Unit- 1 :Software Engineering

Definition –

Software engineering is the process of analyzing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development. In contrast to simple programming, software engineering is used for larger and more complex software systems, which are used as critical systems for businesses and organizations.

Characteristics of Software

The quality of a software product is determined by what it offers and how easily it can be used. Software is judged by different people on different grounds. Customers, for instance, want software that meets their specific needs. Similarly, developers engaged in designing, coding, and maintaining the software determine the quality of the software by assessing its internal characteristics. Let's check them out…

1. Functionality

The functionality of software refers to its ability to perform and function according to design specifications. In simple terms, software systems should function correctly, i.e. perform all the functions for which they are designed.

The functions refer to the features that the end user, as well as the business, expect as basic facilities from the system. All these functions must be integrated into the system. Many software applications out there are designed for simplicity, but ultimately, the purpose of the software is to provide its users with the desired functionality. In order to look like the best software product, it must have a clear appearance, components, and functions. However, there are also those products out there that can provide a great deal of value for your money.

2. Usability (User-friendly)

The user-friendliness of the software is characterized by its ease of use. In other words, learning how to use the software should require less effort or time. Navigating the software is extremely important since it helps determine the journey the user takes within the software. This is imperative to ensure visitors remain on your website and have a positive experience, which leads to an increase in sales and brand loyalty.

3. Efficiency

Essentially, it refers to the software's ability to utilize human and system resources such as time, effort, CPU, memory, computation power, network bandwidth, files, databases, etc., as effectively and efficiently as possible. For a software project to succeed, efficiency is crucial. In addition to meeting the needs for which the software was made, it must also provide excellent features designed to assist users in completing their tasks faster. Software should make efficient use of storage space and execute commands according to timing requirements.

4. Flexibility

Software Flexibility refers to the ability of the software solution to adapt to potential or future changes in its requirements. When evaluating the flexibility of software, look at how simple it is to add, modify, or remove features without interfering with the current operation.

5. Reliability

The reliability of a software product describes the likelihood it will operate without failure over a specified period of time under certain conditions. It determines the ability of software to maintain its level of performance (provide desired functionality) under specified conditions for a specified period of time. Generally speaking, software reliability is measured as the availability of the software. The value should not be less than 99%. In reliability testing, the goal is not perfection, but achieving a level of reliability that is acceptable before a software product is released to customers.

Software Crisis

1.Hardware Oriented World: There was more focus on the hardware and software as people were more focusing on that how instructions can be executed fast. Not on how bugs should be removed frequently.

2. Cost: Hardware vs Software
Rapid increase in software cost and Rapid fall in hardware cost need more resources and well defined and systematic model for software development.
3. Inexperienced Developer:
Developer were hardly one year of experienced old so quite difficult for them to fulfill all coding based requirement.

What is SDLC?

The Software Development Life Cycle (SDLC) is a well-structured process that guides software development projects from start to finish. It provides a clear framework for planning, building, and maintaining software, ensuring that development is systematic and meets quality standards.

Defining the specific SDLC stages ensures that development is organized and executed effectively, resulting in high-quality software that meets user requirements. Following a structured approach, development teams can reduce risks, optimize resources, and produce software that aligns with business goals– all within a reasonable timeframe.

The 7 phases of the software development life cycle

The SDLC process typically consists of several key phases, each contributing to the successful development of software. The main SDLC phases include planning, implementation, testing, and deployment, but that's not all.

Each phase plays a crucial role in effectively designing the software, meeting user needs, and ensuring timely delivery.

Phase 1: Planning

The planning phase is the foundation of any successful software development project. Project goals, objectives, and requirements are gathered and documented during this phase. Project requirements can be based on customer feedback or market research evaluating existing product options. Stakeholders work together to define the project scope, establish timelines, and allocate resources. Planning establishes the project's direction, ensuring that all participants have a clear understanding of what needs to be done and how to achieve it.

Phase 2: Feasibility analysis

Once planning is complete, the feasibility analysis phase begins. During this phase, the project team evaluates whether the project is technically and financially viable. This includes assessing the technical requirements, estimating costs, and performing a risk analysis. Risk assessment is essential to identifying potential challenges and determining if the project is worth pursuing.

Phase 3: System design

The system design phase includes creating the software's architecture and design. Based on the requirements gathered during planning, the team creates a blueprint outlining how the software will function. This includes high-level architecture and detailed design specifications, including user interface design to ensure the software is user-friendly and an assessment of requirements for compatibility with existing products.

