StarContests.com

If it's not here it's not happening.

Software Development Life Cycle model v1.1

Team 8

Instructor - Prof. Asim Banerjee

Group members

Serial No.	Name	ID
1.	HARDIK BELADIYA	201201064
2.	ARCHIT GAJJAR	201201066
3.	SOHAM DARJI	201201070
4.	KRUPAL BAROT	201201074
5.	DHAVAL CHAUDHARY	201201075
6.	PRACHI KOTHARI	201201077
7.	YASH KUMAR JAIN	201201080
8.	RACHIT MISHRA	201201092
9.	SHIVANI THAKKER	201201108

Document Description

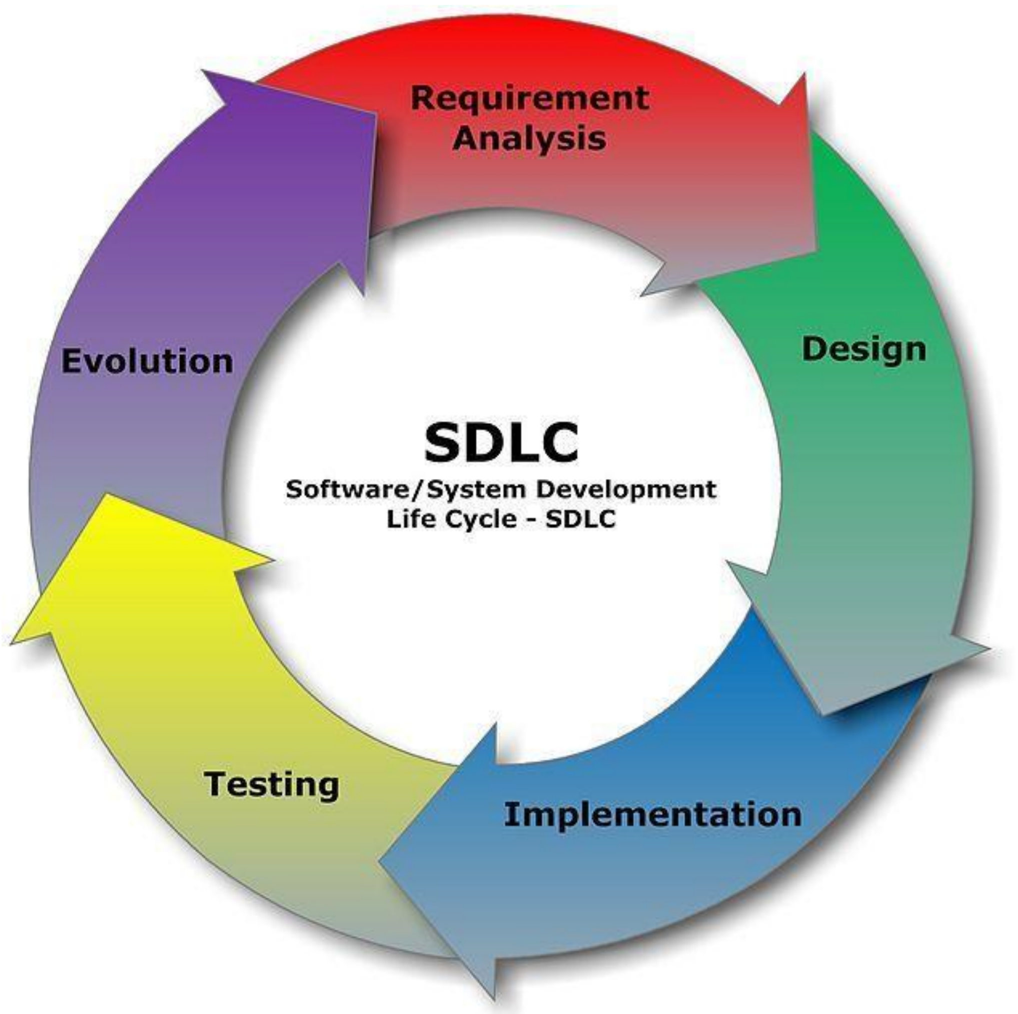
This document will define and illustrate all the software development lifecycle models for building the project and their merits and demerits. It will also justify the reasons for selecting and rejecting various models as discussed and decided by the team members.

