SWOT Analysis of Software Development Process Models

Ashish B. Sasankar[1], **Dr Vinay Chavan**[2]

[1] **P.G. Department of Computer Science ,GHRIIT**
Nagpur, Maharashtra, India

[2] **Department of Computer Science,S.K.Porwal College,Kamptee**
Nagpur, Maharashtra, India

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 has 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 using SWOT method. The objective is to identify Strength ,Weakness ,Opportunities and Threats of Waterfall, Spiral, Prototype etc.

Keywords: *SDLC,SWOT.*

1. Introduction

Software lifecycle models are representations of the sequence and interrelationship of broad phases within the software lifecycle. Their principal purpose is to provide a high-level plan for software lifecycle activities. They are therefore essentially management tools. The use of a software lifecycle model on a software project is important. Without the plan it provides, it can be difficult to effectively manage the project.

Within the field of Computer Science, a large number of software lifecycle models have been proposed. Each model has its own strengths and weaknesses, and each is more appropriate in certain project circumstances than others. It is generally recognised that no single software lifecycle model is appropriate in all circumstances. Because of this, for a particular software project, it is necessary to select a software lifecycle model that suits the project's characteristics. This is an important decision. The use of an inappropriate software lifecycle model can increase project costs and timescales and reduce software quality.

Now what a software lifecycle model is. Some definition are:

"framework of processes and activities concerned with the life cycle that may be organised into stages, which also acts as a common reference for communication and understanding" (ISO/IEC FDIS 12207:200726);

"A partitioning of the life of a product or project into phases." (CMMI-DEV36. This is the definition for a lifecycle model of any product or service. This may be software);

"software life cycle models serve as a high-level definition of the phases that occur during development. They are not aimed at providing detailed definitions but at highlighting the key activities and their interdependencies" (ISO/IEC TR 1975940);

"Lifecycle models describe the interrelationship between software development phases" (The NASA Software Safety Guidebook31);

2. Process Model/Life Cycle Variations

Professional system developers and the customers they serve share a common goal of building information systems that effectively support business process objectives. In order to ensure that cost-effective, quality systems are developed which address an organization's business needs, developers employ some kind of system development *Process Model* to direct the project's life cycle. Typical activities performed include the following:[1]

- System conceptualization
- System requirements and benefits analysis
- Project adoption and project scoping
- System design
- Specification of software requirements
- Architectural design
- Detailed design
- Unit development
- Software integration & testing

- System integration & testing
- Installation at site
- Site testing and acceptance
- Training and documentation
- Implementation
- Maintenance

Process Model/Life-Cycle Variations

While nearly all system development efforts engage in some combination of the above tasks, they can be differentiated by the *feedback* and *control methods* employed during development and the *timing of activities*. Most system development *Process Models* in use today have evolved from three primary approaches: *Ad-hoc Development*, *Waterfall Model*, and the *Iterative* process.

Ad-hoc Development

Early systems development often took place in a rather chaotic and haphazard manner, relying entirely on the skills and experience of the individual staff members performing the work. Today, many organizations still practice *Ad-hoc Development* either entirely or for a certain subset of their development (e.g. small projects).
The Software Engineering Institute at Carnegie Mellon University [2] points out that with *Ad-hoc Process Models*, "process capability is unpredictable because the software process is constantly changed or modified as the work progresses. Schedules, budgets, functionality, and product quality are generally (inconsistent). Performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations. There are few stable software processes in evidence, and performance can be predicted only by individual rather thanorganizational capability." [3]

Figure 1. Adhoc development

"Even in undisciplined organizations, however, some individual software projects produce excellent results. When such projects succeed, it is generally through the heroic efforts of a dedicated team, rather than through repeating the proven methods of an organization with a mature software process. In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project. Success that rests solely on the availability of specific individuals provides no basis for long-term productivity and quality improvement throughout an organization."[4]

2.1 The Waterfall Model

The *Waterfall Model* is the earliest method of structured system development. Although it has come under attack in recent years for being too rigid and unrealistic when it comes to quickly meeting customer's needs, the *Waterfall Model* is still widely used. It is attributed with providing the theoretical basis for other *Process Models*, because it most closely resembles a "generic" model for software development.