Phase 4: Implementation

The implementation phase, also known as the development phase, transforms the design into a functional application. It is here that the actual coding takes place. Developers write the code based on the design specifications, following best practices and coding standards to ensure the result is efficient, secure, and maintainable.

Phase 5: Testing

The testing phase is critical because it generates essential performance and usability feedback while revealing defects and quirks. Various types of software testing can be used, including automated testing, unit testing, integration testing, and system testing. The goal is to identify and fix bugs, ensuring the software operates as intended before being deployed to users.

Phase 6: Deployment

Once internal software testing is complete, the solution can be deployed to end users. This typically includes a beta-testing phase or pilot launch, limited to a select group of real-world users. Depending on the project's needs, deployment can be done on-premise or in the cloud. The deployment strategy determines how easily users can access and use the software.

Phase 7: Maintenance

The last phase of the SDLC is maintenance. Even after the software is deployed, ongoing support is necessary to address issues, apply updates, and add new features. Continuous maintenance ensures that the software remains functional and relevant over time.

What are SDLC models?

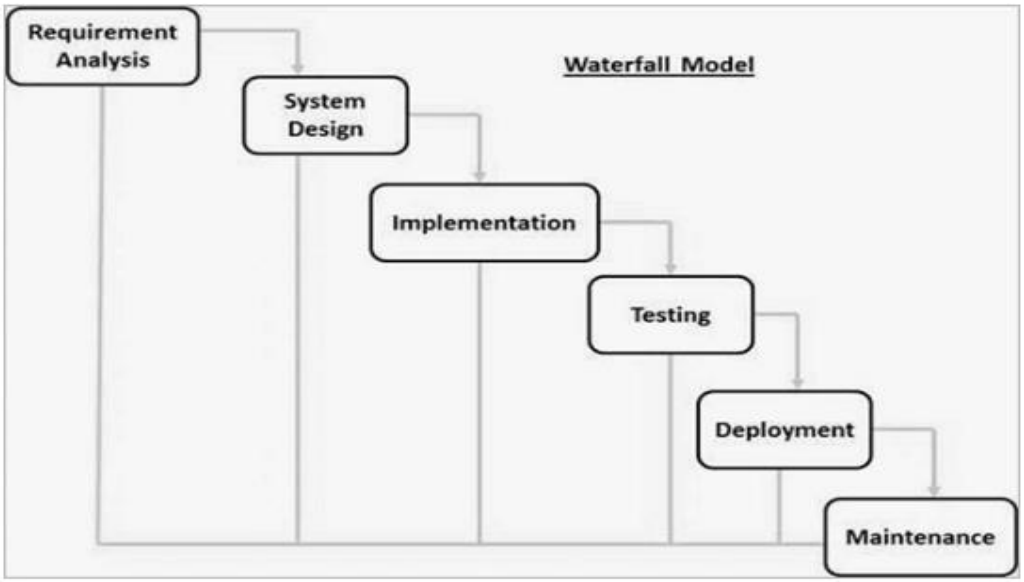
A software development lifecycle (SDLC) model conceptually presents SDLC in an organized fashion to help organizations implement it. Different models arrange the SDLC phases in varying chronological order to optimize the development cycle. We look at some popular SDLC models below.

Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.



The sequential phases in Waterfall model are −

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

- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation** − With inputs from the 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.

- **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.

- **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.

- **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.

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.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow 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.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Print Page

Prototype Model

The Prototype Model is a software development methodology used in the field of software engineering. It is an iterative and incremental approach that focuses on building a preliminary version of a software system, known as a "prototype," to gather user feedback and refine the requirements and design of the final system. The Prototype Model is often employed when the requirements of a project are not well-understood or are subject to change, and it is particularly useful for projects with a high degree of user interaction and user interface design.

Steps of Prototype Model

The development of the prototype Model takes place in various stages.

Requirement, Gathering and Analysis

This stage involves interacting with the community for which the product is being designed and understanding their needs to know their expectations and work in the right direction.

Quick Designing

This stage involves creating a bird-view design of the model, which satisfies the requirements gathered in the first step. It could be a more technical and complete design of the model. It is just like a rough draft.

Build a Prototype

At this stage, an actual model is built using the raw model built in step 2, and all the information is gathered in step 1. It includes a more detailed specification of the various components.

User Evaluation or Review

Since prototype model building is an iterative process, a user review is necessary after every temporary model construction. The concerned user reviews the current prototype model and returns feedback. These are collected and given as input to the developer for the next iteration.

