I.J.E.M.S., VOL.3 (4) 2012: 500-506 ISSN 2229-600X

INTERNATIONAL JOURNAL OF ENGINEERING AND MANAGEMENT SCIENCES

ASSESSMENT OF SOFTWARE PROCESS MODELS

Akhilesh
Research Scholar, Department of Computer Science, Manav Bharti University, Solan (H.P.)

ABSTRACT
The field of software engineering is related to the development of software. Large software needs systematic development unlike simple programs which can be developed in isolation and there may not be any systematic approach being followed. In the last few decades, the computer industry has undergone revolutionary changes in hardware. This research deals with a critical and important issue in computer world. It is concerned with the software management processes that examine the area of software development through the development models, which are known as software development life cycle. It represents five of the development models namely, waterfall, Iteration, V-shaped, spiral and Extreme programming. These models have advantages and disadvantages as well. Therefore, the main objective of this research is to represent different models of software development and make a comparison between them to show the features and defects of each model.

KEY WORDS: Software Management Processes, software Development, Development Models, Software Development Life Cycle.

INTRODUCTION

Any application on computer runs through software. As computer technologies have changed tremendously in the last five decades, accordingly, the software development has undergone significant changes in the last few decades of 20th century. In the early years, the software size used to be small and those were developed either by a single programmer or by a small programming team. The program development was dependent on the programmer's skills and no strategic software practices were present. In the early 1980s, the size of software and the application domain of software increased. Consequently, its complexity has also increased. Bigger teams were engaged in the development of Software. The software development became more bit organized and software development management practices came into existence.

During the previous four decades, software has been developed from a tool used for analyzing information or solving a problem to a product in itself. However, the early programming stages have created a number of problems turning software an obstacle to software development particularly those relying on computers. Software consists of documents and programs that contain a collection that has been established to be a part of software engineering procedures. Moreover, the aim of software engineering is to create a suitable work that construct programs of high quality.

SOFTWARE PROCESS MODELS

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective as:

1. Specification.
2. Design.
3. Validation.
4. Evolution.

General Software Process Models are

1. Waterfall model: Separate and distinct phases of specification and development.
2. Prototype model.
3. Rapid application development model (RAD).
4. Evolutionary development: Specification, development and validation are interleaved.
5. Incremental model.
6. Iterative model.
7. Spiral model.
8. Component-based software engineering : The system is assembled from existing components.

There are many variants of these models e.g. formal development where a waterfall-like process is used, but the specification is formal that is refined through several stages to an implementable design[1].

FIVE MODELS

Software Engineering deals with the development of software. Hence, understanding the basic characteristics of software is essential. Software is different from other engineering products in the following ways

1. Engineering products once developed cannot be changed. To modifications the product, redesigning and remanufacturing is required. In the case of software, ultimately changes are to be done in code for any changes to take effect.
2. The Other Engineering products are visible but the software as such is not visible. That's why, it is said that software is developed, but not manufactured. Though, like other products, it is first designed, then produced, it cannot be manufactured automatically on an assembly line like other engineering products. Nowadays, CASE (Computer Aided Software Engineering) tools are available for software development. Still it depends on the programmer's skill and creativity. The creative skills of the programmer is difficult to quantify and standardise.

Hence, the same software developed by different programmers may take varying amount of time, resources and may have variable cost.

