Software Testing
and
Software Development
Lifecycles

Executive Summary

This paper outlines a number of commonly used software development lifecycle models, with particular emphasis on the testing activities involved in each model.

Irrespective of the lifecycle model used for software development, software has to be tested. Efficiency and quality are best served by testing software as early in the lifecycle as practical, with full regression testing whenever changes are made.

IPL is an independent software house founded in 1979 and based in Bath. IPL was accredited to ISO9001 in 1988, and gained TickIT accreditation in 1991. IPL has developed and supplies the AdaTEST and Cantata software verification products. AdaTEST and Cantata have been produced to these standards.

IPL
Eveleigh House
Grove Street
Bath
BA1 5LR
UK
Phone: +44 (0) 1225 444888
Fax: +44 (0) 1225 444400
email ipl@iplbath.com

Certificate Number FM1589

Last Update:01/08/96 09:37
File: LIFE_C.DOC

1. Introduction

The various activities which are undertaken when developing software are commonly modelled as a **software development lifecycle**. The software development lifecycle begins with the identification of a requirement for software and ends with the formal verification of the developed software against that requirement.

The software development lifecycle does not exist by itself, it is in fact part of an overall **product lifecycle**. Within the product lifecycle, software will undergo maintenance to correct errors and to comply with changes to requirements. The simplest overall form is where the product is just software, but it can become much more complicated, with multiple software developments each forming part of an overall system to comprise a product.

There are a number of different models for software development lifecycles. One thing which all models have in common, is that at some point in the lifecycle, software has to be tested. This paper outlines some of the more commonly used software development lifecycles, with particular emphasis on the testing activities in each model.

2. Sequential Lifecycle Models

The software development lifecycle begins with the identification of a requirement for software and ends with the formal verification of the developed software against that requirement. Traditionally, the models used for the software development lifecycle have been sequential, with the development progressing through a number of well defined phases. The sequential phases are usually represented by a V or waterfall diagram. These models are respectively called a **V lifecycle model** and a **waterfall lifecycle model**.

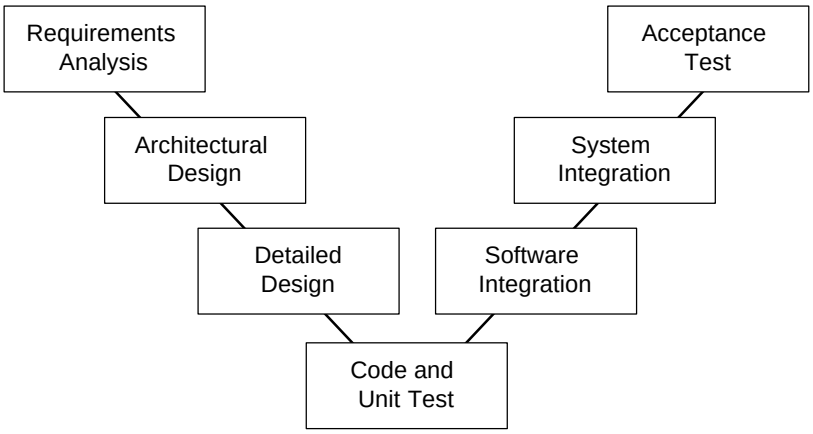

Figure 1 V Lifecycle Model

There are in fact many variations of V and waterfall lifecycle models, introducing different phases to the lifecycle and creating different boundaries between phases. The following set of lifecycle phases fits in with the practices of most professional software developers.

- The **Requirements** phase, in which the requirements for the software are gathered and analyzed, to produce a complete and unambiguous specification of what the software is required to do.

- The **Architectural Design** phase, where a software architecture for the implementation of the requirements is designed and specified, identifying the components within the software and the relationships between the components.

