AN EMPIRICAL STUDY OF SOFTWARE ENGINEERING MODEL FOR SOFTWARE DEVELOPMENT

[1]V.Rajiv Jetson Associate Professor, Dept of CSE Kallam Haranadha reddy Institute of Tech.
rajivjetson@gmail.com

[2]Dr.G.Satyanarayana Prasad Professor & Dean, Dept of CSE , R.V.R& J.C college of Engineering
satyam.gp@gmail.com

Abstract

Software worth billions and trillions of dollars have gone waste in the past due to lack of proper techniques used for developing software resulting into software crisis. Historically, the processes of software development have played an important role in the software engineering. A number of life cycle models have been developed in last three decades. This paper is an attempt to analyze the software process model by various documented standard model.

Keywords
ISO, CMMI, TickIT.

I. Introduction

In the current scenario, information systems are important part of any organization. As compared to 1970's and 1980's, they are becoming more and more complex. In the early years i.e., 1990's, software development was not an independent established discipline. Instead it was only an extension of the hardware process. Earlier programs were written mostly in assembly language and were not complex. The persons or users who did programming were the one who also executed, tested and fixed the problem or errors in the software.

As the information systems became more and more complex and organizations became more dependent on software, a need was felt to develop the software in a systematic fashion. A survey was conducted by researchers in seventies and it was found that most of the softwares used by companies was of poor quality. Also companies were spending most of their time and money in maintaining the software. They found that software product was not same as hardware product as it was to be engineered or developed as it did not wear out. Programmers were clear about civil, mechanical, electrical and computer engineering but it was always a topic of debate that what engineering might mean for software. Most of the people consider the program and software to be same. But software not only consists of programs but also the supporting documents. Software lifecycle models are used as tools for planning and monitoring software projects. Numerous models have been proposed. Each model's effectiveness varies with project

circumstances. It is widely acknowledged that no single model is effective in all situations. Because of this, an effective model must be selected for every project.

II. Software Lifecycle Models

There are large number of software lifecycle models [1-3]. The software lifecycle models mentioned in the software engineering standards considered in this report have been identified as a starting point.
These are:

- ISO/IEC FDIS 1107:20074: This report is based on Waterfall, Spiral, Incremental Development and Evolutionary Development models. This standard states that ISO/IEC TR 24748 will provide additional information on software life cycle models [4].

- CMMI-DEV [6]: This is based on Waterfall, Spiral, Evolutionary, Incremental and Iterative models.

- The TickIT Guide[5]: This is based on Iterative Prototyping, Waterfall, V-model, Rapid Application Development (RAD) models, or a combination of these.

- ISO/IEC TR 19759 (SWEBOK)"[2]: This is based on Waterfall, Throwaway Prototyping, Evolutionary Development, Incremental/iterative Delivery, Spiral, Reusable Software Model, and Automated Software Synthesis models.

Additionally, Sommerville [3], in his popular textbook on software engineering, states that most software lifecycle models are based on one of three general models: Waterfall, Iterative Development and Component-based Software Engineering (CBSE).Each model's strengths and weaknesses are identified, as these are of key importance when selecting an appropriate software lifecycle model for a project.

This review shows that new software lifecycle models have generally been developed to address the shortcomings of the models current at the time, which is to be expected. In reality, software lifecycle models are often hybridized[3]. The hybridisation of two or more models can help address the weaknesses of the individual models. The type of hybridization that is appropriate will depend on project circumstances.

A. Classical Development Model

The Classical development software lifecycle model is a Delivery Strategy Model1. It is shown in fig. 1. In this model, all the stakeholder requirements the software must satisfy are implemented in a single

iteration and then delivered to the customer1. The software then enters the operation and maintenance phase1.

This model contrasts with incremental Delivery Strategy Models, in which the software is delivered to the customer in a number of iterations. In these models, each iteration delivers software that implements a subset of its stakeholder requirements, only the final iteration implements all the Requirements.

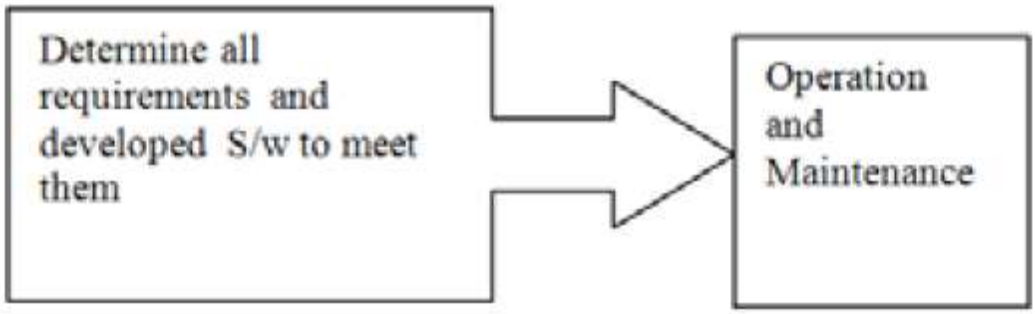


Fig. 1: Classical Development Model (Adapted from Alexander et al1)

It should be noted that software delivery strategy is often dictated by customer requirements. In the opinion of the author however these include

Strengths :

- Simple to understand.
- The customer receives software that satisfies all agreed stakeholder requirements.
- There is no risk of dependencies between different increments of software not being identified

Weaknesses :

- The customer must wait until all requirements are implemented before receiving working software.
- The customer cannot provide feedback on whether the software meets their needs until the entire development effort is complete.
- The development project is likely to be larger and more complex in comparison to incremental Delivery Strategy Models.