3. Software does not fail in the traditional sense. The engineering products has wear and tear in the operation. Software can be run any number of times without wear and tear. The software is considered as failed if:
   a) It does not operate correctly.
   b) Does not provide the required number of features.
4. Engineering products can be perfectly designed, but in the case of software, however good the design, it can never be 100% error free. Even the best quality software is not completely error free. Software is called good quality software if it performs the required operation, even if it has a few errors.
5. The testing of normal engineering products and software engineering products are on different parameters. In the former, it can be full load testing, etc., whereas in the case of software, testing means identification of test cases in which software may fail. Thus, testing of software means running of software for different inputs. By testing, the presence of errors is identified.
6. Unlike most of the other engineering products, software can be reused. Once a piece of code is written for some application, it can be reused.
7. The management of software development projects is a highly demanding task, since it involves the assessment of the developers creative skills. The estimation regarding the time and cost of software needs standardization of developers creativity, which can be a variable quantity. It means that software projects cannot be managed like engineering products. The correction of a bug in the case of software may take hours But, it may not be the case with normal engineering products.
8. The Software is not vulnerable to external factors like environmental effects. But the same external factors may harm hardware. The hardware component may be replaced with spare parts in the case of failure, whereas the failure of a software component may indicate the errors in design.

Thus, the characteristics of software are quite different from other engineering products. Hence, the software industry is quite different from other industries. There are numbers of general models for software processes, like: Waterfall model, Evolutionary development, Formal systems development and Reuse- based development, etc. This research will view the following five models :

1. Waterfall model.
2. Iteration model.
3. V-shaped model.
4. Spiral model.
5. Extreme model.

These models are chosen because their features correspond to most software development programs.

THE WATERFALL MODEL

The waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern.

The pure waterfall lifecycle consists of several non-overlapping stages, as shown in the following figure. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance. The waterfall model serves as a baseline for many other lifecycle models.

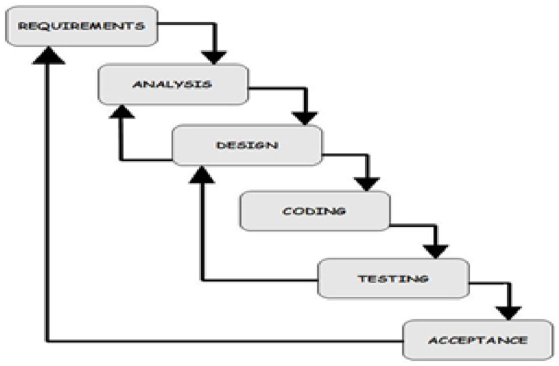

Fig. 2 Waterfall Model[4].

The following list details the steps for using the waterfall model:

Phase I: Requirements

The first phase involves understanding what you need to design and what is its function, purpose etc. Unless you know what you want to design, you cannot proceed with the project. Even a small code such as adding two integer numbers, needs to be written with the output in mind. Here, in this stage, the requirements which the software is going to satisfy are listed and detailed. These requirements are then presented to the team of programmers. If this phase is completed successfully, it ensures a smooth working of the remaining phases, as the programmer is not burdened to make changes at later stages because of changes in requirements

Phase II: Analysis

As per the requirements, the software and hardware needed for the proper completion of the project is analyzed in this phase. Right from deciding which computer language should be used for designing the software, to the database system that can be used for the smooth functioning of the software, such features are decided at this stage.

Phase III: Design

The algorithm or flowchart of the program or the software code to be written in the next stage, is created now. It is a very important stage, which relies on the previous two stages for its proper implementation. The proper design at this stage, ensures a execution in the next stage. If during the design phase, it is noticed that there are some more requirements for designing the code, the analysis phase is revisited and the design phase is carried out according to the new set of resources.

Phase IV: Coding

Based on the algorithm or flowchart designed, the actual coding of the software is carried out. This is the stage where the idea and flowchart of the application is physically created or materialized. A proper execution of the previous stages ensures a smooth and easier implementation of this stage.

Phase V: Testing

With the coding of the application complete, the testing of the written code now comes into scene. Testing checks if there are any flaws in the designed software and if the software has been designed as per the listed specifications. A proper execution of this stage ensures that the client interested in the created software, will be satisfied with the finished product. If there are any flaws, the software development process must step back to the design phase. In the design phase, changes are implemented and then the succeeding stages of coding and testing are again carried out.