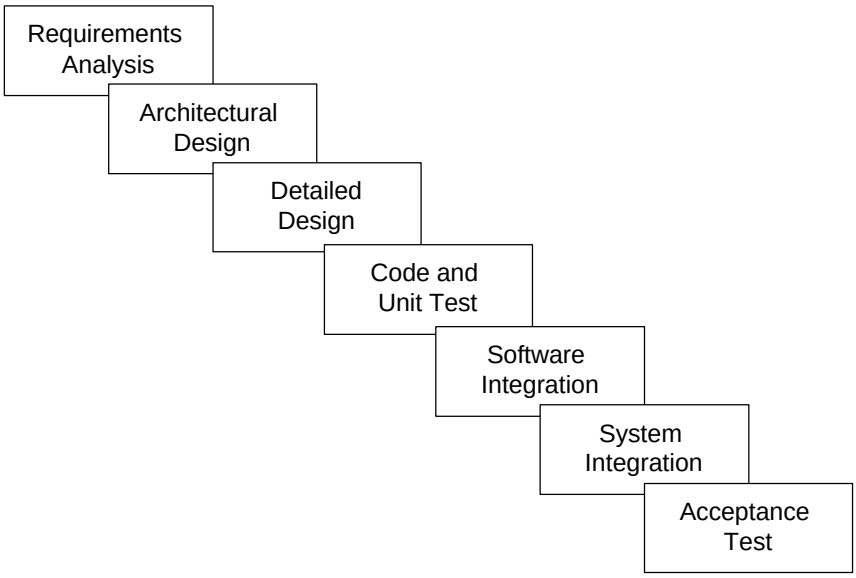

Figure 2 Waterfall Lifecycle Model

- The **Detailed Design** phase, where the detailed implementation of each component is specified.
- The **Code and Unit Test** phase, in which each component of the software is coded and tested to verify that it faithfully implements the detailed design.
- The **Software Integration** phase, in which progressively larger groups of tested software components are integrated and tested until the software works as a whole.
- The **System Integration** phase, in which the software is integrated to the overall product and tested.
- The **Acceptance Testing** phase, where tests are applied and witnessed to validate that the software faithfully implements the specified requirements.

Software specifications will be products of the first three phases of this lifecycle model. The remaining four phases all involve testing the software at various levels, requiring test specifications against which the testing will be conducted as an input to each of these phases.

3. Progressive Development Lifecycle Models

The sequential V and waterfall lifecycle models represent an idealised model of software development. Other lifecycle models may be used for a number of reasons, such as volatility of requirements, or a need for an interim system with reduced functionality when long timescales are involved. As an example of other lifecycle models, let us look at progressive development and iterative lifecycle models.

A common problem with software development is that software is needed quickly, but it will take a long time to fully develop. The solution is to form a compromise between

timescales and functionality, providing "interim" deliveries of software, with reduced functionality, but serving as a stepping stones towards the fully functional software. It is also possible to use such a stepping stone approach as a means of reducing risk.

The usual names given to this approach to software development are **progressive development** or **phased implementation**. The corresponding lifecycle model is referred to as a **progressive development lifecycle.** Within a progressive development lifecycle, each individual phase of development will follow its own software development lifecycle, typically using a V or waterfall model. The actual number of phases will depend upon the development.

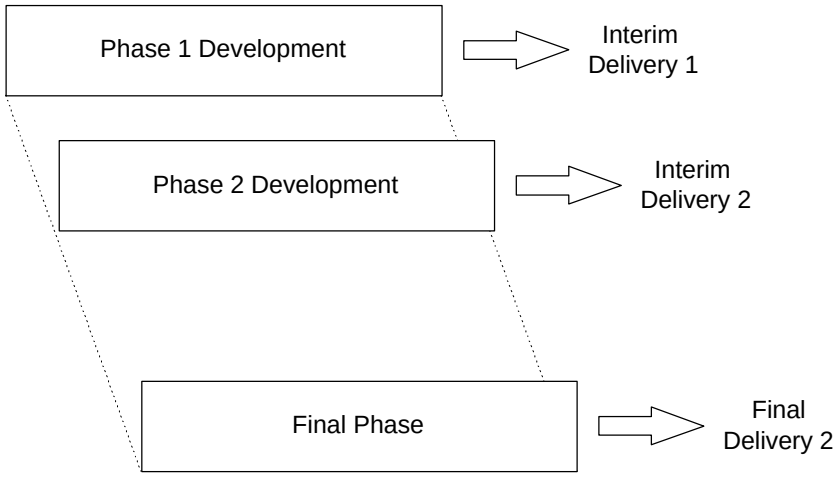

Figure 3 Progressive Development Lifecycle

Each delivery of software will have to pass acceptance testing to verify the software fulfils the relevant parts of the overall requirements. The testing and integration of each phase will require time and effort, so there is a point at which an increase in the number of development phases will actually become counter productive, giving an increased cost and timescale, which will have to be weighed carefully against the need for an early solution.

The software produced by an early phase of the model may never actually be used, it may just serve as a **prototype**. A prototype will take short cuts in order to provide a quick means of validating key requirements and verifying critical areas of design. These short cuts may be in areas such as reduced documentation and testing. When such short cuts are taken, it is essential to plan to discard the prototype and implement the next phase from scratch, because the reduced quality of the prototype will not provide a good foundation for continued development.