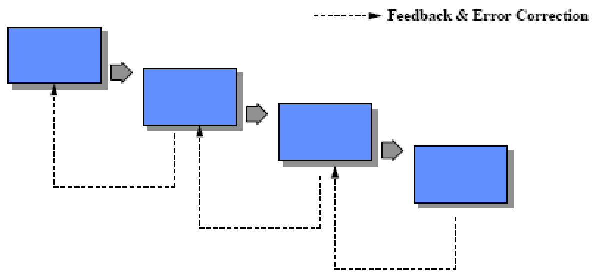


Figure 2 Waterfall model

The *Waterfall Model* consists of the following steps:

· **System Conceptualization.** System Conceptualization refers to the consideration of all aspects of the targeted business function or process, with the goals of determining how each of those aspects relates with one another, and which aspects will be incorporated into the system.

· **Systems Analysis.** This step refers to the gathering of system requirements, with the goal of determining how these requirements will be accommodated in the system. Extensive communication between the customer and the developer is essential.

· **System Design.** Once the requirements have been collected and analyzed, it is necessary to identify in detail how the system will be constructed to perform necessary tasks. More specifically, the System Design phase is focused on the data requirements (what information will be processed in the system?), the software construction (how will the application be constructed?), and the interface construction (what will the system look like? What standards will be followed?).

· **Coding.** Also known as programming, this step involves the creation of the system software. Requirements and systems specifications from the System Design step are translated into machine readable computer code.

· **Testing.** As the software is created and added to the developing system, testing is performed to ensure that it is working correctly and efficiently. Testing is generally focused on two areas: internal efficiency and external effectiveness. The goal of external effectiveness testing is to verify that the software is functioning according to system design, and that it is performing all necessary functions or sub-functions. The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented. Testing can be a labor-intensive process, due to its iterative nature.

Problems/Challenges associated with the Waterfall Model

Although the *Waterfall Model* has been used extensively over the years in the production of many quality systems, it is not without its problems. In recent years it has come under attack, due to its rigid design and inflexible procedure.

Criticisms fall into the following categories:

· Real projects rarely follow the sequential flow that the model proposes.

· At the beginning of most projects there is often a great deal of uncertainty about requirements and goals, and it is therefore difficult for customers to identify these criteria on a detailed level. The model does not accommodate this natural uncertainty very well.

· Developing a system using the *Waterfall Model* can be a long, painstaking process that does not yield a working version of the system until late in the process.

Critic

The waterfall model lacks prescribed technique of implementing management control over a project; planning, controlling, and risk management are not enveloped within the model itself. Moreover, forecasting the estimated time and cost are complicated for each stage. The life cycle can take long as the original requirements may no longer be valid, with little possibility for prototyping.

The waterfall model of system development works best when any reworking of products is kept to a minimum and the products remain unchanged. It still remains useful for steady and non-volatile types of projects, and if properly implemented, generates significant cost and timesaving. If the system is likely to go through significant changes and if the system requirements are unpredictable then different approaches are recommended, one such alternate approach is popularly know as the spiral model.

2.2 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.

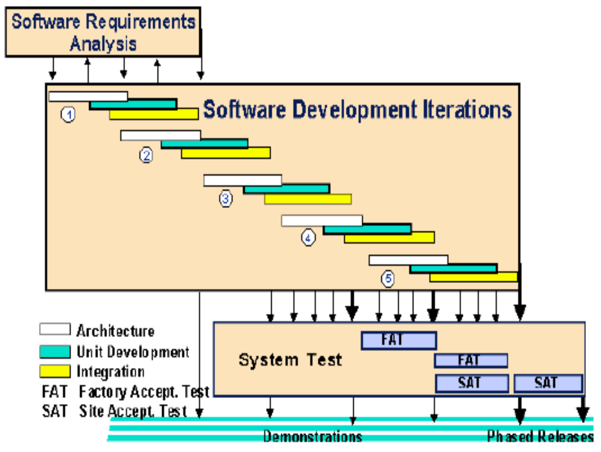


Figure 3. Iterative Development [5]

Problems/Challenges associated with the Iterative Model

While the *Iterative Model* addresses many of the problems associated with the *Waterfall Model*, it does present new challenges.

· The user community needs to be actively involved throughout the project. While this involvement is a positive for the project, it is demanding on the time of the staff and can add project delay.

· Communication and coordination skills take center stage in project development.