Revision History

Date	Document	Version	Created By	Reviewed By
22nd February 2015	Software Development Life cycle model	1.0	All members	Prachi, Shivani

Table of Contents

INTRODUCTION



SDLC, Software Development Life Cycle, also called as Software Development Process, is a process used by software industry to design, develop and test high quality software. The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. The SDLC aims to produce high quality systems that meet or exceed customer expectations, based on customer requirements, by delivering systems which move through each clearly defined phase, within scheduled time-frames and cost estimates.

A typical Software Development life cycle consists of the following stages:

1. Stage 1: Planning and Requirement Analysis

2. Stage 2: Defining Requirements

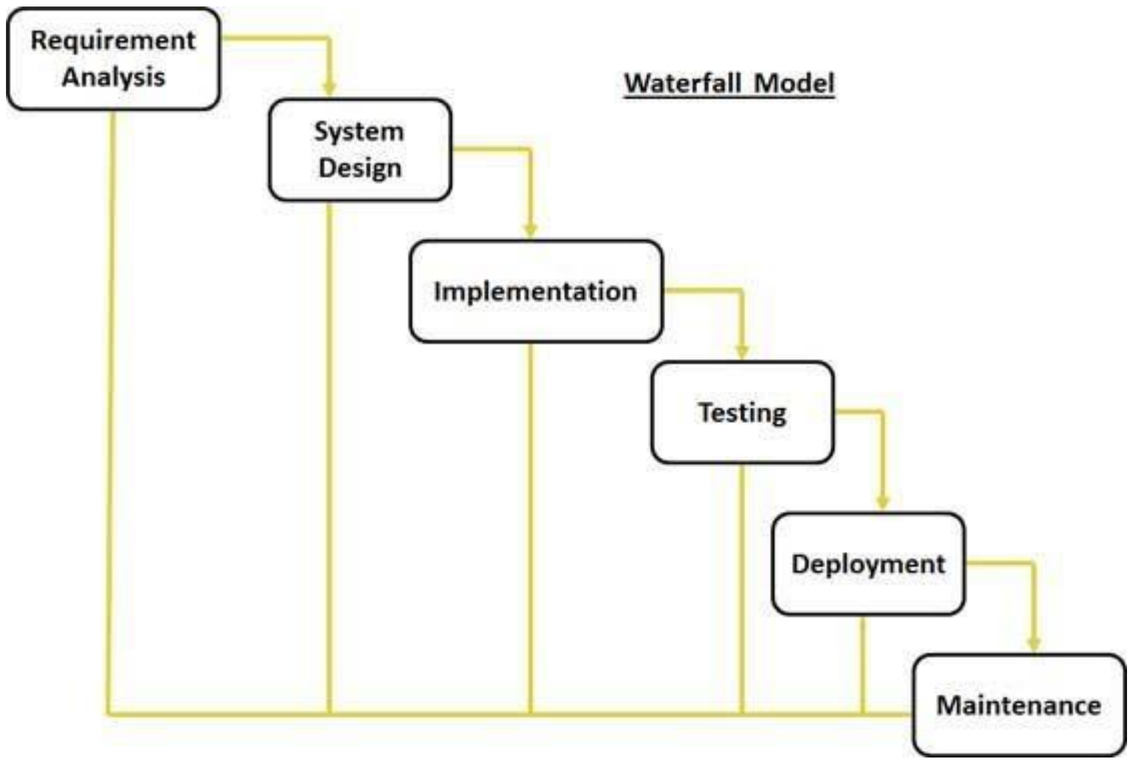
3. Stage 3: Designing the product architecture

4. Stage 4: Building or Developing the Product

5. Stage 5: Testing the Product

6. Stage 6: Deployment in the Market and Maintenance

WATERFALL MODEL



The sequential phases in Waterfall model are:

1. **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

2. **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

3. **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

4. **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

5. **Deployment of system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

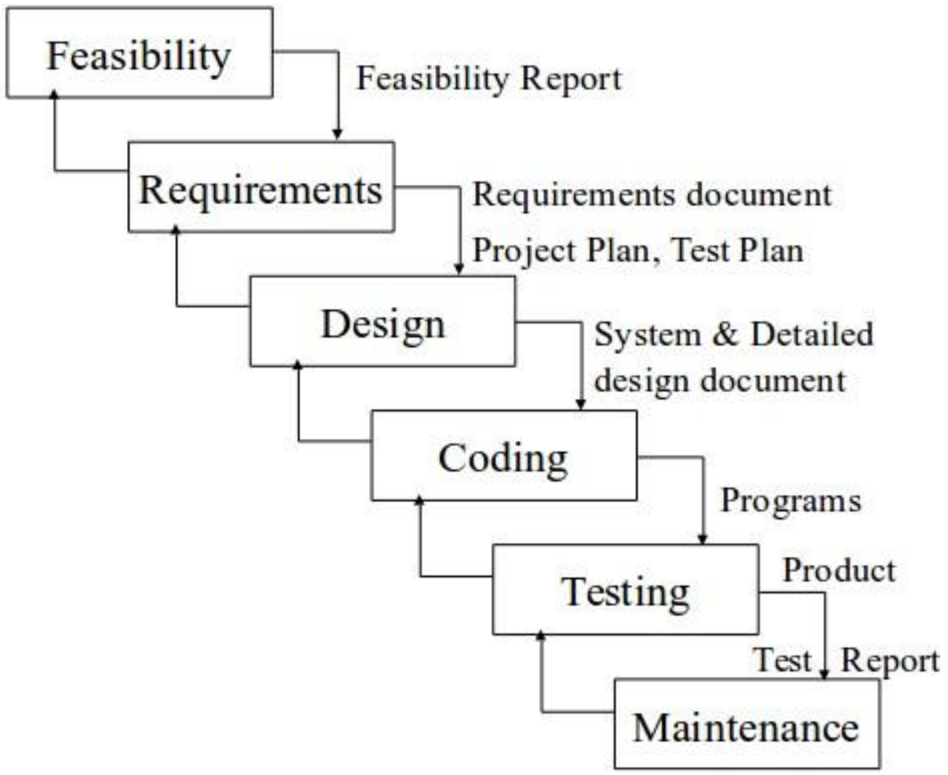
6. **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

7. All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

Justification: Waterfall model is simple easy to understand and implement
. The main requirement of this model is requirements should be well understood in the very beginning. It can be used for enhancing a project or taking it on another platform, but in project like ours requirements cannot be crystal clear at the very 1st stage of development and are very likely to change during any time in the project so it includes lots of risk to implement this model. Plus no software is produced until late in life cycle which again has high risk. We all discarded the waterfall model because it has high risk of failure.

ITERATIVE WATERFALL MODEL



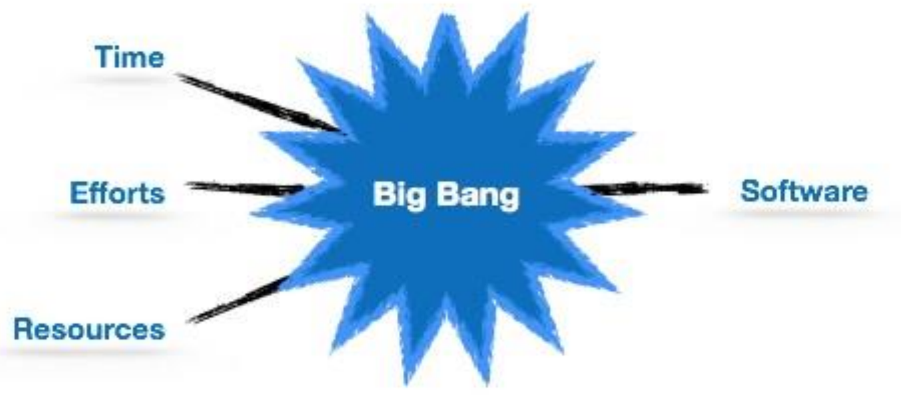
This model follows the principle of "Phase containment of errors": If and when errors occur, they should be detected (and corrected) as early as possible. This extensively reduces rework as the defects are corrected at earliest whereas in other models, the errors may be carried forward which increase the rework to be done on them for correction. In real life, defects are likely to be introduced during each phase of development. It is desirable to minimize (if not eliminate) human errors during software development. These defects get detected (later) as the development progresses. Detected defects are then corrected by going back to the appropriate phase (where the defect got introduced). This process can become time consuming and costly. Hence it is desirable to detect the defects as early as possible, preferably in the same phase itself i.e. before moving on to the next phase.

Justification:
We are choosing this SDLC model for our software development as it is cost and time effective and also, considering our lack of experience in software engineering, we have to keep check for errors regularly and continuously checking that we are on the correct path. The error would not be carried forward in this way and lead to a better development

1. Delivers work fast, the first potentially shippable features are available after only a very small amount of time.
2. People keep focused on the goal for the iteration; there is no chance to get distracted.
3. Very tight learning feedback loop allows for quick discovery of optimal solutions. 4. Useful for building small to medium sized systems and to build systems of high reliability.
5. Useful for testing the system within development phase.
6. Every phase work is verified and tested before moving on to next phase and if some error detected in next phase we can loop back to the previous phase to correct them.