B. The Waterfall Model

The Waterfall model is one of the most used model of 1970's. It was proposed by W.W.Royce in 1970 as an alternative to Build and Fix software development method in which code was written and debugged. System was not formally designed and there was no way to check the quality criteria. Different phases of Waterfall model are shown in fig.2. Given below is a brief description of different phases of Waterfall Model.

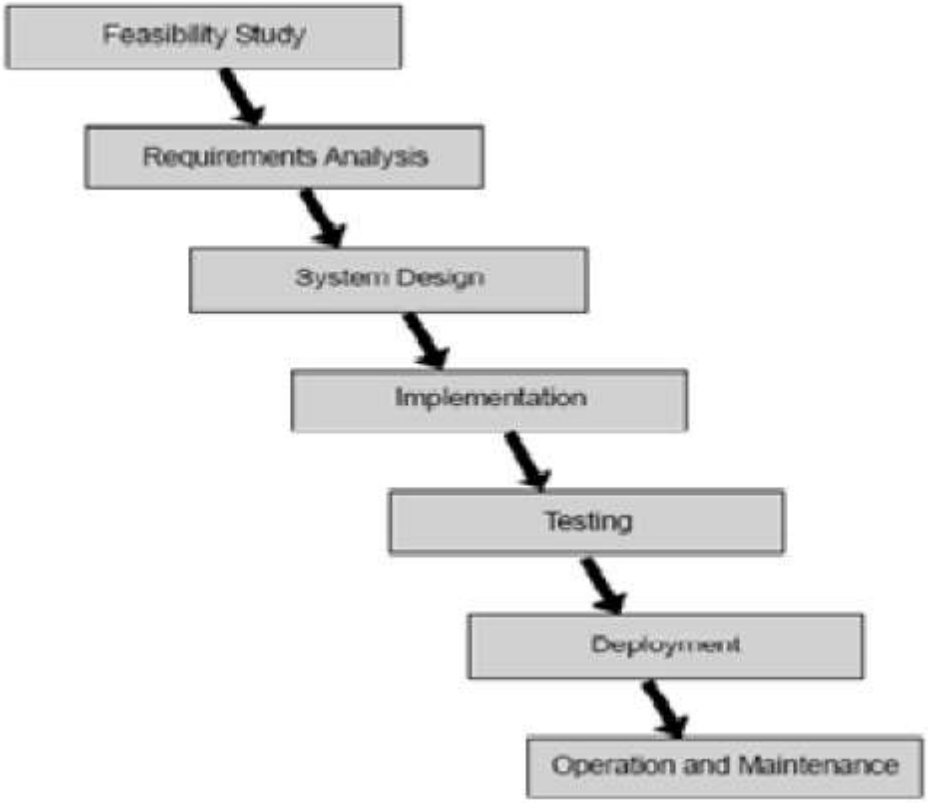


Fig. 2: Waterfall model

Feasibility study explores system requirements to determine project feasibility. All projects are feasible given unlimited resources and infinite time (Roger Pressman 1992).

Feasibility can be categorized into

- **Economic feasibility**
- **Technical feasibility**
- **Operational feasibility**
- **Schedule feasibility**
- **Legal and contractual feasibility**
- **Political feasibility**

Advantages and disadvantages of the Waterfall model are listed below:

Advantages

- Easy to understand even by non-technical persons i.e.customers.
- Each phase has well defined inputs and outputs e.g., input to system design stage is Requirement Specification
- Document(RSD) and output is the design document.
- Easy to use as software development proceeds.
- Each stage has well defined deliverables or milestones.
- Helps the project manager in proper planning of the project.

Disadvantages

- The biggest drawback of Waterfall model is that it does not support iteration.
- Software development on the other hand is iterative i.e., while designing activities are being carried out, new requirements can come up. Similarly while product is being coded, new design and requirement problems can come up.
- Another disadvantage of Waterfall model is that it is sequential in nature. One cannot start with a stage till preceding stage is completed e.g., one cannot start with the system design till all the requirements are understood and represented.
- Users have little interaction with the project team. Their feedback is not taken during development.
- Customer gets opportunity to review the product very late in life cycle because the working version of product is available very late in software development life cycle.
- Model is very rigid because output of each phase is prerequisite for successive stage.
- The Waterfall model also has difficulty in accommodating changes in the product after the development process starts. Amount of documentation produced is very high.