· Informal requests for improvement after each phase may lead to confusion -- a controlled mechanism for handling substantive requests needs to be developed.

· The *Iterative Model* can lead to “scope creep,” since user feedback following each phase may lead to increased customer demands. As users see the system develop, they may realize the potential of other system capabilities which would enhance their work.

Critic

One traditional process model is the waterfall model and according to Schacchi was only accepted just until the early 1980s because of its lack of functionality. The waterfall model is said to be the easiest model to understand and I do believe with this. It is easily understood because it provides a sequential succession of phases to be followed but then it is not that reliable. Just seeing a figure of the flow of the waterfall model you would just see the sequence of phases to go through but the problem here is it would not go through a cycle but just have a one-way flow just like a waterfall. Because of its simplicity it would only be suitable for certain classes of software development and would not work well with the other software like interactive applications. This model does not have risk management and management during the life cycle and mainly document-driven or code-driven that is why it would not work as smoothly as the other model.

2.3 Variations on Iterative Development

A number of *Process Models* have evolved from the *Iterative* approach. All of these methods produce some demonstrable software product early on in the process in order to obtain valuable feedback from system users or other members of the project team. Several of these methods are described below.

Prototyping

The *Prototyping Model* was developed on the assumption that it is often difficult to know all of your requirements at the beginning of a project. Typically, users know many of the objectives that they wish to address with a system, but they do not know all the nuances of the data, nor do
they know the details of the system features and capabilities. The *Prototyping Model* allows for these conditions, and offers a development approach that yields results without first requiring all information up-front . When using the *Prototyping Model*, the developer builds a simplified version of the proposed system and presents it to the customer for consideration as part of the development process. The customer in turn provides feedback to the developer, who goes back to refine the system requirements to incorporate the additional information. Often, the prototype code is thrown away and entirely new programs are developed once requirements are identified.

There are a few different approaches that may be followed when using the *Prototyping Model*:
· creation of the major user interfaces without any substantive coding in the background in order to give the users a "feel" for what the system will look like,
· development of an abbreviated version of the system that performs a limited subset of functions; development of a paper system (depicting proposed screens, reports, relationships etc.), or · use of an existing system or system components to demonstrate some functions that will be included in the developed system.[6]

Prototyping is comprised of the following steps:
· **Requirements Definition/Collection.** Similar to the Conceptualization phase of the *Waterfall Model*, but not as comprehensive. The information collected is usually limited to a subset of the complete system requirements.
· **Design.** Once the initial layer of requirements information is collected, or new information is gathered, it is rapidly integrated into a new or existing design so that it may be folded into the prototype.
· **Prototype Creation/Modification.** The information from the design is rapidly rolled into a prototype. This may mean the creation/modification of paper information, new coding, or modifications to existing coding.
· **Assessment.** The prototype is presented to the customer for review. Comments and suggestions are collected from the customer.
· **Prototype Refinement.** Information collected from the customer is digested and the prototype is refined. The developer revises the prototype to make it more effective and efficient.
· **System Implementation.** In most cases, the system is rewritten once requirements are understood. Sometimes, the *Iterative* process eventually produces a working system that can be the cornserstone for the fully functional system.

Problems/Challenges associated with the Prototyping Model

Criticisms of the *Prototyping Model* generally fall into the following categories:
· **Prototyping can lead to false expectations.** *Prototyping* often creates a situation where the customer mistakenly believes that the system is "finished" when in fact it is not. More specifically, when using the *Prototyping Model*, the pre-implementation versions of a system are really nothing more than one-dimensional structures. The necessary, behind the-scenes work such as database normalization, documentation, testing, and reviews for efficiency have not been done. Thus the necessary underpinnings for the system are not in place.
· **Prototyping can lead to poorly designed systems.** Because the primary goal of *Prototyping* is rapid development, the design of the system can sometimes suffer because the system is built in a series of "layers" without a global consideration of the integration of all other components. While initial software development is often built to be a "throwaway, " attempting to retroactively produce a solid system design can sometimes be problematic.

2.4 Variation of the Prototyping Model

A popular variation of the *Prototyping Model* is called Rapid Application Development (RAD).

RAD introduces strict time limits on each development phase and relies heavily on rapid application tools which allow for quick development.