4. Iterative Lifecycle Models

An **iterative lifecycle** model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

Consider an iterative lifecycle model which consists of repeating the four phases in sequence, as illustrated by figure 4.

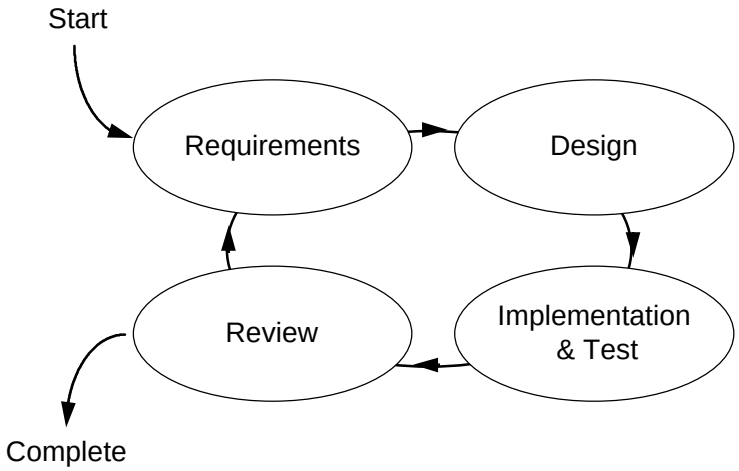

Figure 4 Iterative Lifecycle Model

- A **Requirements** phase, in which the requirements for the software are gathered and analyzed. Iteration should eventually result in a requirements phase which produces a complete and final specification of requirements.
- **Design** phase, in which a software solution to meet the requirements is designed. This may be a new design, or an extension of an earlier design.
- An **Implementation and Test** phase, when the software is coded, integrated and tested.
- A **Review** phase, in which the software is evaluated, the current requirements are reviewed, and changes and additions to requirements proposed.

For each cycle of the model, a decision has to be made as to whether the software produced by the cycle will be discarded, or kept as a starting point for the next cycle (sometimes referred to as **incremental prototyping**). Eventually a point will be reached where the requirements are complete and the software can be delivered, or it becomes impossible to enhance the software as required, and a freash start has to be made.

The iterative lifecycle model can be likened to producing software by successive approximation. Drawing an analogy with mathematical methods which use successive approximation to arrive at a final solution, the benefit of such methods depends on how rapidly they converge on a solution.

Continuing the analogy, successive approximation may never find a solution. The iterations may oscillate around a feasible solution or even diverge. The number of iterations required may become so large as to be unrealistic. We have all seen software developments which have made this mistake!

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification (including testing) of each version of the software against those requirements within each cycle of the model. The first three phases of the example iterative model are in fact an abbreviated form a sequential V or waterfall lifecycle model. Each cycle of the model produces software which requires testing at the unit level, for software integration, for system integration and for acceptance. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

5. Maintenance

Successfully developed software will eventually become part of a product and enter a **maintenance phase**, during which the software will undergo modification to correct errors and to comply with changes to requirements. Like the initial development, modifications will follow a software development lifecycle, but not necessarily using the same lifecycle model as the initial development.

Throughout the maintenance phase, software tests have to be repeated, modified and extended. The effort to revise and repeat tests consequently forms a major part of the overall costs of developing and maintaining software.

The term **regression testing** is used to refer to the repetition of earlier successful tests in order to make sure that changes to the software have not introduced side effects.

6. Summary and Conclusion

Irrespective of the lifecycle model used for software development, software has to be tested. Efficiency and quality are best served by testing software as early in the lifecycle as practical, with full regression testing whenever changes are made.

Such practices become even more critical with progressive development and iterative lifecycle models, as the degree of retesting needed to control the quality of software within such developments is much higher than with a more traditional sequential lifecycle model.

Regression testing is a major part of software maintenance. It is easy for changes to be made without anticipating the full consequences, which without full regression testing could lead to a decrease in the quality of the software. The ease with which tests can be repeated has a major influence on the cost of maintaining software.

A common mistake in the management of software development is to start by badly managing a development within a V or waterfall lifecycle model, which then degenerates into an uncontrolled iterative model. This is another situation which we have all seen causing a software development to go wrong.

AdaTEST and Cantata are tools which facilitate automated, repeatable and maintainable testing of software, offering significant advantages to developers of Ada, C and C++ software. The benefits of repeatable and maintainable testing, gained from using AdaTEST or Cantata, become even more important when a progressive development or iterative model is used for the software development lifecycle.

There are a wide range of software development lifecycle models which have not been discussed in this paper. However, other lifecycle models generally follow the form and share similar properties to one of the models described herein, offering similar benefits from the use of AdaTEST or Cantata.