete when all the tests pass, and the programmers can't think of any more tests that are needed. Design and architecture emerge out of refactoring, and come after coding. Design is done by the same people who do the coding. (Only the last feature — merging design and code — is common to all the other agile processes.) The incomplete but functional system is deployed or demonstrated for (some subset of) the users (at least one of which is on the development team). At this point, the practitioners start again on writing tests for the next most important part of the system.

***Code and fix***

"Code and fix" development is not so much a deliberate strategy as an artifact of naiveté and schedule pressure on software developers.[2] Without much of a design in the way, programmers immediately begin producing code. At some point, testing begins (often late in the development cycle), and the inevitable bugs must then be fixed before the product can be shipped.