Critic

Criticisms of the Prototyping Model generally fall into the following categories:

• Prototyping can lead to false expectations. Prototyping often creates a situation where the customer mistakenly believes that the system is "finished" when in fact it is not. More specifically, when using the Prototyping Model, the pre-implementation versions of a system are really nothing more than one-dimensional structures. The necessary, behindthe- scenes work such as database normalization, documentation, testing, and reviews for efficiency have not been done. Thus the necessary underpinnings for the system are not in place.

• Prototyping can lead to poorly designed systems. Because the primary goal of prototyping is rapid development, the design of the system can sometimes suffer because the system is built in a series of "layers" without a global consideration of the integration of all other components. While initial software development is often built to be a "throwaway, " attempting to retroactively produce a solid system design can sometimes be problematic.

This model cannot be used in robust application. It is convenient because it is fast from the word itself. It can replace the specification phase but not the design phase because it mainly relates to the designing phase. In the waterfall model every phase should directly right at the first time while prototyping changes frequently and the discarded if wrong.

2.5 The Exploratory Model

In some situations it is very difficult, if not impossible, to identify any of the requirements for a system at the beginning of the project. Theoretical areas such as Artificial Intelligence are candidates for using the *Exploratory Model*, because much of the research in these areas is based on guess-work, estimation, and hypothesis. In these cases, an assumption is made as to how the system might work and then rapid iterations are used to quickly incorporate suggested changes and build a usable system. A distinguishing characteristic of the *Exploratory Model* is the absence of precise specifications. Validation is based on adequacy of the end result and not on its adherence to pre-conceived requirements.

The *Exploratory Model* is extremely simple in its construction; it is composed of the following steps:

· **Initial Specification Development.** Using whatever information is immediately available, a brief System Specification is created to provide a rudimentary starting point.

· **System Construction/Modification.** A system is created and/or modified according to whatever information is available.

· **System Test.** The system is tested to see what it does, what can be learned from it, and how it may be improved.

· **System Implementation.** After many iterations of the previous two steps produce satisfactory results, the system is dubbed as "finished" and implemented.

Problems/Challenges associated with the Exploratory Model

There are numerous criticisms of the *Exploratory Model*:

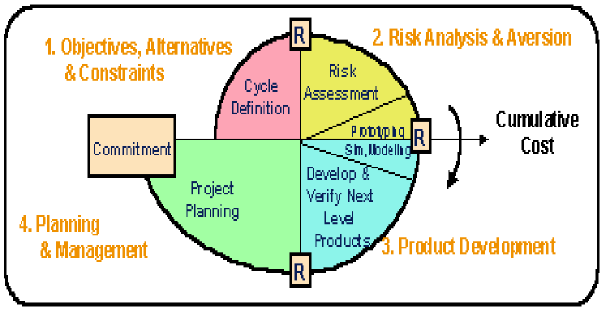
· It is limited to use with very high-level languages that allow for rapid development, such as LISP.

· It is difficult to measure or predict its cost-effectiveness.

· As with the *Prototyping Model*, the use of the *Exploratory Model* often yields inefficient or crudely designed systems, since no forethought is given as to how to produce a streamlined system.

The Spiral Model

The *Spiral Model* was designed to include the best features from the *Waterfall* and *Prototyping Models*, and introduces a new component - risk-assessment. The term "spiral" is used to describe the process that is followed as the development of the system takes place. Similar to the *Prototyping Model*, an initial version of the system is developed, and then repetitively modified based on input received from customer evaluations. Unlike the *Prototyping Model*, however, the development of each version of the system is carefully designed using the steps involved in the *Waterfall Model*. With each iteration around the spiral (beginning at the center and working outward), progressively more complete versions of the system are built.6



R=Review
Figure 4. Spiral Model[7]

Risk assessment is included as a step in the development process as a means of evaluating each version of the system to determine whether or not development should continue. If the customer decides that any identified risks are too great, the project may be halted. For example, if a substantial increase in cost or project completion time is identified during one phase of risk assessment, the customer or the developer may decide that it does not make sense to continue with the project, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

The *Spiral Model* is made up of the following steps:
· **Project Objectives.** Similar to the system conception phase of the *Waterfall Model*. Objectives are determined, possible obstacles are identified and alternative approaches are weighed.
· **Risk Assessment.** Possible alternatives are examined by the developer, and associated risks/problems are identified. Resolutions of the risks are evaluated and weighed in the consideration of project continuation. Sometimes prototyping is used to clarify needs.
· **Engineering & Production.** Detailed requirements are determined and the software piece is developed.
· **Planning and Management.** The customer is given an opportunity to analyze the results of the version created in the Engineering step and to offer feedback to the developer.