Phase VI: Acceptance

This is the last stage of the software development in the waterfall model. A proper execution of all the preceding stages ensures an application as per the provided requirements and most importantly, it ensures a satisfied client. However, at this stage, you may need to provide the client with some support regarding the software you have developed. If the client demands further enhancements to be made to the existing software, then the development process must begin anew, right from the first phase, i.e., requirements.

The waterfall model continues to remain one of the most commonly used methodologies. No doubt, new models have been used, but the widespread use of this model is the reason why it is studied in various software management subjects. With the above diagram in hand, you will not have much difficulty in understanding the process of software development. This is not only one of the simplest software process models for application development, but it is also known for its ease of implementation in the field of software development.

In some organizations, a change control board maintains the quality of the product by reviewing each change made in the maintenance stage. Consider applying the full waterfall development cycle model when correcting problems or implementing these enhancement requests.

In each stage, documents that explain the objectives and describe the requirements for that phase are created. At the end of each stage, a review to determine whether the project can proceed to the next stage is held. Your prototyping can also be incorporated into any stage from the architectural design and after.

Many people believe that this model cannot be applied to all situations. For example, with the pure waterfall model, the requirements must be stated before beginning the design, and the complete design must be stated before starting coding. There is no overlap between stages. In real-world development, however, one can discover issues during the design or coding stages that point out errors or gaps in the requirements.

The waterfall method does not prohibit returning to an earlier phase, for example, returning from the design phase to the requirements phase. However, this involves costly rework. Each completed phase requires formal review and extensive documentation development. Thus, oversights made in the requirements phase are expensive to correct later.

Because the actual development comes late in the process, one does not see results for a long time. This delay can be disconcerting to management and customers. Many people also think that the amount of documentation is excessive and inflexible.

Although the waterfall model has its weaknesses, it is instructive because it emphasizes important stages of project development. Even if one does not apply this model, he must consider each of these stages and its relationship to his own project [4].

Advantages

1. Easy to understand and implement.
2. Widely used and known (in theory!).
3. Reinforces good habits: define-before- design, design-before-code.
4. Identifies deliverables and milestones.
5. Document driven, URD, SRD, … etc. Published documentation standards, e.g. PSS-05.
6. Works well on mature products and weak teams.

Disadvantages :

1. Idealized, doesn't match reality well.
2. Doesn't reflect iterative nature of exploratory development.
3. Unrealistic to expect accurate requirements so early in project.
4. Software is delivered late in project, delays discovery of serious errors.
5. Difficult to integrate risk management.
6. Difficult and expensive to make changes to documents, "swimming upstream".
7. Significant administrative overhead, costly for small teams and projects [6].

PURE WATERFALL

This is the classical system development model. It consists of discontinuous phases:

1. Concept.
2. Requirements.
3. Architectural design.
4. Detailed design.
5. Coding and development.
6. Testing and implementation.

Table 1: Strengths & Weaknesses of Pure Waterfall

Strengths	Weaknesses
• Minimizes planning overhead since it can be done up front. • Structure minimizes wasted effort, so it works well for technically weak or inexperienced staff.	• Inflexible • Only the final phase produces a non-documentation deliverable. • Backing up to address mistakes is difficult.

The pure waterfall model performs well for products with clearly understood requirements or when working with well understood technical tools, architectures and infrastructures. Its weaknesses frequently make it inadvisable when rapid development is needed. In those cases, modified models may be more effective.

MODIFIED WATERFALL

The modified waterfall uses the same phases as the pure waterfall, but is not based on a discontinuous basis. This enables the phases to overlap when needed. The pure waterfall can also split into subprojects at an appropriate phase (such as after the architectural design or detailed design).

Strengths	Weaknesses
• More flexible than the pure waterfall model. • If there is personnel continuity between the phases, documentation can be substantially reduced. • Implementation of easy areas does not need to wait for the hard ones.	• Milestones are more ambiguous than the pure waterfall. • Activities performed in parallel are subject to miscommunication and mistaken assumptions. • Unforeseen interdependencies can create problems.