Big Bang Model

The big bang model is SDLC model where there is no specific process followed. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement.



Merits of Big Bang Model:

1. It is very simple model and it is very easy to manage it.
2. Little or no planning is required.
3. Very less resources are required.
4. Gives flexibility to developers and is a good learning aid for newcomers or students.

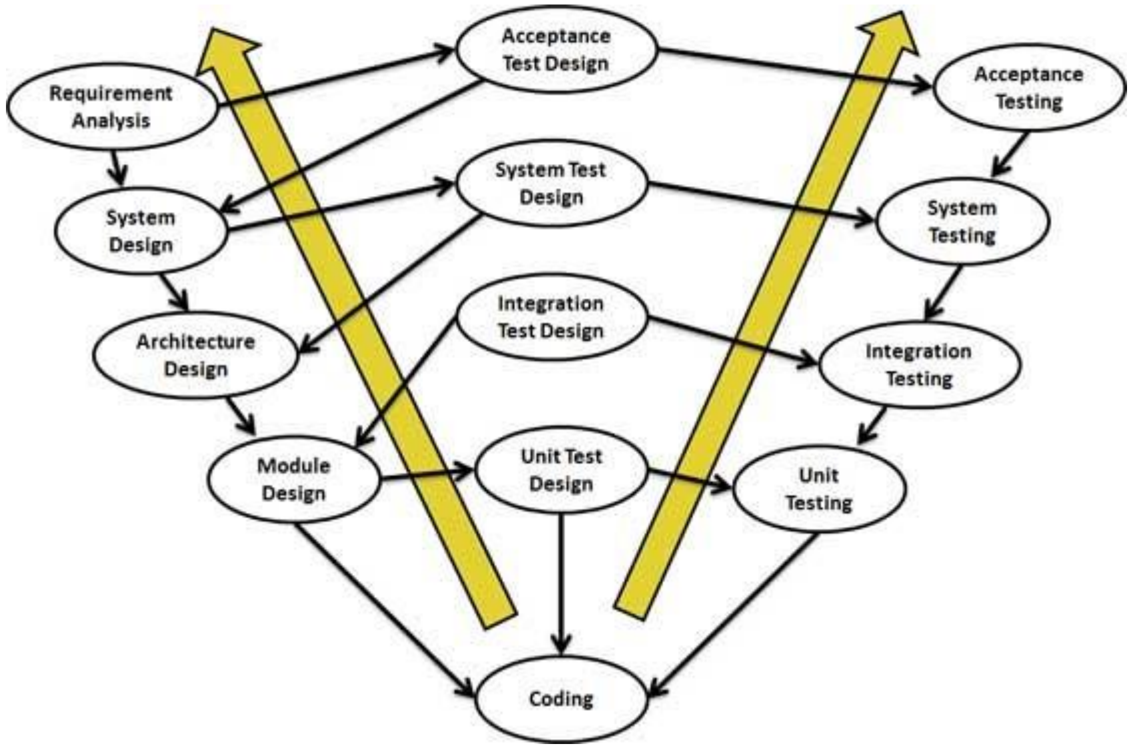
Justification for not selecting the model:

1. We are working on this project with a lot of planning but as this model doesn’t require much planning it is not relevant for our project.
2. Very high risk and uncertainty are there in this model.
3. It is not a good model for complex and object-oriented projects.
4. It is poor model for long and ongoing projects.
5. It can turn out to be very expensive if requirements are misunderstood.
6. In this model even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.
7. This is one of the SDLC methodologies typically used for small projects with only one or two software engineers.

V MODEL

The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.



