The Iterative Project Model (IPM)

A lightweight methodology for project management

The Iterative Project Model

This paper is an introduction to simple idea I call the Iterative Project Model.

There are important parallels between software development and project management. In fact I think of software development as a specialised kind of project management. This will be heresy to many of the purveyors of proprietary software development methodologies but I feel it is true none-the-less.

The model is born out of software development models and my involvement with software development and IT projects. I started my career in software development and spent many hours studying, writing about and evolving software development methodologies. Later in my career I took responsibility for larger projects and went through a similar process learning about project management methodology and delivering more broad based IT projects. After a while I realised the approach was basically the same and it had similarities with other disciplines such as engineering and architecture.

Out of these common ideas was born the Iterative Project Model, a simple methodology that *you can use to get things done!*

A project, be it a software development project or any other kind, is a basic process of collecting requirements, designing a solution, developing the solution, testing it and then implementing it. The same process could be applied to building a bridge, writing a university curriculum, manufacturing a new product or even writing a bit of code. Many existing project management methodologies add layer upon layer of complexity to their basic models and frustrate novice project manager's attempts to actually achieve something! Experienced project managers strip back layers of detail and simply use whatever techniques they prefer to get the job done but new project managers often founder in the quagmire of impenetrable, convoluted methodologies.

The Importance of a Model

A model is a communications tool.

In the context of this paper a model is used to describe the intangible process of project management so that everyone involved in that process can have the same conceptual understanding of it.

It allows individuals to collaborate on a project and gives them a common basis on which they can discuss various aspects of the project. It also helps those individuals understand their role in the pursuit of the common objectives. Project management without models would be a bit like driving without the highway code. Everyone would still be able to drive their own cars but arriving at your destination would be a bit of an adventure!

Traditional project management models have often had the drawback of being overly complicated or convoluted to understand. While project management is a necessarily complex discipline there is no reason why the concepts behind it shouldn't be easily

explained. To this end the model I present here is a simple one which allows people in day-to-day roles to apply the principals of project management in a practical way.

There are three basic principles we look for in a project management model : a model should be simple, scalable and it should have utility.

Simple

A model is by definition simpler than its original. A model which is more complicated than the process it represents is worse than useless. How simple should it be ? As simple as is practically useful.

There is a dichotomy between the power of a model and its simplicity. A model which is complex and difficult to understand can be extremely powerful in the hands of an expert but nearly useless in the hands of a novice. Conversely a model which is extremely simple is very useful to the novice (granting them conceptual insight) but can be useless for an expert. The easiest solution is to have a model which is inherently simple but can cope with the complexity of larger systems.

Scalable

In a similar fashion a model needs to be adaptable to not only the requirements of the individuals using it but also to the project to which it is applied. A model should suit both large scale projects and small scale projects since the tasks represented in each are generic and change only in content, not in character, from project to project. In general the same tasks must be achieved in a project no matter if it involves one person or one hundred.

Usable

A purely theoretical model which has only limited application in the ‘real world’ offers little or no benefits for those who actually develop software. As such it does not satisfy the condition of being a tool which can be used in the achievement of the goal of software development.

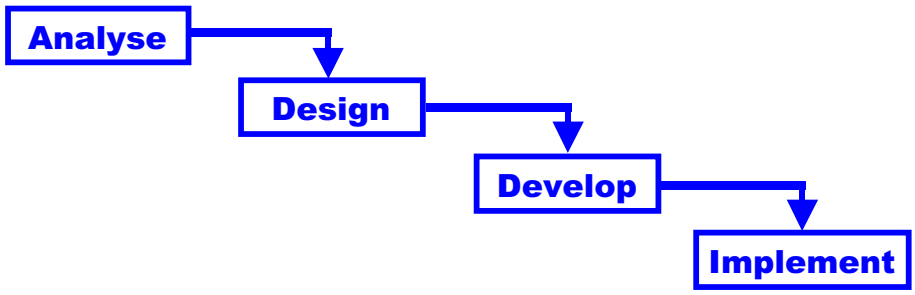
In the day to day world of software development a useful model can be used to explain, plan and execute the various phases of development. The model should also seek to highlight deficiencies or errors in the process in order to allow rectification of these deficiencies.

Simplicity is again a requirement for utility. The more complicated and bureaucratic the model the more difficult it is to implement, the more likely it is to be ignored and the more probable it is to be rejected by the very people it is supposed to support. The model must be applicable to day-to-day tasks to have any utility whatsoever.

The Linear Project Model

A project can be thought of as a linear sequence of events. This is neither particularly accurate nor particularly useful in most situations but it does allow you to visualise the series of events in a project in the simplest way. It also emphasises the importance of delivery with steps being taken towards a conclusion.

Below is the “Waterfall Model” which shows the typical project tasks flowing into each other. At the start of a project the analysis and design of the project are undertaken and flow towards a conclusion in the execution and implementation phases.

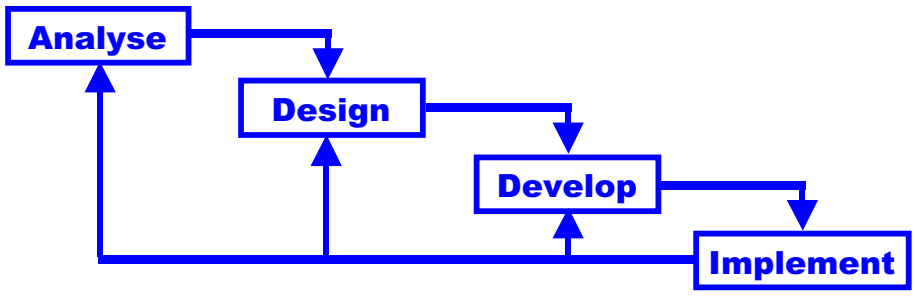


The Waterfall Model was widely adopted in the early days of software development and a lot of blame has been laid at its door. The model was derived from engineering models and while it does prove a useful point it is intrinsically flawed.

