International Journal of Advanced Education and Research
ISSN: 2455-5746, Impact Factor: RJIF 5.34
www.alleducationjournal.com
Volume 2; Issue 1; January 2017; Page No. 73-78

Different software engineering methodologies: Overview

Sushila

Research Scholar, FCI Sonipat, Haryana, India

Abstract
Different models and methods are used to develop a software system. This paper includes an introduction to all the different methodologies included in software development according to their categories. It discusses all the phases being carried out during development process. This paper gives the details for every methodology so that we can compare all of them. In this way, we will be able to find out which methodology will be best suited for development of our software system.

Keywords: sequential, cyclical, water sluice

Introduction

For developing and maintaining the software engineering projects 3 types of methodologies are used. These are: sequential, cyclical, and Water Sluice. The sequential and cyclical methodologies, informally known as the waterfall and spiral methodologies, are generic in design and have been simplified to emphasize a key aspect.

As we know that different phases are included in a software project life cycle. Methodologies discuss mainly four phases: analysis, design, implementation, and testing. Let us see in this paper how these phases are carried out in all these methodologies.

In a sequential methodology, the four phases are carried out sequentially one following the other. In a cyclical methodology, the four phases are carried out with each cycle producing an incremental contribution to the give the final system. The Water Sluice is a hybrid of these two above methodologies which borrows the steady progress of the sequential methodology and the iterative increments of the cyclical methodology and adds priority and governors to control change.

These three categories can be compared to find out the advantages and disadvantages of all of them. In the theory paper, the categories are analyzed in detail. Based on this categorization the performance characteristics of established methodologies can be analyzed.

The Boehm Waterfall methodology is an example of sequential methodology as in this methodology all the phases are carried out one after the other in a sequence. Likewise, the Boehm-spiral methodology, also analyzed later, is most quoted as a cyclical methodology but behaves more like a sequential methodology with many stages. Yet the term spiral has come to mean any cyclical methodology.

A Sequential Methodology

In a sequential methodology, informally known as the waterfall, the first phase carried out is the analysis phase, followed by the design phase, then implementation phase is conducted, and finally the testing phase is carried out. The team that does each phase may be different, and there may be a management decision point at each phase transition. Figure 1 shows how sequential methodology phases are implemented:

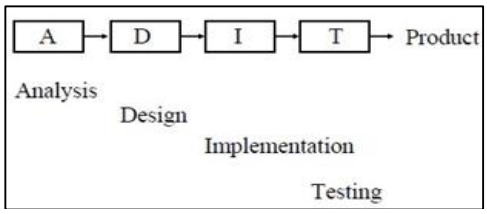

Fig 1: A Sequential Methodology

Why it works

A sequential methodology is successful when the complexity of the system is low and requirements are static. It is a simple methodology that states that first one should think about what is to be built, then make the plan for how it will be built, and then build it with quality. A discipline process is used to avoid the pressures of writing long code before it is known what is going to be built.

Sometimes it is required for the implementation team to develop some code before the completion of analysis phase just to check at the end whether that code is needed or will contribute little to the end product. Unfortunately, this early code development becomes very costly because it is difficult to abandon and difficult to change. A sequential methodology forces analysis and planning before implementation. This is good practice in many software engineering situations.

The process forces the analysis team to precisely define their requirements first as it is much easier to build something if it is known what that something really is.

The successes in the development of various historical software systems were in part due to the usage of a formal sequential methodology at the time when pressures of change coming from external sources were limited.

Why It Does Not Work

A sequential methodology might fail for many reasons. A sequential methodology requires the analysis team to be nearly clairvoyant. They must define ALL details up front. The errors and mistakes cannot be corrected after the final requirements are released.