Table 2: Strengths & Weaknesses of Modified Waterfall

Risk reduction spirals can be added to the top of the waterfall to reduce risks prior to the waterfall phases. The waterfall can be further modified using options such as prototyping, JADs or CRC sessions or other methods of requirements gathering done in overlapping phases [5].

ITERATIVE DEVELOPMENT

The problems with the Waterfall Model created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With Iterative Development, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products, which are produced at the end of each step (or series of steps), can go into production immediately as incremental releases.

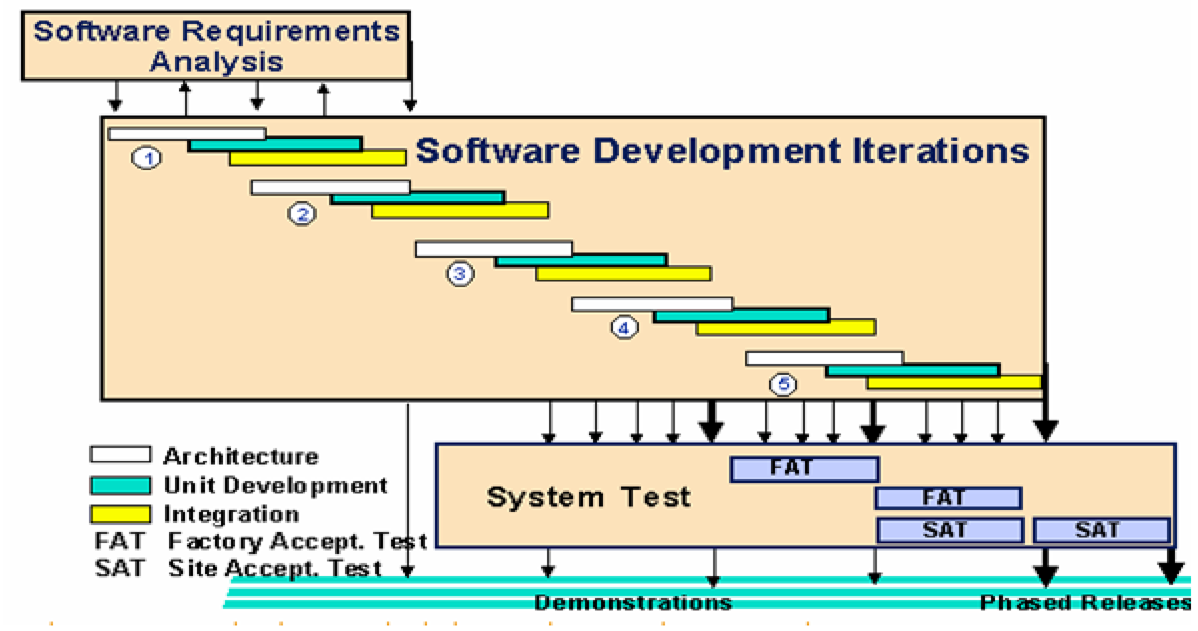

Fig. 4 Iterative Development.

V-SHAPED 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. Testing is emphasized in this model more than the waterfall model. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in requirements gathering.

The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Advantages

1. Simple and easy to use.
2. Each phase has specific deliverables.

3. Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.
4. Works well for small projects where requirements are easily understood.

Disadvantages
1. Very rigid like the waterfall model.
2. Little flexibility and adjusting scope is difficult and expensive.
3. Software is developed during the implementation phase, so no early prototypes of the software are produced.
4. This Model does not provide a clear path for problems found during testing phases [7].