It is very rare that requirements analysis can be entirely completed before design and design before execution and so on. In a small project this is not a significant problem since the span from “analyse” to “implement” may be a period of days or even weeks. For a larger scale project spanning months or even years the gap becomes significant. The more time that passes between analysis and implementation the more of a gap between the delivered project and the requirements of end-user’s.

Gathered requirements may be accurate at the point of capture but decay with frightening speed. In the modern world, the chances of your analysis of requirements being valid years after it has been conducted is very slim indeed.

The shortcomings of this model became apparent and other versions of the waterfall model have been developed. One, the Iterative Waterfall Model, includes a loop as shown below :

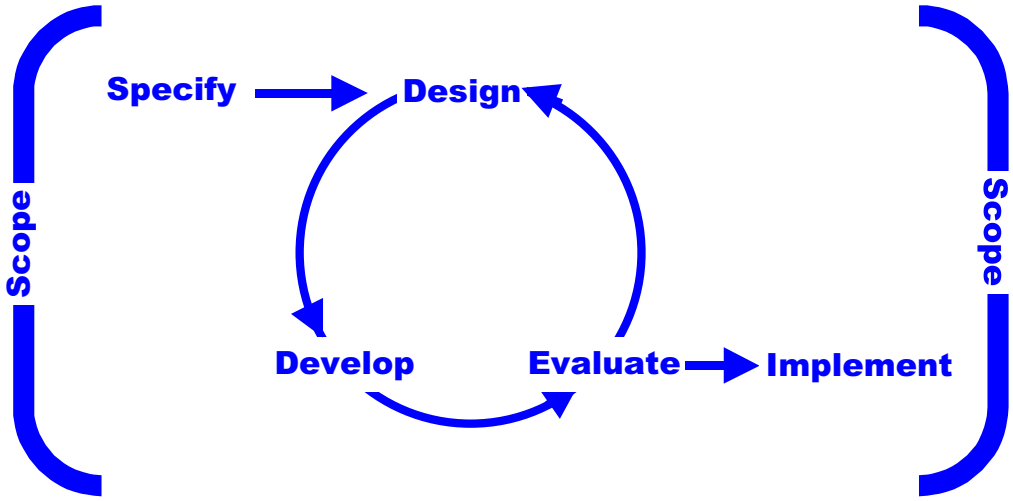


This model attempts to overcome the limitations of the original Waterfall Model by adding an “iterative” loop to the end of the cycle. That is, in order to keep up with changing requirements the “analysis” phase is revisited at the end of the cycle and the process starts over again.

This alleviates the situation somewhat but still introduces a considerable lag between analysis and implementation. If requirements change throughout the life of the project the waterfall model requires the completion of a full cycle before they can be revisited. In truth requirements don't often change wildly but the understanding of them often improves as the project progresses and the waterfall model makes no allowance for this.

The Iterative Project Model

With the advent of techniques in software development such as "rapid prototyping", new models began to appear on the scene. A number of these models have been popular in recent years including models such as Barry Boehm's Spiral Model, DSDM and RAD. The iterative model I present here has its origins in all of these models but has been adapted for generic project management and for the sake of utility.



The model above shows the five phases of a project lifecycle bounded by the 'scope' of the project.

The five phases are :

- *Specify* the requirements of users and stakeholders in order to set project objectives
- *Design* your system or product to meet those requirements
- *Develop* your system or product to deliver upon those requirements
- *Evaluate* the delivered system to measure your success in delivering requirements
- *Implement* your system by delivering to the end-users

The emphasis in the Iterative model is on fast iterations through the cycle of designing and developing your project. Used in conjunction with rapid prototyping the model

works on the principle of integrating users into the design process and using prototypes to illustrate design decisions. The model is particularly suited to projects in rapidly changing environments where the team can quickly adapt to different situations.

But be warned ! The Iterative model comes with its own intrinsic caveats however. The fact that it is a less rigid model than something like the waterfall model and that it is used in environments with rapid change often leads project managers to believe they have carte blanche to ignore all of the other tenets of project management. Nothing could be father from the truth!

The Iterative model requires even more finesse and control than a more structured model. Although it is powerful in the hands of the right manager, in the hands of a lacklustre project manager it can be a disaster. One of the particular sins that is committed by many project managers time and time again is the headlong dive into the "development" phase of a project.

Without proper planning and control any project is doomed to failure. Without strong project management control the Iterative model quickly spirals out of control and you have a runaway project on your hands!

Summary

The Iterative model is not a new proprietary methodology or some mystical insight gifted to me one night by the ghost of Charles Babbage. It is merely an observation on how things work. It is drawn from my real world involvement in projects and software development and a genuine frustration with existing methodologies.

The iterative cycle of "design-develop-evaluate" closely parallels the standard scientific method of "observe-theorise-experiment" and is a natural process for most contemporary software professionals.

The nicest thing about the model is that it is both simple and usable. It doesn't require special training to understand, it doesn't have it's own specific terminology or cabalistic diagrams and yet it works. It will help you get your job done, deliver software or systems on time and under budget that will make users happy.

I hope you find it useful.

http://www.members.tripod.com/nickjenkins/program/