Problems/Challenges associated with the Spiral Model
Due to the relative newness of the *Spiral Model*, it is difficult to assess its strengths and weaknesses. However, the risk assessment component of the *Spiral Model* provides both developers and customers with a measuring tool that earlier *Process Model*s do not have. The measurement of risk is a feature that occurs everyday in real-life situations, but (unfortunately) not as often in the system development industry. The practical nature of this tool helps to make the *Spiral Model* a more realistic *Process Model* than some of its predecessors.

Critic
Another traditional process model is the spiral model which is suggested by Barry Boehm in 1988. Spiral model is still regarded as one of the best model because it is a combination of the prototyping model and the waterfall model and comprises the strengths of the other software models.. According to Boehm, "the major distinguishing feature of the Spiral Model is that it creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process. It incorporates many of the strengths of other models and resolves many of their difficulties" [Boehm 1988]. This model is better than the waterfall because it may allow iteration. The main concept of the spiral model is that it aims to minimize risks with the use of repeated use of prototypes so that certain changes may be applied over again if there appears a problem upon the development.

3. SWOT Analysis

3.1 Waterfall model:-

1) STRENGTH:-

- Easy adaptability by Non Technical person(End-user).
- Provides structure to inexperienced staff.
- No planning needed.
- Works well for small projects with fixed and clear requirements.
- Milestones are well defined and understood.
- Sets requirements stability.
- Good for management control (plan, staff, track).
- Works well when quality is more important than cost or schedule.
- Each phase has well defined inputs and outputs.

2) WEAKNESS:-

- All requirements must be known upfront.
- Deliverables created for each phase are considered frozen inhibits flexibility.
- Longest tangible delivery time. The customer does not see anything but the whole product when it’s ready.
- It can give a false impression of progress.
- Does not reflect problem-solving nature of software development. i.e iterations of phases.
- Integration is one big bang at the end.
- Little opportunity for customer to preview the system.
- Unsuitable for large projects and where requirements are not clear.

3) OPPORTUNITIES:-

- Requirements are very well known.
- Product definition is stable.
- Technology is understood.
- New version of an existing product.
- Porting an existing product to a new platform.
- Helpful for developing similar type of software.

4) THREATS:-

The problem with the waterfall model is that it has become hardwired into the thinking of project planners. It has become so pervasive that the requirements, design, build, and test progression is a given in most projects.
In the early days of simple, stand-alone applications, the waterfall model worked well spawning a host of voluminous methodologies, but it does not suit the

problems of the complex, risky, and integrated projects that IT has to deliver today.

IT developed stand-alone, batch applications. The complexities of integrating applications were only dreamed of by ambitious database architects. Today, hardly any development is made in isolation unless, like the NHS IT project, you give yourself the luxury of a scorched earth IT strategy. Because of its origins, the waterfall method does not address integration but ignores it until the end of the project, when we encounter the familiar task of trying to stitch together disparate applications and change schedules to the annoyance of the operations manager.

Another change in the nature of IT projects is that most of today's projects have a high proportion of reuse - implementing packages and reusing frameworks. The waterfall idea of creating a detailed set of requirements and then trying to find a package that fits is neither economic not practical. Increasingly, organisations are seeing the benefits of solution-constrained development rather than greenfield design.

The steps in waterfall model are fixed and the steps cannot change them. Model is self restricted.

If the model is not perfect, there must be some potential risks. Just as some poor descriptions and requirement changing are principal sources of project risk. In waterfall model if there is a misunderstanding in the analysis phase and that could not be found. The result could be destructive. This is almost the slowest step of development.

”The most difficult part is the communication between humans.” (Yacov, 2002).

How to manage the risks in the Waterfall model?