Prototype Refinement

Next, the developer analyzes the suggestions and reviews collected from the user, and the appropriate changes are made to the existing model. This process continues until the user is convinced that the model fulfills his requirements.

Implement and Maintain Product

The model is constructed and implemented according to user specifications. The constructed model is the basis for implementation. Regular maintenance is performed on the product or the system to minimize downtime and prevent significant failures.

Types of Prototyping Models

The following are the diverse categories of the prototype model. They may be classified on various parameters, including implementation, cost-effectiveness, and user involvement.

- Rapid Throwaway Prototyping

- Evolutionary Prototyping

Rapid Throwaway Prototyping
In this kind of prototyping, the model is developed quickly and discarded or 'thrown away' as soon as the product or design is built. It helps to step forward from the prototyping stage quickly.
This kind of prototyping is done for getting an idea of how the product will look. Customer feedback is taken into account for promoting any demand change. The re-creation of the prototype takes place until the need is basic.

Evolutionary Prototyping
In this type, first of all, an initial prototype model is built. After that, the customer needs and requirements are analyzed and integrated with the existing model. The review and changes are made until the customer is fully satisfied.
Evolutionary prototyping is generally used in high-end projects in which the testing of every feature is done and when any unknown replacement technology is used in a project. Through this prototyping, we can avoid the wastage of both effort and time.

Advantages of Prototype Model
The advantages of the prototype model are as follows.

- The customers/users are fully satisfied with the product as they can review the model at every stage, give suggestions and ensure it meets their demands.
- It minimizes the risk of product failure as these factors have already been studied intensely in the prototype development process.
- It gives the customer space to put forth his ideas as they shape his mind. It only requires some of the specifications at the very start.
- It only requires a few skilled developers to develop the product or system.
- Flexible designing.
- Saves product construction cost and time.

Disadvantages of Prototype Model
The disadvantages of the prototype model are as follows.

- It is a slow and time-consuming process.
- It requires customer interaction and interference at every stage.
- Customers may change their product developer if they are satisfied with the initial designs. They can cause massive losses to the developers.

- The money invested in prototype model development could be better utilized. This is because the prototype is useless after final product development.

- It promotes unnecessary changes in the constructed model.

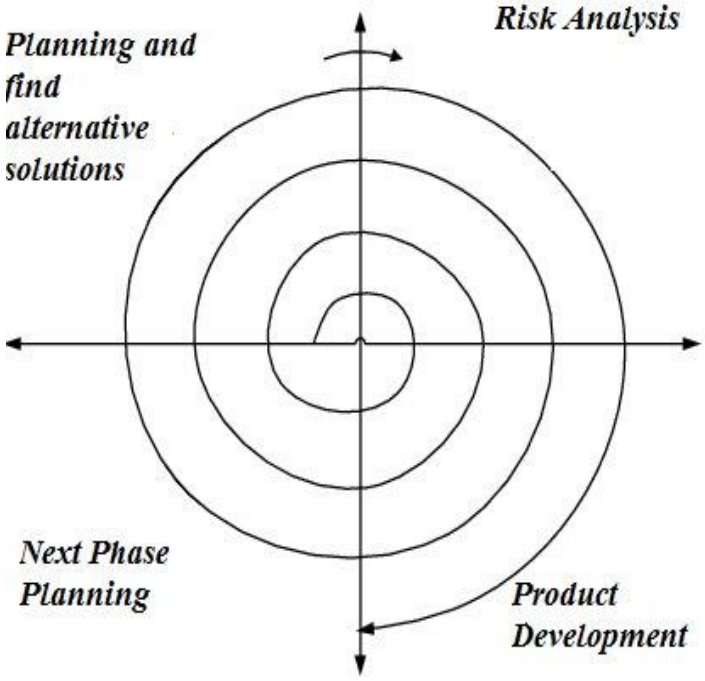
Spiral Model

The spiral model is a systems development lifecycle (SDLC) method used for risk management that combines the iterative development process model with elements of the Waterfall model. The spiral model is used by software engineers and is favored for large, expensive and complicated projects.

Risk Handling in Spiral Model

The spiral model analyzes all proposed solutions and identifies, analyzes and addresses all potential risks. Following this, methods such as prototyping, simulation, benchmark testing, analytical models, and user research are used to develop the lowest-risk, most cost-effective strategy.

Spiral Model Phases



The different phases of the spiral model are-

1. Planning

This phase begins by gathering business requirements into a baseline spiral. In the subsequent spiral, all system, subsystem and unit requirements are identified at this stage as the product matures.