- The model in no way supports delivery of system in pieces.
- The model is not suitable for new projects because of uncertainty in the specifications.

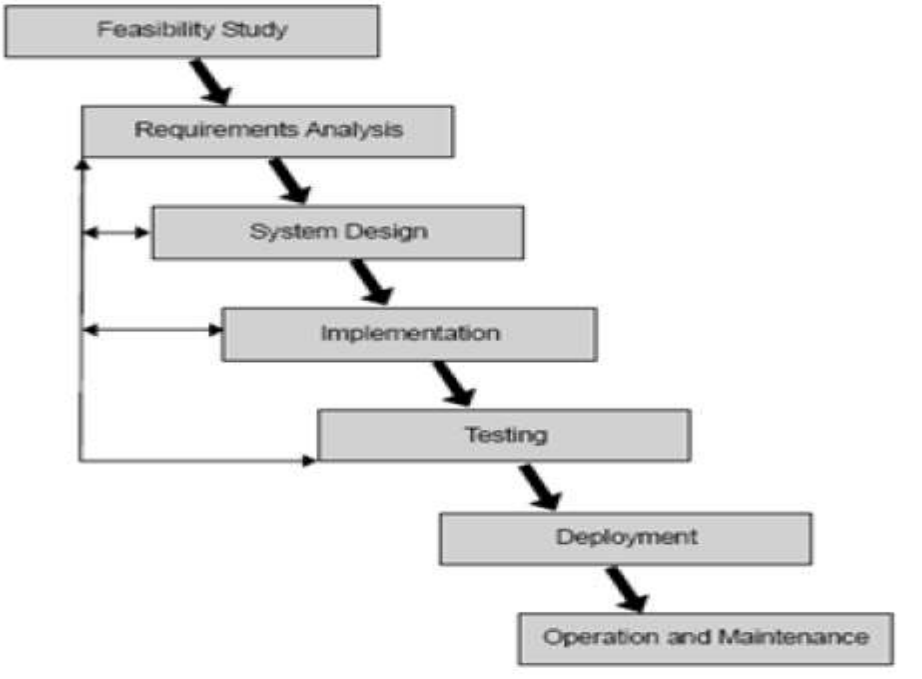


Fig. 3: Waterfall model with feedback

Though Waterfall model has been used for large projects in the past, its use must be limited to projects in which requirements are well understood or the company is working on a product of similar kind which it has developed in the past. Modified version of Waterfall model shown in Fig. 3 allows feedback to preceding stages and hence is not very rigid. This model clearly shows that the development team can go back to previous phases in order to have better clarity and understanding.

The Waterfall model is suited for well understood projects using familiar technology. It can also be used for existing projects if changes to be made are well defined.

C. The V-Model

This model was developed to relate the analysis and design activities with the testing activities and thus focuses on verification and validation activities of the product The advantages and disadvantages of the model are listed below.

Fig. 4: V Model

Advantages

- The model is simple and easy to use.
- The V model focuses on testing of all intermediate products, not only the final software.
- The model plans for verification and validation activities early in the life cycle thereby enhancing the probability of building an error free and good quality product

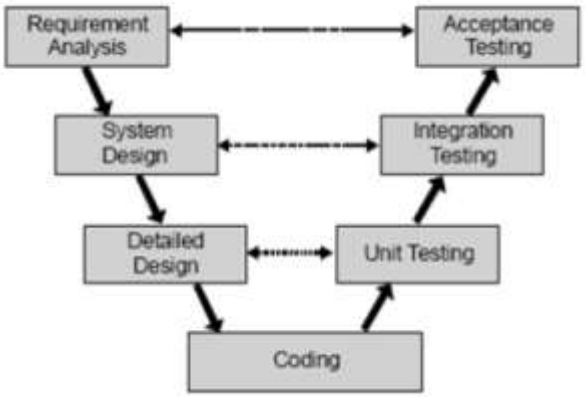
Disadvantages

- The model does not support iteration of phases and change in requirements throughout the life cycle.
- It does not take into account risk analysis. The V model is used for systems in which reliability is very important e.g., systems developed to monitor the state of the patients, software used in radiation therapy machines.

D. The Prototype Model

The concept of prototyping is not new in various streams of engineering. A prototype is a partially developed product. Robert T. Futrell and Shafer in their book Quality Software Project Management define prototyping as a process of developing working replica of a system(Robert02). This activity of prototyping now forms the basis of prototype software development life cycle model. Most of the users do not exactly know what they want until they actually see the product. Prototyping is used for developing a mock-up of product and is used for obtaining user feedback in order to refine it further as shown in fig. 4. Two approaches of prototyping can be followed:

1. Rapid Throwaway Prototyping



This approach is used for developing the systems or part of the systems where the development team does not have the understanding of the system. The quick and dirty prototypes are built, verified with the customers

and thrown away. This process continues till a satisfactory prototype is built. At this stage now the full scale development of the product begins.