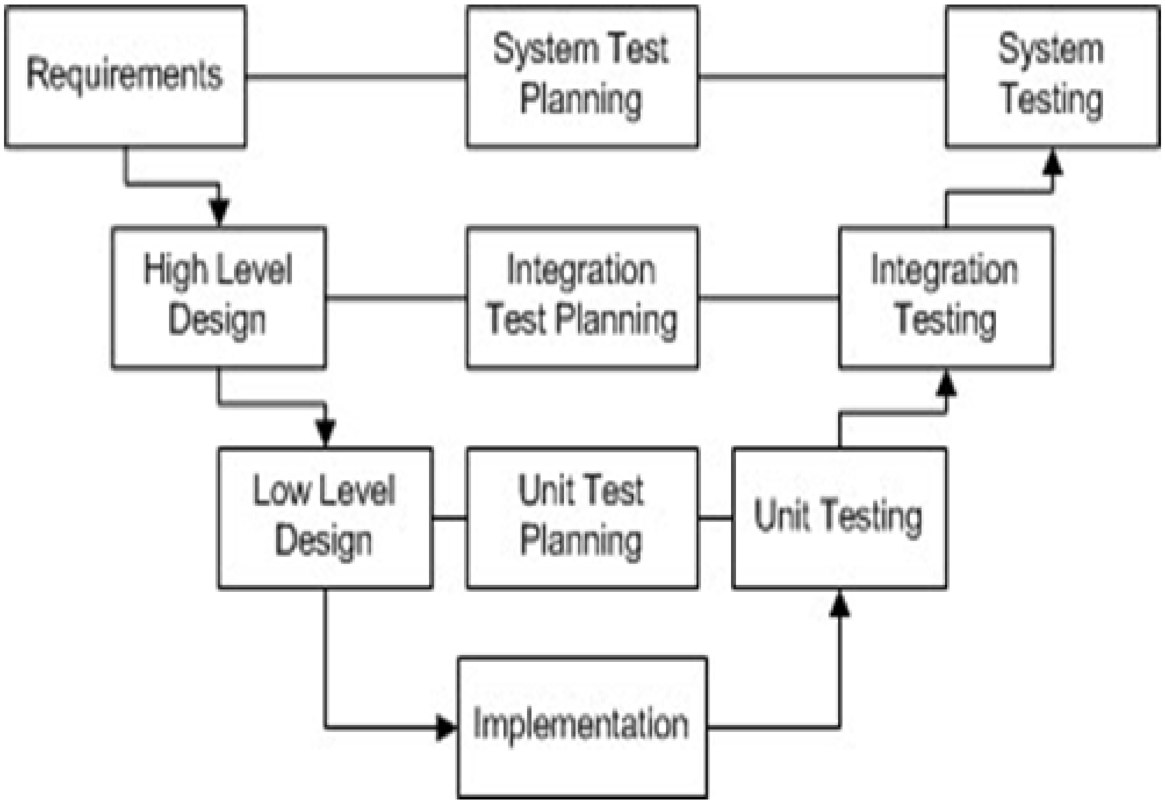

Fig. 6 V-Shaped Life Cycle Model[7].

SPIRAL MODEL
The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

Advantages
1. High amount of risk analysis.
2. Good for large and mission-critical projects.
3. Software is produced early in the software life cycle.

Disadvantages
1. Can be a costly model to use.
2. Risk analysis requires highly specific expertise.
3. Project’s success is highly dependent on the risk analysis phase.
4. Doesn’t work well for smaller projects [7].

Spiral model sectors
1. Objective setting :Specific objectives for the phase are identified.
2. Risk assessment and reduction: Risks are assessed and activities are put in place to reduce the key risks.
3. Development and validation: A development model for the system is chosen which can be any of the general models.
4. Planning: The project is reviewed and the next phase of the spiral is planned [1].