Merits of V Model:

1. This is a highly disciplined model and Phases are completed one at a time.
2. Works well for smaller projects where requirements are very well understood.
3. Simple and easy to understand and use.
4. Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

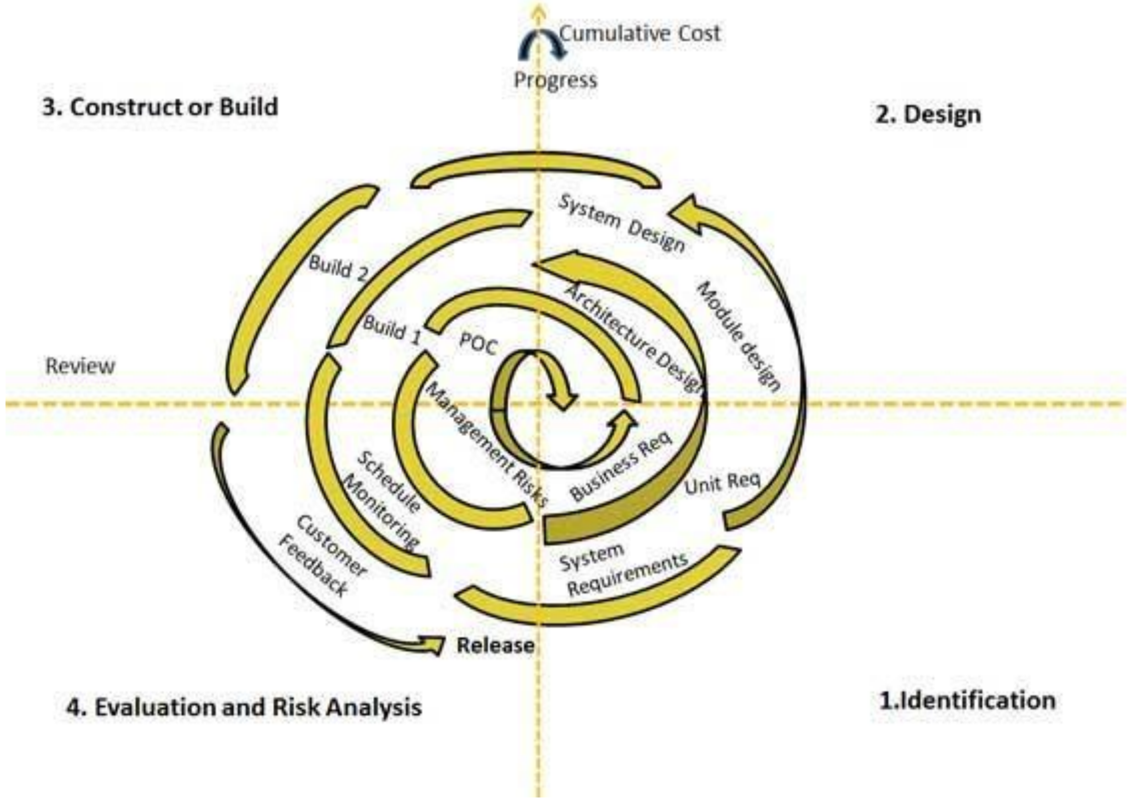
Justification for not selecting the model:

1. This model is not suitable for the projects where requirements are at a moderate to high risk of changing and we are doing project in which requirements can be change according to client and users of the project.
2. This model is not good for object oriented and complex project and its poor model for ongoing and long project so it's not suitable model in our case.
3. In this model once we enter in testing stage it's difficult to go back and change the functionality and we don't want that to happen.
4. Very high risk and uncertainty are there in this model.
5. No working software is produced until late during the life cycle.

SPIRAL MODEL

Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.



Merits of Spiral Model:

1. Changing requirements can be accommodated.
2. Allows for extensive use of prototypes 3. Requirements can be captured more accurately.
4. Users see the system early.
5. Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

Justification for not selecting this model:

1. Management is more complex.
2. End of project may not be known early.
3. Not suitable for small or low risk projects and could be expensive for small projects.

4. Process is complex.
5. Spiral may go indefinitely.
6. Large number of intermediate stages requires excessive documentation.

RAD (Rapid Action Development) MODEL

RAD is an incremental model. Multiple tasks are performed parallel. In RAD model the components or functions are developed in parallel as if they are mini projects and sub tasks are assigned. Developments are then assembled into a working prototype which gives the customer what they need. Business modelling, data modelling, process modelling are various phases in Rapid Action Development models.

Merits of RAD:

1. Reduced Development time.
2. Increased reusability of components.
3. Integration solves a lot of issues.

Justification for not selecting this model:

We decided to reject this model because it was almost inapplicable to cheaper projects as cost of modelling and automated code generation is very high. Also, this project depends highly on modelling skills and required highly skilled developers and designers.