2. Evolutionary Prototyping

This approach is used when there is some understanding of the requirements. The prototypes thus built are not thrown away but evolved with time. The block diagram of the prototype model is shown in fig. 5. The concept of prototyping has also led to the Rapid prototyping model and the Spiral model.

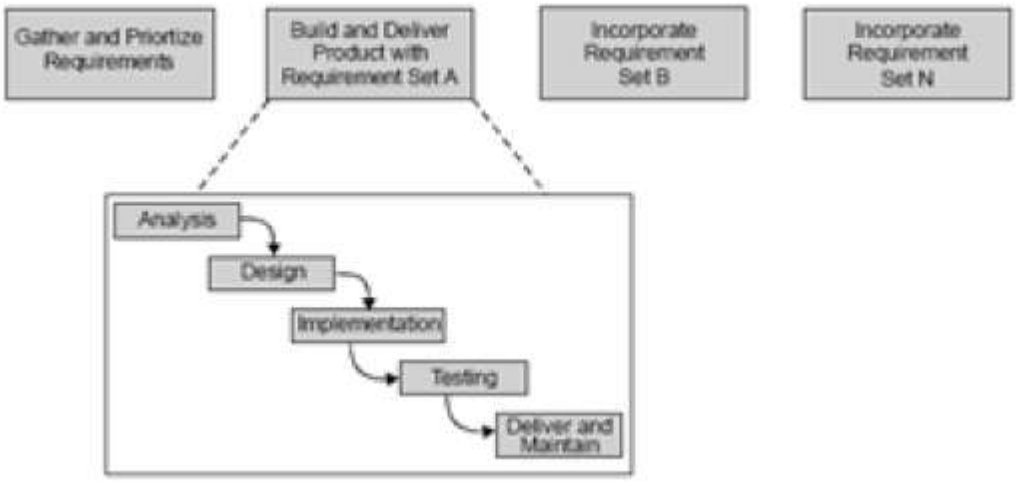
The **advantages and disadvantages** of the prototyping model are listed below:

Advantages

- A partial product is built in the initial stages. Therefore customers get a chance to see the product early in the life cycle and thus give necessary feedback.
- New requirements can be easily accommodated, as there is scope for refinement.
- Requirements become more clear resulting into an accurate product.
- As user is involved from the starting of the project, he tends to be more secure, comfortable and satisfied.
- Flexibility in design and development is also supported by the model.

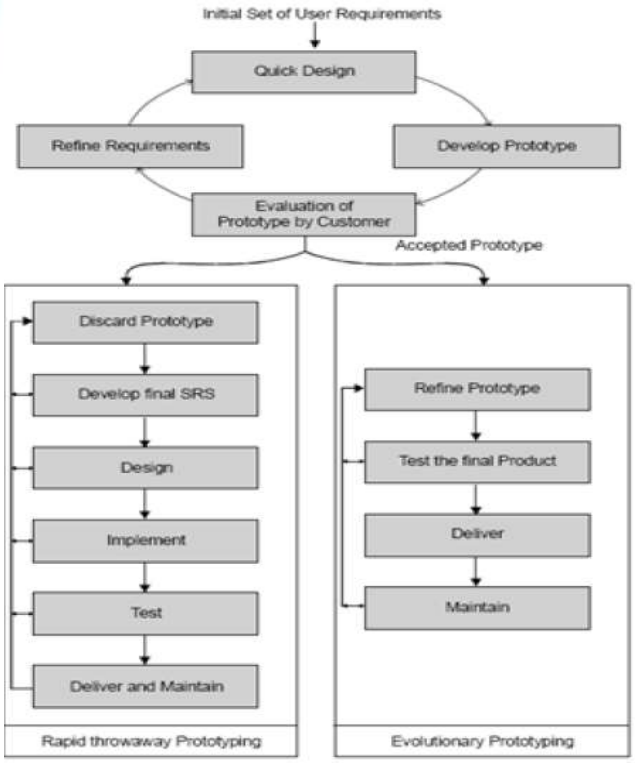
Disadvantages

- After seeing an early prototype end users demand the actual system to be delivered soon.
- End users may not like to know the difference between a prototype and a well engineered fully developed system.
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- If not managed properly, the iterative process of prototype demonstration and refinement can continue for long duration.
- If end user is not satisfied with initial prototype, he may loose interest in the project.
- Poor documentation.



Prototype model Fig 5

2.5 The Incremental Life Cycle Model complex systems is changing business requirements coming to have a model the changes in the discussed earlier do the evolutionary Evolutionary models nature. The development life popular evolutionary used by industry. working of the



Software Development Software like all other bound to evolve due to requirements or new up. Hence there is a need which can accommodate product. The models not take into consideration nature of the product. are also iterative in incremental software cycle model is one of the software process model The fig. 6 shows the incremental model.