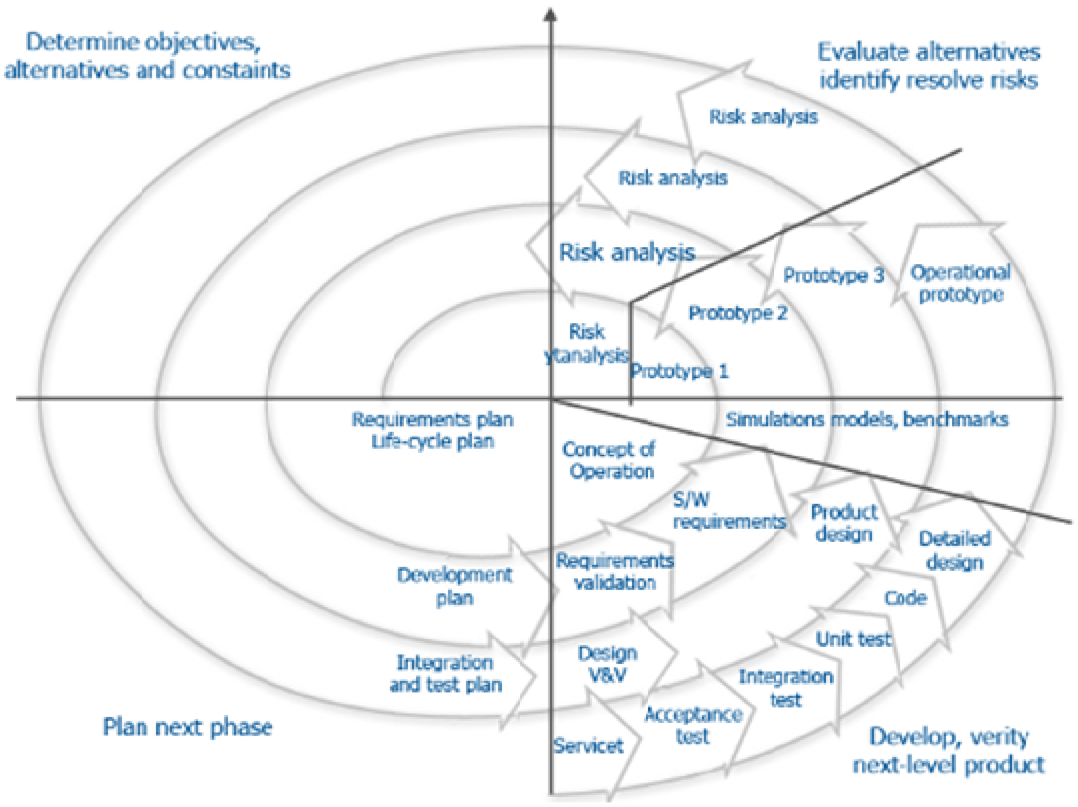

Fig. 7 Spiral Model of the Software Process[1].

WIN WIN SPIRAL MODEL

The original spiral model began each cycle of the spiral by performing the next level of elaboration of the prospective system's objectives, constraints and alternatives. A primary difficulty in applying the spiral model has been the lack of explicit process guidance in determining these objectives, constraints, and alternatives. The Win-Win Spiral Model uses the theory W (win-win) approach to converge on a system's next-level objectives, constraints, and alternatives. This Theory W approach involves identifying the system's stakeholders and their win conditions, and using negotiation processes to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders. In particular, as illustrated in the figure, the nine-step Theory W process translates into the following spiral model extensions:

1. **Determine Objectives**: Identify the system life-cycle stakeholders and their win conditions and establish initial system boundaries and external interfaces.
2. **Determine Constraints**: Determine the conditions under which the system would produce win-lose or lose- lose outcomes for some stakeholders.
3. **Identify and Evaluate Alternatives**: Solicit suggestions from stakeholders, evaluate them with respect to stakeholders' win conditions, synthesize and negotiate candidate win-win alternatives, analyze, assess, resolve win-lose or lose-lose risks, record commitments and areas to be left flexible in the project's design record and life cycle plans.
4. **Cycle through the Spiral**: Elaborate the win conditions evaluate and screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans [8].