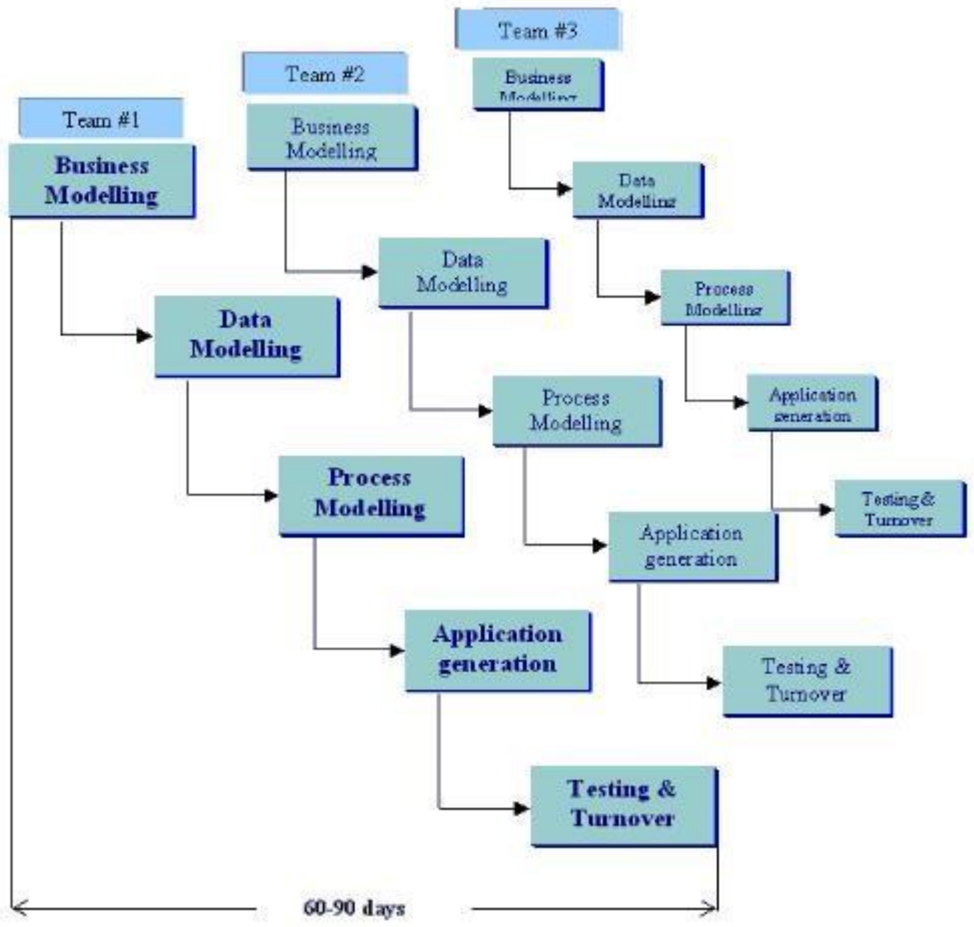


Figure 1.5 – RAD Model

Scrum Model

This model focuses on process adaptability and customer satisfaction by rapid delivery of working software product. Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly affected or removed based on feedback. This effectively gives the customer the finished system they want or need. Both system developers and stakeholders alike, find more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available, meaning the project can continue to move forward without fear of reaching a sudden standstill.

Merits:

1. In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

2. Project Managers find that planning and tracking are easier and more concrete compared to waterfall processes.

3. Due to short sprints and constant feedback, it becomes easier to cope with the changes.

4.Fast moving, cutting edge developments can be quickly coded and tested using this method, as a mistake can be easily rectified.

Justification for not selecting this model:

1. This methodology needs experienced team members only.

2. If a task is not well defined, estimated project cost and time will not be accurate. In such a case, the task can be spread over several sprints.

3. Customer should be approached regularly which is not feasible in our case.\

Software Prototyping

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software. Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

1. Prototype is a working model of software with some limited functionality.
2. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

3. Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is the stepwise approach to design a software prototype:

1. Basic Requirement Identification: This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

2. Developing the initial Prototype: The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

3. Review of the Prototype: The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

4. Revise and enhance the Prototype: The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like, time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

These are different types of SDLC models used in the industry, we have decided to use **iterative waterfall model of development**

Iterative Waterfall model of development:

As we have justified above on page 8, the iterative waterfall model allows us to go back to the previous phase to correct the errors if detected and also, keep check on the process going on in the correct path. The errors are detected at earliest and corrected which reduces rework and development forwards in cost and time effective way.

So, after discussing all the models in detail in the team, we decided on working with the iterative waterfall model in the project.