- It cannot be possible to avoid all the risks in the waterfall model because of the waterfall model itself. But there are still some ways to settle the problems. If team have experienced members in every job and cannot have any mistakes from the very beginning to the very end, then waterfall model is successful .
- The general method is getting prepared before the project really started. Have a essential Risk Analysis in the pre-phase can avoid the failure of every steps and rework which rise up the cost of the project.
- Making a Scheme of risk team can take a fast react in case there are some risk happened.
- Avoid the deal with the risk in surprise and make some bigger damage. Try to control every step in waterfall model.
- Do not forget to sign a contract after confirm the requirement with enduser. So that they will not ask you to add more extra functions in the software.
- Do remember that confirm there is not any mistakes and potential risks in one step. And then start your next step.
- The Project manager must take the most important point of the project. Concentrate resources on this point.
- Change the way of work from passive to active.

3.2 V-Shaped (Modified Waterfall) model:-

1) STRENGTH:-

- Emphasize planning for verification and validation of the product in early stages of product development.
- Each deliverable must be testable.
- Higher chances of success as test planning starts early in the SDLC cycle.
- Project management can track progress by milestones.
- Quickest for project where requirements are fixed and clearly defined.
- Easy to use

2) WEAKNESS:-

- Does not easily handle concurrent events.
- Does not handle iterations or phases.
- No early prototypes are available.
- Needs ample skilled resources.
- Does not easily handle dynamic changes in Requirements.
- Does not contain risk analysis activities.

3) OPPORTUNITIES:-

- Excellent choice for systems requiring high reliability.
- All requirements are known up-front.
- When it can be modified to handle changing requirements beyond analysis phase.
- Solution and technology are known.

4) THREATS:-

- The V-Shaped model is inappropriate for complex projects.

- The V-shaped model should have risk to used for large scale projects where requirements are unclearly defined and unfixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations, therefore, confidence of customer should be very high in order for choosing the V-Shaped model approach.

3.3 Evolutionary Prototype model:-

1) STRENGTH:-

- Customers can “see” the system requirements as they are being gathered.
- Gains customer’s confidence as developers and customers are in sync with each other’s expectations continuously.
- Developers learn from customers.
- Ideal for online systems where high level of human computer interaction is involved.
- A more accurate end product.
- Very flexible, as changes in requirements can be accommodated much more easily with every new review and refining.
- Unexpected requirements accommodated.
- Allows for flexible design and development.
- Steady, visible signs of progress produced.
- Interaction with the prototype stimulates awareness of additional needed functionality.
- Software built through prototyping needs minimal user training as users get trained using the prototypes on their own from the very beginning of the project.
- Integration requirements are very well understood and deployment channels are decided at a very early stage.

2) WEAKNESS:-

- Tendency to abandon structured program development for “code-and-fix” development
- Bad reputation for “quick-and-dirty” methods.
- Overall maintainability may be overlooked
- The customer may want the prototype delivered.
- Process may continue forever (scope creep).

3) OPPORTUNITIES:-

- Requirements are unstable or have to be clarified.
- As the requirements clarification stage of a waterfall model.
- Develop user interfaces.
- Short-lived demonstrations.
- New, original development.
- With the analysis and design portions of object-oriented development.

4) THREATS:-

- Prototyping often creates a situation where the customer mistakenly believes that the system is "finished" when in fact it is not. More specifically, when using the Prototyping Model, the pre-implementation versions of a system are really nothing more than one-dimensional structures. The necessary, behind-the-scenes work such as database normalization ,documentation, testing, and reviews for efficiency have not been done.
- The primary goal of Prototyping is rapid development, the design of the system can sometimes suffer because the system is built in a series of "layers" without a global consideration of the integration of all other components. While initial software development is often built to be a "throwaway, " attempting to retroactively produce a solid system design can sometimes be problematic.

3.4 Rapid Application model:-

1) STRENGTH:-

- Reduced cycle time and improved productivity with fewer people means lower costs.
- Time-box approach mitigates cost and schedule risk.
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs.
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.
- Increases reusability of components.
- High modularization achieves a more flexible and maintainable system.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.
- Business owners actively participate

2) WEAKNESS:-

- Accelerated development process must give quick responses to the user.
- Risk of never achieving closure.
- Hard to use with legacy systems.
- Requires a system that can be modularized.
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.
- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high for cheaper budgeted projects to befit.

3) OPPORTUNITIES:

- Reasonably well-known requirements.
- User involved throughout the life cycle.
- Project can be time-boxed.
- Functionality delivered in increments.
- High performance not required.
- Low technical risks.
- System can be modularized.