There is no feedback about the complexity of delivering code corresponding to each one of the requirements. It may be possible that an easily stated requirement significantly

increase the complexity of the implementation, and sometimes not even possible to be implemented with today's technology.
Communication between teams can also be a big reason for failure. Traditionally, the four teams may be different and communication between them may be limited. The main channels of communication are the documents that are completed by one team and then passed to another team with little feedback. The requirement team has completed the analysis and is disbanded when the implementation team starts. The requirement documents can only capture a small fraction of the knowledge and typically do not capture any information dealing with quality, performance, behavior, or motivation.
As the technology is changing at a vast pace, it may be possible that by the time products using sequential methodology are delivered, they become obsolete. A sequential methodology puts so much emphasis on planning, that in a fast-moving target arena, it cannot respond fast enough to change. There is no early feedback from the customer and customers may change their requirements.
Frequently, once the customers see a prototype of the system, the customers change their requirements.

Waterfall Model

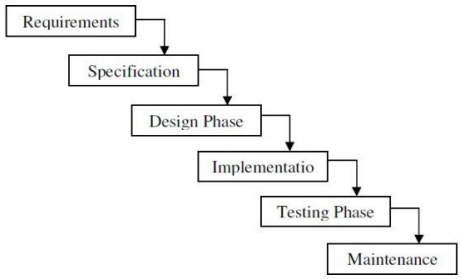

Fig 2: Waterfall model, an example of Sequential methodology

Strengths and weaknesses of Waterfall Model are

Table 1

	Strengths	Weaknesses
1	Simple and easy to use.	No working software is produced until late during the life cycle.
2	Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process	High amounts of risk and uncertainty
3	Milestones are better understood sets requirements stability hases are processed and completed one at a time.	Idealized, doesn't match reality well
4	Works well for smaller projects where requirements are very well understood.	Software is delivered late in project, delays discovery of serious errors.
5	Reinforces good habits: define before- design, design-before-code	After project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team.
6	Works well when quality is more important than cost or schedule.	It might not a good model for complex projects or projects that take more than a few months to complete

A Cyclical Methodology
Some of the problems aroused in the previous methodology are solved by cyclical methodology which is informally known as Spiral methodology. It also consists of four phases: analysis, design, implementation and testing with the difference that a little time is initially spent in each phase, followed by several iterations over all four phases.
Basically this methodology works on an idea of think a little, plan a little, implement a little, then test a little. The document structures and deliverable types from each phase incrementally change in structure and content with each cycle or iteration.
More detail is generated as the methodology progresses. Finally, after several iterations, the product is complete and ready to ship. The cyclical methodology may continue shipping multiple versions of the product. Ideally, each phase is given equal attention.

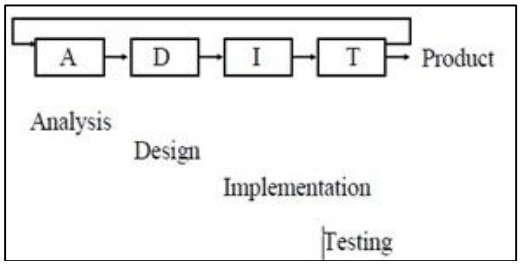

Fig 2: A Cyclical Methodology

Why It Works
A cyclical methodology is an incremental improvement on a sequential methodology. It allows for feedback from each team about the complexity of each requirement.
There are stages where mistakes in the requirements can be corrected. The customer gets a peek at the results and can feed back information especially important before final product release. The implementation team can feed performance and viability information back to the requirement team and the design team. The product can track technology better. As new advances are made, the design team can incorporate them into the architecture.

Why It Does Not Work
A cyclical methodology has no governors to control oscillations from one cycle to another cycle. Without governors, each cycle generates more work for the next cycle leading to time schedule slips, missing features, or poor quality. More often than not, the length or number of cycles may grow. There are no constraints on the requirement team to "get things right the first time." This leads to sloppy thinking from the requirement team, which gives the implementation team many tasks that eventually get thrown out.
The architecture team is never given a complete picture of the product and hence may not complete a global architecture which scales to full size. There are no firm deadlines. Cycles continue with no clear termination condition. The

implementation team may be chasing a continuously changing architecture and changing product requirements.