Fig. 6: The incremental model

The advantages and disadvantages of incremental model are listed below:

Advantages

- As product is to be delivered in parts, total cost of project is distributed.
- Limited number of persons can be put on project because work is to be delivered in parts.
- As development activities for next release and use of early version of product is done simultaneously, if found errors can be corrected.
- Customers or end users get the chance to see the useful functionality early in the software development life cycle.
- As a result of end user's feedback requirements for successive releases become more clear.
- As functionality is incremented in steps, testing also becomes easy.
- Risk of failure of a product is decreased as users start using the product early.

Disadvantages

- As product is delivered in parts, total development cost is higher.
- Well defined interfaces are required to connect modules developed with each phase.
- The model requires well defined project planning schedule to distribute the work properly.
- Testing of modules also results into overhead and increased cost.

E. The Spiral Model

The Spiral model is also one of the popular evolutionary process model used by the industry. The Spiral model was proposed by Boehm in 1988 and is a popular model used for large size projects. The model focuses on minimizing the risk through the use of prototype. One can view the Spiral model as a Waterfall model with each stage preceded by the risk analysis stage. A simplified view of Spiral model is shown in fig. 7.

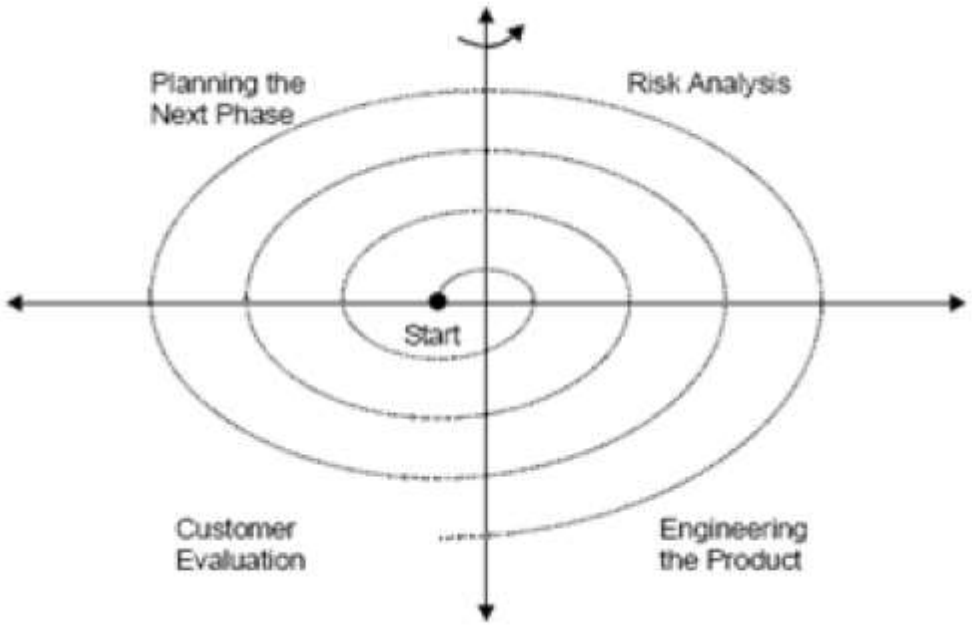


Fig. 7: The Spiral model

The radial coordinate in the diagram represents the total costs incurred till date. Each loop of the spiral represents one phase of the development. The model is divided into four quadrants, each with a specific purpose. Each spiral represents the progress made in the project The advantages and disadvantages of spiral model are listed below:

Advantages

- The model tries to resolve all possible risks involved in the project starting with the highest risk.
- End users get a chance to see the product early in life cycle.
- With each phase as product is refined after customer feedback, the model ensures a good quality product.
- The model makes use of techniques like reuse, prototyping and component based design

Disadvantages

- The model requires expertise in risk management and excellent management skills.
- The model is not suitable for small projects as cost of risk analysis may exceed the actual cost of the project.
- Different persons involved in the project may find it complex to use. Spiral model is generally used for large projects with medium to high risk because in small projects it is possible that cost

of risk analysis may exceed the actual cost of project. In other words, Spiral model is a practical approach for solving large scale software development related problems. This can also be used if requirements of the project are very complex or if the company is planning to introduce new technologies. Some areas where

- Spiral model is successfully used are decision support system, defense, aerospace, and large business projects.

F. The Rapid Application Development (RAD) Model

The Rapid Application Development (RAD) model was proposed by IBM in 1980s and later on was introduced to software community by James Martin through his book Rapid Application development. The important feature of RAD model is increased involvement of the user/customer at all stages of life cycle through the use of powerful development tools. Block diagram of RAD model is shown in fig. 8.

The RAD model consists of following four phases:

1. Requirements Planning – focuses on collecting requirements using elicitation techniques like brainstorming,
2. User Description – Requirements are detailed by taking users feedback by building prototype using development tools.
3. Construction – The prototype is refined to build the product and released to the customer.
4. Cutover – involves acceptance testing by the user and their training.