4) THREATS:-

- Rapid Application Development is an iterative and incremental process, there are certain risks to using RAD. It can lead to a succession of prototypes that never results in a satisfactory end product.
- The risks in RAD as opposed to "waterfall" development are related to the fact that RAD does not rely on a single requirements analysis phase.

3.5 Incremental model:-

1) STRENGTH:-

- Develop high-risk or major functions first.
- Each release delivers an operational product.
- Customer can respond to each build.
- Uses “divide and conquer” breakdown of tasks.
- Lowers initial delivery cost.
- Initial product delivery is faster.
- Customers get important functionality early.
- Risk of changing requirements is reduced.
- More flexible than waterfall.

2) WEAKNESS:-

- Requires good planning and design.
- Requires early definition of a complete and fully functional system to allow for the definition of increments.
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower.

3) OPPORTUNITIES:-

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time.
- A need to get basic functionality to the market early.
- On projects which have lengthy development schedules.
- On a project with new technology.

3.6 Spiral model:-

1) STRENGTH:-

- Provides early indication of insurmountable risks, without much cost.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first.
- The design does not have to be perfect.
- Users can be closely tied to all lifecycle steps.
- Early and frequent feedback from users.
- Cumulative costs assessed frequently.

2) WEAKNESS:-

- Time spent for evaluating risks too large for small or low-risk projects.
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive.
- The model is complex.
- Risk assessment expertise is required.
- Spiral may continue indefinitely.
- Developers must be reassigned during non-development phase activities.
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration.

3) OPPORTUNITIES:-

- When creation of a prototype is appropriate.
- When costs and risk evaluation is important.

- For medium to high-risk projects.
- Long-term project commitment unwise because of potential changes to economic priorities.
- Users are unsure of their needs.
- Requirements are complex.
- New product line.
- Significant changes are expected (research and exploration).

4) THREATS:-

- The risk of spiral model is the events that took place that makes the project not to achieve clients requirement or what the users want.

4. Conclusions

Selecting an SDLC model can be compared in many ways to the specification of user requirements, the more data gathered and examined, the higher the chances for successful completion of the project. Just as the specifications of user requirements are vital in the stages of design and computer system development, so can the knowledge and regulations which constitute the basis for SDLC model selection determine the success or failure of a given project.
A SWOT analysis is a tool to assess and to develop strategies to remain competitive. To sum up, selecting an appropriate SDLC model is a complex and a challenging task, which requires not only broad theoretical knowledge, but also consultation with experienced expert managers.

References

[1] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University; list is partially created from Software Engineering Best Practices,1997.
[2] Information on the Software Engineering Institute can be found at http://www.sei.cmu.edu.
[3] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush, "Key Practices of the Capability Maturity Model, Version 1.1," Software Engineering Institute, February 1993, p 1.
[4] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, February 1993, p 18.
[5] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University, from lecture notes: Software Engineering Best Practices, 1997.
[6] Linda Spence, University of Sutherland, “Software Engineering,” available at http://osiris.sunderland.ac.uk/rif/linda_spence/HTML/contents.html
[7] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University, from lecture notes: Software Engineering Best Practices, 1997.
[8] Frank Kand, “A Contingency Based Approach to Requirements Elicitation and Systems Development,” London School of Economics, J. Systems Software 1998; 40: pp. 3-6.
[9]Bryant, A. (2000), “Chinese Encyclopaedias and Balinese Cockfights – Lessons for Business Process Change and Knowledge Management,” In *Knowledge Engineering and Knowledge Management*,
[10]Wang, Y. (2002a), “The Real-Time Process Algebra (RTPA),” *Annals of Software Engineering 14*.

Prof. Ashish B. Sasankar had done MCA, M.Phil(Comp. Sci), M.Tech(CSE) and pursuing Phd in Software Engineering from RTM, Nagpur University(INDIA). He is having 12 years of Experience in Education field. He is currently working in GHRIIT, Nagpur(India). He had published 15 international and national papers. He is member of IEEE and CSI .

Dr. Vinay Chavan had Phd ,Msc in computer science. He is working as Professor in Computer Science Dept, S.K.Porwal College Nagpur (INDIA) .