Spiral Model

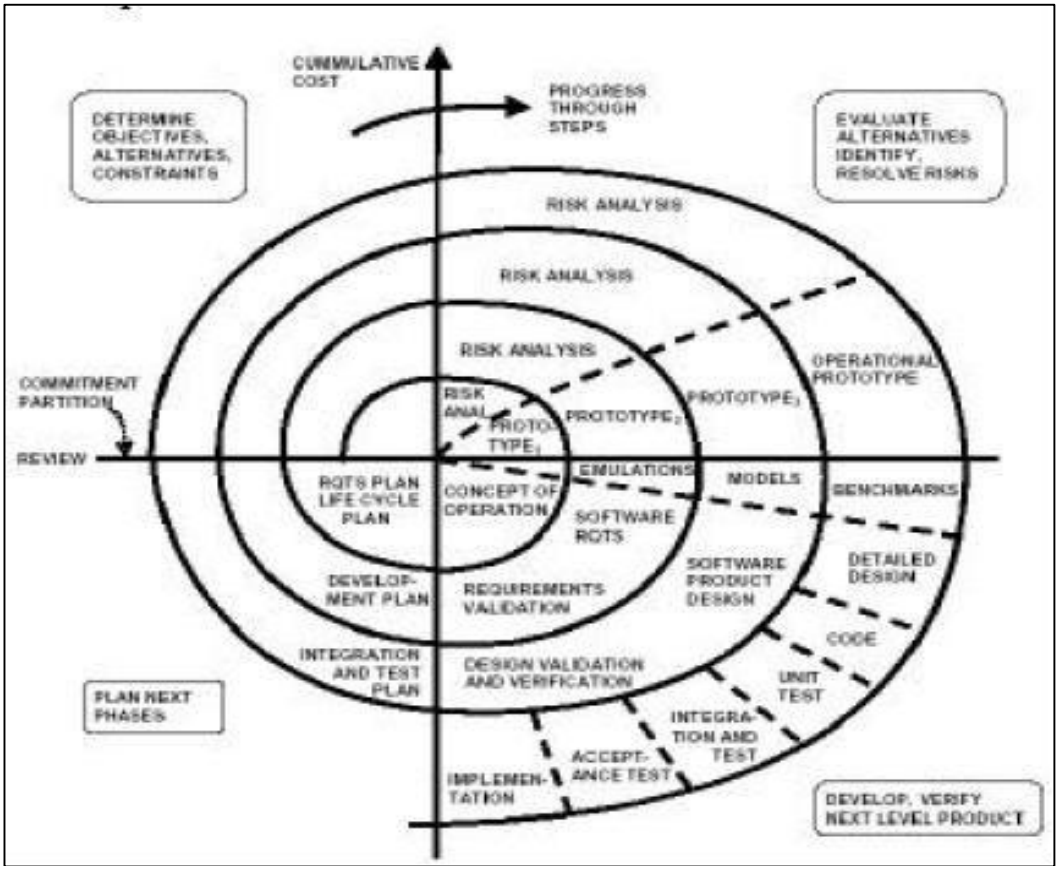

Fig 3: Spiral model, an example of cyclic methodology

The strengths and weaknesses of Spiral model are as follows

Table 2

	Strengths	Weaknesses
1	High amount of risk analysis	Can be a costly model to use.
2	Good for large and mission critical projects	Risk analysis requires highly specific expertise
3	Software is produced early in the software life cycle.	Project's success is highly dependenton the risk analysis phase
4	Provides early indication of insurmountable risks, without much cost.	Doesn't work well for smaller projects
5	Users see the system early because of rapid prototyping tools	
6	Critical high-risk functions are developed first	
7	Early and frequent feedback from users	

The Water Sluice: Introduction

A water sluice is a gold mining technique. Crushed ore and gravel are mixed with fast moving water and then channeled over a long trough with a series of perpendicular to- the-flow slats.