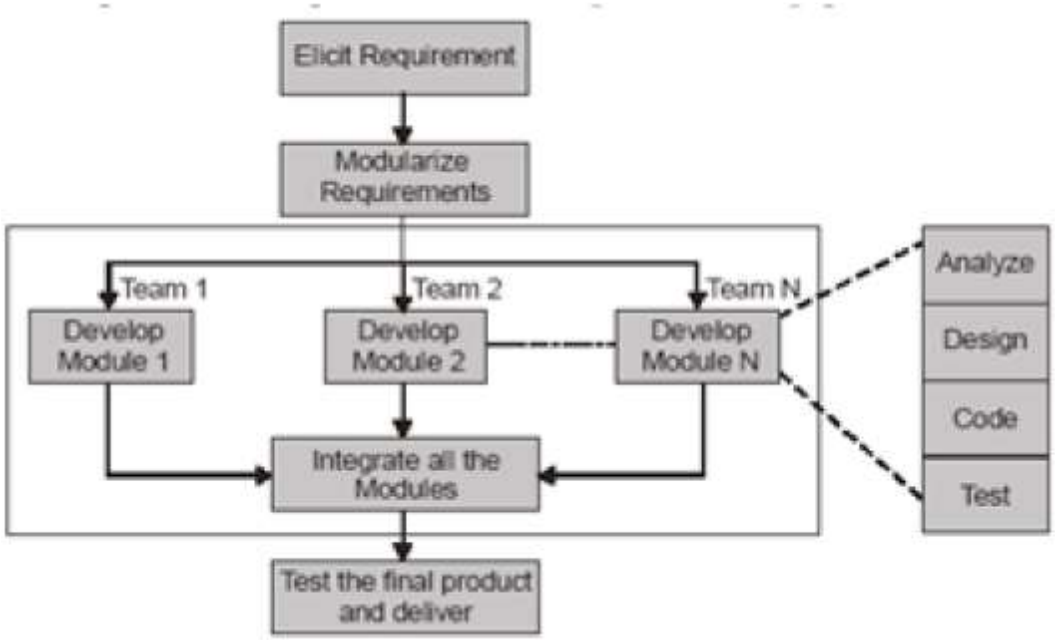


Fig. 8: The RAD MODEL

The advantages and disadvantages of RAD model are discussed below:

Advantages

- As customer is involved at all stages of development, it leads to a product achieving customer satisfaction.
- Usage of powerful development tools results into reduced software development cycle time.
- Feed back from the customer/user is available at the initial stages.
- Makes use of reusable components, to decrease the cycle time.
- Results into reduced costs as less developers are required.

Disadvantages

- The model makes use of efficient tools, to develop the prototype quickly, which calls for hiring skilled
- professional.
- Team leader must work closely with developers and customers/users to close the project in time.
- Absence of reusable components can lead to failure of the project.

The RAD model should be used for system with known requirements and requiring short development time. Also It is appropriate to use RAD model for the system that can be modularized and also reusable components are available for development. The model can also be used when already existing system components can be reused in developing a new system with minimum changes. Comparison of different process models in tabular form is shown in Table 1.

Table 1: Comparison of Software Process Model

		Water fall	V Model	Increm-ental	Spiral	Proto type	RAD
1	Well Defined Requirement	Yes	Yes	No	No	No	Yes
2	Domain Knowledge of Team member	Adeq-uate	Adeq-uate	Adeq-uate			Adeq-uate
3	Expertise of Users in Problem Domain	Very Less	Very Less	Adeq-uate	Very Less	Adeq-uate	Adeq-uate
4	Availability of Reusable Components	No	No	No	Yes	Yes	Yes
5	User involvement in all SDLC Phase	No	No	No	No	Yes	Yes
6	Complexity of the System	Simple	Simple	Complex	Complex	Complex	Medium

III. Software Lifecycle Model Selection Criteria

ISO/IEC TR 197592 identifies references by Comer et al [7], Davis et al[8] and Alexander et al [1] as providing comparisons between different software lifecycle models. Davis et al [8] and Alexander et al [1] are discussed , along with various relevant software engineering standards and other references. Unfortunately it was not possible to obtain Comer et al [7], which may include additional useful information. Software lifecycle model selection is not mentioned in CAP 67014 or Def Stan00-56 Issue 429, 30.

A. The Importance of Software Lifecycle Model Selection

The selection for a software lifecycle model for a project is an important decision [5]. It impacts on project success by affecting the following:

- The software lifecycle's overall cost [3];
- The distribution of cost over the software lifecycle [3];
- Software development speed [9];
- Software quality [9-3];
- The ability to tracking and control the project [9];
- The project's overhead [9];
- The level of risk associated with the project [9];
- Client relations [9].

No single software lifecycle model is appropriate for all situations [5, 4, 9, 2, 6, 3]. This is due to the diversity in project, system and organizational characteristics [3]. This fact is reflected by the standards. None of these references prescribe the use of a particular software lifecycle model. Because of this, a software lifecycle model must to be selected to suit each project's characteristics [5, 8, 4, 6].