EXTREME PROGRAMMING

An approach to development, based on the development and delivery of very small increments of functionality. It relies on constant code improvement, user involvement in the development team and pair wise programming . It can be difficult to keep the interest of customers who are involved in the process. Team members may be unsuited to the intense involvement that characterizes agile methods. Prioritizing changes can be difficult where there are multiple stakeholders. Maintaining simplicity requires extra work. Contracts may be a problem as with other approaches to iterative development.

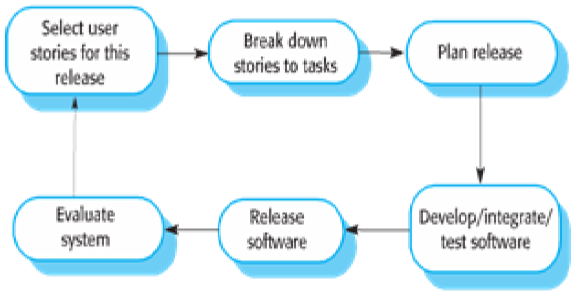
Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Fig. 8 The XP Release Cycle

EXTREME PROGRAMMING PRACTICES

Incremental planning: Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "Tasks".

Small Releases: The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

Simple Design: Enough design is carried out to meet the current requirements and no more.

Test first development: An automated unit test framework is used to write tests for a new piece of functionality before functionality itself is implemented.

Refactoring: All developers are expected to re-factor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Pair Programming: Developers work in pairs, checking each other's work and providing support to do a good job.

Collective Ownership: The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.

Continuous Integration: As soon as work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.

Sustainable pace: Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.

On-site Customer: A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

1. Incremental development is supported through small, frequent system releases.
2. Customer involvement means full-time customer engagement with the team.
3. People not process through pair programming, collective ownership and a process that avoids long working hours.
4. Change supported through regular system releases.
5. Maintaining simplicity through constant refactoring of code [1].

Advantages

1. Lightweight methods suit small-medium size projects.
2. Produces good team cohesion.
3. Emphasises final product.
4. Iterative.
5. Test based approach to requirements and quality assurance.

Disadvantages

1. Difficult to scale up to large projects where documentation is essential.
2. Needs experience and skill if not to degenerate into code-and-fix.
3. Programming pairs is costly.
4. Test case construction is a difficult and specialized skill [6].

CONCLUSION AND FUTURE WORK

After completing this research , it is concluded that :

1. There are many existing models for developing systems for different sizes of projects and requirements.
2. These models were established between 1970 and 1999.
3. Waterfall model and spiral model are used commonly in developing systems.
4. Each model has advantages and disadvantages for the development of systems , so each model tries to eliminate the disadvantages of the previous model

Finally, some topics can be suggested for future works:

1. Suggesting a model to simulate advantages that are found in different models to software process management.
2. Making a comparison between the suggested model and the previous software processes management models.
3. Applying the suggested model to many projects to ensure of its suitability and documentation to explain its mechanical work.

REFERENCES

[1] Ian Sommerville, "Software Engineering", Addison Wesley, 7th edition, 2004.

[2] CTG. MFA – 003, "A Survey of System Development Process Models", Models for Action Project: Developing Practical Approaches to Electronic Records Management and Preservation, Center for Technology in Government University at Albany / Suny,1998 .

[3] Steve Easterbrook, "Software Lifecycles", University of Toronto Department of Computer Science, 2001.

[4] National Instruments Corporation, "Lifecycle Models", 2006, http://zone.ni.com.

[5] JJ Kuhl, "Project Lifecycle Models: How They Differ and When to Use Them",2002 esolutions.com.

[6] Karlm, "Software Lifecycle Models', KTH,2006 .

[7] Rlewallen, "Software Development Life Cycle Models", 2005 ,http://codebeter.com.

[8] Barry Boehm, "Spiral Development: Experience, Principles, and Refinements", edited by Wilfred J. Hansen, 2000.