Each row of slats gets smaller as the water flows longer in the channel. Since gold is heavier than the surrounding rock, the gold nuggets collect at these slats. The larger nuggets collect at the bigger slats while the finer specks collect at the smaller slats. The sluice separates the valuable gold from the gravel, concentrating on the big nuggets first. The final product is a smelt of all the nuggets into one gold bar.

Similarly, the WaterSluice software engineering methodology separates the important aspects from the less important and concentrates on solving them first. As the process continues, finer and finer details are refined until the product is released. The WaterSluice borrows the iterative nature of a cyclical methodology along with the steady progression of a sequential methodology.

The Process

See Figure 4 for an overview of the WaterSluice methodology.

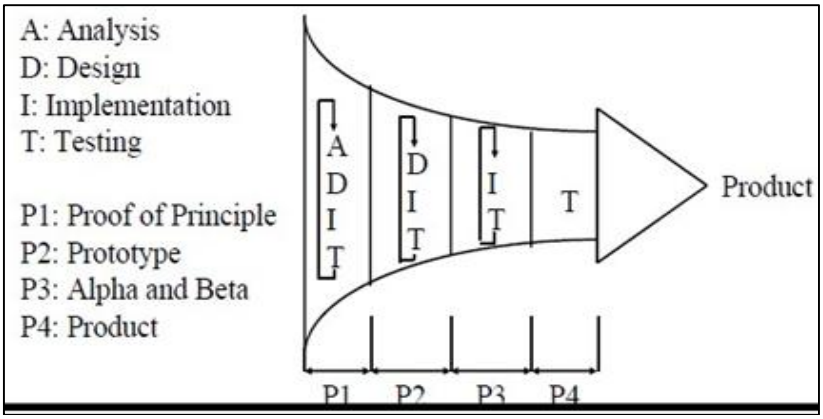

Fig 4: The WaterSluice methodology

Beginning the Process

At the beginning of the project, in an iterative process, the analysis, design, implementation, and test phases are broken into many potential tasks yet to be accomplished by team members. Each potential task is assigned a priority by team members. This priority reflects the benefit to the final goal of accomplishing the task based on what has already been accomplished. The highest priority task is accomplished next.

Depending on the size of the team, multiple high priority tasks may be accomplished in parallel. The remaining, lower priority tasks are held for later review. Exactly how many tasks or the granularity of the tasks is dependent on the size of the project, the size of the team building the project, and the scheduled delivery time for the project.

It is important that the decomposition of the problem is done well, regardless of the methodology being used for efficient performance.

Iterating the Process

As a result of accomplishing these tasks, new analysis, design, implementation, or testing tasks may be discovered. These newly discovered tasks are then added to the known remaining open tasks and again prioritization is required. The next highest priority tasks are then accomplished.

Completion of the Process

This process continues until the product is ready for release.

Priority Function

Defining the priority function is of high importance. This priority function is domain-specific as well as institution-specific, representing trade-offs between quantity and quality, between functionality and resource constraints, and between expectations and the reality of delivery. The priority function orders the different metrics and their values. However, all priority functions should have the product delivery as a high priority goal.

The priority function serves two goals. One goal is to establish priority. Important tasks need to be accomplished first over lower priority tasks. This is the traditional role of a priority function.

The second goal of the priority function is to manage conflicting and non-monotonic tasks. The priority function needs to divide the tasks into consistent collections. The priority function needs to guide the selection of the consistent collection and then followed by the selection of the tasks within that consistent selection.

As more and more of the system is established, the priority function is weighted to choose tasks that are consistent with the already established system. A non-monotonic task is inconsistent with the established base requiring that some of the already accomplished system to be thrown out. The non-monotonic task should not be taken, unless the addition of the non-monotonic task is absolutely necessary to the success of the entire system. The priority function guides this decision.

Focus on the goal

Once a component is completed to the satisfaction of the team, it is placed under change-order control. When a component is placed under the change-order control process, changes to the component are now frozen. If a change is absolutely necessary, and the teams are willing to delay the project to enforce the consequences of the change, then the change is fulfilled. Changes should be few, well justified, and documented.