B. ISO/IEC FDIS 1107:2007

ISO/IEC FDIS 1107:20074 does not mandate the use of a particular software lifecycle model. Neither does it mandate the software lifecycle phases. It does state that an appropriate software lifecycle model should be defined for a project [4]. The standard's users are responsible for selecting this model and for mapping processes, activities and tasks.. ISO/IEC.

FDIS 1107:20074 states that it encourages iteration between lifecycle activities. This implies that the selected model should allow iteration between phases. ISO/IEC FDIS 1107:20074 states that it prefers the use of software lifecycle models defined by an organization for use on multiple projects. Policies

and procedures for deploying the models in projects will also be defined during this process. No guidance is provided on the contents of these policies and procedures however. The selection of a software lifecycle model for a particular project from the organizationally-defined models occurs during the Project Planning Process.

The following attributes are identified as influencing software lifecycle model selection:

- Project scope;
- Project size;
- Project complexity;
- Changing needs and opportunities.

C. CMMI-DEV

CMMI-DEV [6] does not prescribe the use of a particular lifecycle model. Its approach to life cycle model selection for a project is very similar to that of ISO/IEC FDIS 1107:20074. CMMI-DEV [6] states that tailoring a lifecycle model for a project could include modifying it, or combining it with elements of another lifecycle model. This is hybridization, CMMI-DEV [6] does not specify lifecycle model selection criteria, but states that the following project characteristics may affect lifecycle model selection:

- Project size;
- Staff experience and familiarity with the lifecycle model;
- Project constraints such as time and acceptable defect levels.

D. IEEE 1074-2006

ISO/IEC FDIS 1107:20074 states that IEEE 1074-200613 may be useful in implementing its Life Cycle Model Management Process. The IHS website [11] briefly describes IEEE 1074-200613 defines a methodology for developing a software lifecycle process for a particular project (referred to as a software project life cycle process, or SPLCP) [11].

The approach appears similar to that described in ISO/IEC FDIS 1107:20074 and CMMI-DEV6. IEEE 1074-200613 does not mandate the use of a particular software lifecycle model[11].

E. TickIT Guide

The TickIT Guide [5] does not mandate the use of a particular software lifecycle model. It does state that an appropriate software lifecycle model should be defined for a project [5]. It states that this model should

be tailored to the project's characteristics and documented [5]. Lifecycle model tailoring is consistent with CMMI-DEV [6]. The TickIT Guide [5] states that the software lifecycle models used by an organization will typically be documented as part of the organization's quality management system. This is consistent with the use of organizationally-defined models in ISO/IEC FDIS 1107:20074 and CMMI-DEV [6]. It also states that if an organization uses more than one software lifecycle model, criteria for the selection of a particular model should be defined [5]. It identifies the following as factors in the selection of a model [5]:

- The size and type of system being developed;
- The project requirements as understood at the start of the project.

The TickIT Guide [5] states that in some situations a combination of several models may be appropriate (i.e. hybridisation).

E. IEC61508

IEC61508 Part 38 allows the use of a software lifecycle model. The software lifecycle model must also allow the objectives and requirements of all IEC61508 Part 38 clauses to be met 6. If any of these objectives and requirements is not met, this must be justified6. IEC61508 Part 38 also states that the software lifecycle model may be customised to suit the project and organisation 8. IEC61508 therefore only provides a limited guidance on software lifecycle model selection.

F. IEC61511

IEC615113 [9] states that a software lifecycle model should be specified for developing application software. It states that this should be done as part of the safety planning activity. This model must integrate with the system lifecycle model. No software lifecycle model selection criteria are mentioned.

G. DO-118B

DO-118B [10] does not specify the use of a particular software lifecycle model. It states that a number of separate and different software lifecycle models may be used to develop different components of a single software product10. DO-118B [10] states that appropriate software lifecycle models should be selected during the software planning process. These should be appropriate to the project's characteristics. It identifies the following as factors that should influence software lifecycle model selection:

- System functionality;
- System and software complexity;

- Software size;
- Requirements stability;
- Use of previously developed results;
- Development strategies;
- Hardware availability.

DO-118B [10] does not explain what is meant by development strategies.

H. McConnell

McConnell [9] lists a number of questions that he states ought to be answered when selecting a software lifecycle model for a project. He also defines a set of software lifecycle model selection criteria against which the answers to these questions can be compared. A single criterion corresponds to each question (with the exception of the "Has low overhead" criterion). McConnell's questions are reproduced below:

- How well do my customer and I understand the requirements at the beginning of the project?; is our
- understanding likely to change significantly as we move through the project?";
- "How well do I understand the system architecture? Am I likely to need to make major architectural changes midway through the project?";
- "How much reliability do I need?";
- "How much do I need to plan ahead and design ahead during this project for future versions?";
- "How much risk does this project entail?";
- "Am I constrained to a predefined schedule?";
- "Do I need to be able to make midcourse corrections?";
- "Do I need to provide my customers with visible progress throughout the project?";
- "Do I need to provide management with visible progress throughout the project?";
- "How much sophistication do I need to use this lifecycle model successfully?".