Obviously, early in the process, analysis tasks are naturally a high priority. Later in the process, testing and quality become a higher priority. This is where the change order control process becomes important. At the beginning of the process all four categories of analysis, design, implementation, and testing are available for prioritizing and scheduling. At the P1-P2 transition point, see previous Figure in the process, the analysis phase is subjected to change-order control process. Having the analysis phase frozen focuses attention on the remaining three categories of tasks.

In a similar fashion, at the P2-P3 transition point, the design phase is frozen and at the P3-P4 transition point the implementation phase is frozen. At the final stage only changes that affect quality are allowed. This leads to a definition of temporal stages in the methodology, specifying priorities.

Don't confuse phases with stages. A phase is a grouping of similar activities. A stage is a temporal grouping of tasks within phases at particular times. Stages follow one another.

Stages

The main stages are called proof-of-principle, prototype, alpha and beta release, and product. With the exception of the proof-of-principle stage, these stages should not be new concepts to software engineers. The proof-of-principle stage represents the more traditional specification stage. Rapid prototyping offers a similar proof-of principle stage.

Proof-of-Principle Stage

In the first stage, the teams work simultaneously on all phases of the problem. The analysis team generates requirements. The design team discusses requirements and feeds back complexity issues to the requirement team and feeds critical implementation tasks to the implementation team. The testing team prepares and develops the testing environment based on the requirements.

The implementation team has to be focused on the critical tasks which is usually the hardest task. This contrasts the common practice of doing the simple things first and waiting until late in the product implementation to tackle the harder tasks. Most products that follow this practice end up failing. Once the critical task components have been implemented, the system, still a child in the first period of life, is ready for transition to the prototype stage.

Prototype Stage

In the second stage, the prototype stage, the requirements and the requirement document are frozen and placed under change-order control. Changes in requirements are still allowed but should be very rare. Any new requirements after this point are very costly. Only if the requirement change is absolutely necessary to the success of the product, despite the potential delays in the product delivery or cost over-runs, is the requirement change allowed. The main idea is to force control on any new requirements. This forces the cycle to be completed and enables product delivery. The architecture is still allowed to vary a little as technology pressures deliver new options.

Once the critical tasks are done well, the implementations associated with the critical tasks are expanded to cover more and more of the application. One of the goals of this stage is for the team to convince non-team members that the solution can be accomplished. At the end of this stage, the process is ready for transition into the alpha and beta release stages.

Alpha and Beta Release Stages

The first version in field release is usually called an alpha release, while a second release is called the beta. The product may be immature in the alpha release. Only critical tasks have been implemented with high quality. Usually, only a limited number of customers are willing to accept an alpha version of the product and assume the associated risk.

During the beta release, enough of the system should be working to convince the customer that soon the beta application will be a real product. The beta release is more mature and is given to a much larger customer base. When enough of the system is built, the system is ready for a transition into the next stage: releasing a high quality product.

Product

In the fourth stage, the implementation is frozen and focus is primarily on quality. At the end of the stage, the product is delivered.

One of the goals of the last stage is to make the product sound and of high quality. No known critical errors are allowed in the final product. Sometimes, there is a gray area of definition between a product feature and a product error with the provider of the product, most often then not, providing features, while the customers viewing some features as errors. The process is then repeated for the next version of the product.

The Water Sluice allows for phase interactions while at the same time setting firm temporal deadlines. The Water Sluice forces all four phases to communicate up front and to work together. The Water Sluice software engineering methodology assumes the presence of five levels in a supporting software engineering environment.

Versioning is used to move the product from one version to another version by repeating the methodology for each version. Risk management is assumed throughout the process. The major components of analysis, the details in the design phase, the four main phases of implementation, and levels of testing proceed as previously described.

Change-Order Control

Change-order control is a software engineering process that manages change, or lack thereof. The process is weighted to prevent change. Tools help to manage this process, while senior decision makers accept or decline change decisions. Frequently, the senior decision makers are independent of the teams. Once a component is completed to the satisfaction of the team, it is placed under change-order control. When a component is placed under the change-order control process, changes to the component are now frozen. If a change is absolutely necessary, and the senior decision makers are willing to delay the project to enforce the consequences of the change, then the change is fulfilled. Changes should be few, well justified, and documented.