McConnell [9] states that development speed will increase the more closely the software lifecycle conforms to the Waterfall model, provided that this can be used effectively. He also suggests that weaknesses identified in a particular model could be addressed by hybridizing it with other models.

I. Alexander et al

Alexander et al [1] state that software lifecycle models are often selected on an ad hoc basis using a set of undocumented and unjustified criteria. Given the general lack of detailed guidance provided by the references reviewed in this literature survey, this seems a reasonable statement. Alexander et al [1]

propose twenty “criteria” for selecting an appropriate software lifecycle model for a project. These criteria are actually a set of project characteristics (e.g. “Problem Complexity”).

- The criteria fall into five categories:
- Personnel: criteria relating to software’s users and developers;
- Problem: criteria relating to the problem the software solves;
- Product: criteria relating to software itself;
- Resource: criteria relating to the resources available to develop the software;
- Organizational: criteria relating to the impact of organizational policies on software • development.
- Alexander et al [1] identify number of key factors to be considered when answering these questions:
- The opportunity to modify, validate and/or verify the software;
- The degree of functionality delivered and when;
- The level of activity with respect to time.

J. Davis et al

Davis et al [8] propose the following set of metrics for comparing alternative software lifecycle models:

- Shortfall: the difference, at a particular point in time, between the user’s actual needs and the user needs the software meets;
- Lateness: the time that elapses between a new user need arising and the software satisfying that need;
- Adaptability: the rate at which software meets new user needs;
- Longevity: the length of time during which it is viable to modify the software. This is a measure of the length of time from the software first entering service to it being withdrawn;
- Inappropriateness: the integral of shortfall over a period of time.

Davis et al [8] use these metrics to compare the use of the Waterfall model against the following: Throwaway Prototyping; Incremental Development; Evolutionary Prototyping; Reusable Software and Automated Software Synthesis.

Davis et al’s[8] approach has a number of shortcoming. Firstly, it makes general statements about the relative effectiveness of various software lifecycle models without considering the project circumstances in which they are applied.

IV. Conclusions

The following key conclusions can be drawn from the survey: • Software lifecycle models are high-level abstractions of the real software lifecycle. They represent the sequence and interrelationship of broad phases within the lifecycle. These phases represent processes directly concerned with the production and verification of software. Software lifecycle models typically do not represent processes that occur continuously during the software lifecycle. They also do not represent the details of how each phase is performed (i.e. the methods and tools to be used);

- Software lifecycle models principally act as tools to help manage the software lifecycle. They also promote a common understanding of the lifecycle amongst stakeholders and help to promote process consistency and improvement;

- A number of alternative software lifecycle models have been proposed. Additionally, two or more models can be hybridized. Each model has its own strengths and weaknesses. Some models are more appropriate in certain project circumstances than others. No single model is appropriate to all circumstances. The methods and tools to be used in combination with a software lifecycle model
- affect its effectiveness. A software lifecycle model therefore must be selected for each project based on the project's characteristics and methods and tools to be used;

- The selection of a software lifecycle model for a project is an important decision. The use of an inappropriate model can be detrimental to project success and software quality;

A systematic process has been proposed in an academic paper (Alexander et al1) but this is unlikely to be well known to practitioners. This process has good features but also limitations.

References

[1] Alexander, L, Davis, A, "Criteria for Selecting Software Process Models", presented at COMPSAC 1991.

[2] International Organisation for Standardization/ International Electrotechnical Commission 2004. Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK),ISO/IEC TR, 1975.

[3] Somerville, I, "Software Engineering", 8th edn., Addison Wesley, 2006.

[4] International Organisation for Standardization/ International Electrotechnical Commission 2007. Systems and software engineering — Software life cycle processes, ISO/IEC FDIS 1107, 2007 (E) (Final Draft)

[5] British Standards Institution 2001. The TickIT Guide. Issue 5.0.

[6] Carnegie Mellon University 2006. CMMI® for Development Version 1.2, CMMI-DEV V1.2

[7] Comer, E,“Alternative Software Life Cycle Models”, presented at International Conference on Software Engineering 1997.

[8] Davis, A, Bersoff, E, Comer, E, “A Strategy for Comparing Alternative Software Development Life Cycle Models”, IEEE Transactions on Software Engineering, vol.14, iss.10, pp. 1453-1461, 1988

[9] McConnell, S, "Rapid Development", Redmond: Microsoft Press, 1996

[10] RTCA,"Software Considerations in Airborne Systems and Equipment Certification. DO-118B, 1992

[11] IHS website. [Online] Available: http://electronics.ihs.com/document/abstract/OGLJFAAAAAAAAAA. 2009

[12] Boehm, B, W, ‘A spiral model of software development and enhancement’, IEEE Computer, Vol.1, Issue 5, pp. 61-72. 1988

[13] International Organisation for Standardization, "Quality management systems", Fundamentals and vocabulary, BS EN ISO [9]000:2005