Many change requests are postponed and incorporated into the next version of the product. Some of these change requests contribute to the requirement document for the next version, while some contribute to the architecture and implementation. Still, others may improve the quality.

Why It Works

There are many things that work well in the Water Sluice methodology. The Water Sluice methodology recognizes that people make mistakes and no decision can be absolute. The teams are not locked into a requirement or an architecture decision that turns out to be wrong or no longer appropriate. The methodology forces explicit freeze dates. This allows for the product to be built and shipped. It forces accountability by having decision points where, for the most part, things need to be completed. The first stage is iterative allowing for the correction of mistakes. Even after a portion of the system goes under change-order control, a decision can be changed if it is absolutely necessary.

The Water Sluice methodology forces the teams to think but does not require the teams to be clairvoyant. Sufficient time is allowed for the first stage to establish the confidence level needed for success. Communication is emphasized.

The Water Sluice methodology allows for fast interaction, up front, between all phases of analysis, design, implementation, and testing. This feeds critical information between all four phases. The implementation team doesn't waste time working on throw-away code because requirements are validated early in the process for feasibility of implementation. The Water Sluice methodology can respond to market changes more quickly due to the iterative nature in each stage allowing requirements to enter and exit at each stage. The Water Sluice methodology tries to move all mistakes to the beginning of the process, where a restart is not very costly.

Why It Does Not Work

The Water Sluice methodology forces accountability by having clearly defined stages where activities are frozen and placed under change order control. Many people are not willing to take that responsibility. For the Water Sluice methodology to work, it is necessary to create an environment where taking responsibility and accountability for a decision need not be detrimental to the individual if the decision later leads to a failure. Otherwise, people will avoid accepting accountability, leading to missed goals.

An attitude change towards testing is necessary by the teams since all teams are involved in testing from the beginning. The Water Sluice methodology requires that people communicate well up front, which is difficult since all four phases represent different perspectives. The methodology trades off total edibility with the reality of product delivery.

Conclusion

Now, we have a clear idea of what all the software engineering methodologies are so, if we make a comparison, we will find out that the Water Sluice methodology developed by Dr. Ron Bur back, Presents a different scope to the team members who are in the development of a project, a better goal and control management. It is the best methodology to put into use as it has all the features of a sequential and cyclical methodology.

Water Sluice methodology is the most flexible methodology, which can be used to accomplish projects within the given timelines without wasting much effort from the team members. That is to say that, in Water Sluice methodology, when a code is developed and it cannot promptly comply with the needs of the user, it need not be discarded but can be re-consider the change in requirement of phases into the iterative process. This procedure saves a lot of effort and resources leading to the focused goal.

Therefore, it can be summed up that Water Sluice methodology is one of the best available methodologies that can be put into practice to achieve more reliable results from a software solution company. This is very important in the present day scenario where labor and time are both very precious.

References

1. Schwalbe K. Information Technology Project Management" 6th Edition, Cengage Learning, 2009.
2. Bhattacharjee S. Software Development Life Cycle (SDLC), MOF, College of Agricultural Banking, RBI, Pune, 2009.
3. Nabil Mohammed Ali Munassar 1 and A. Govardhan A Comparison Between Five Models Of Software Engineering International Journal of Computer Science Issues. 2010; 7.
4. James E. Purcell Comparison of Software Development Lifecycle Methodologies.
5. Softeality. About us.[http://softeality.com/about_us] (Accessed on 11-16-2007)
6. Boehm B. Software engineering. In Proceedings of the 4th International Conference on Software Engineering
7. Boehm B. Software architectures critical success factors and cost drivers. In Proceedings of the 16th International Conference on Software Engineering.
8. Boehm B, Ross R. Theory-w software project management: a case study. In Proceedings of the 10th International Conference on Software Engineering.