This phase also includes understanding system requirements through ongoing customer and system analyst communication. The product will be deployed in the identified market at the end of the spiral. This includes iteration cost, schedule, and resource estimates. This

includes understanding system requirements for ongoing communication between system analysts and customers.

2. Risk Analysis

After the “plan” phase, the team prepares for the “risk” phase. The “risk” phase is designed to consider the variability in the rate at which a given product might fail. It is designed to account for the uncertainty in the rate at which a given product might fail. During the “risk” phase, the team evaluates various aspects of the current state of the product, such as the state of its code, the state of its design, and the state of its prototype. The team then makes adjustments to the current state of the product based on the changes made in the “plan” phase and then follows up with a “sales” phase to collect customer feedback.

Once risks are identified, risk mitigation strategies are planned and completed.

Briefly, risk analysis involves identifying, estimating and monitoring technical feasibility and management risks such as schedule slippage and cost overrun. After testing the build, customers rate the software at the end of the first iteration and provide feedback.

3. Product development

In the next quadrant, prototypes are built and tested. This step includes architectural design, module design, physical product design and final design. Convert the proposals made in the first two quadrants into usable software.

This phase also includes the actual implementation of features in a project, which are verified by performing testing.

4. Next phase planning

In this phase, the customer evaluates the software and gives feedback. The team prepares for the next phase of the planning process. The next phase of the planning process is known as the “spiral” phase. During the “spiral” phase, the team determines the order of events in the current state of the product and then follows these events up with a “revision” phase to “Revise” the current state of the product so that it is ready for production. The “revision” phase is also called the “reproduction” phase, one of the most important aspects of the planning process.

Benefits of the Spiral Model

The benefits of the spiral model include:

- The spiral model works for development as well as enhancement projects.
- The spiral model is a software development model designed to control risk. 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.
- Spiral models are ideal for large and complex projects, as continuous prototyping and evaluation help reduce risk.
- This model supports customer feedback and the implementation of Change Requests (CR), which is not possible with traditional models such as waterfall.

- Customers see the prototype at each stage, which increases the chances of customer satisfaction.

Limitations of the Spiral Model

Here are some of the limitations of the Spiral model:

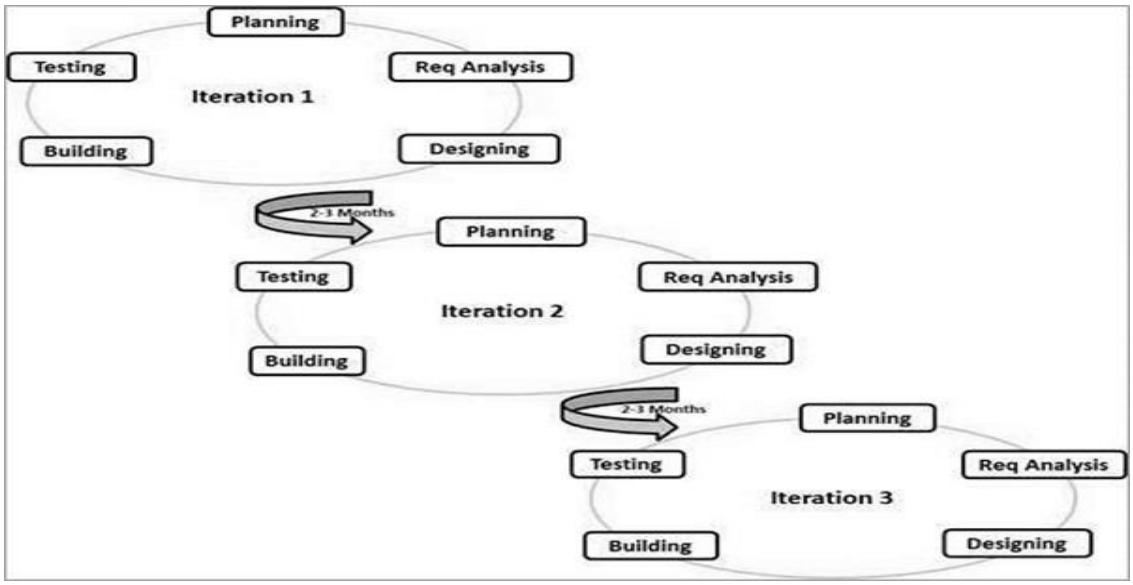
- The most obvious limitations of the spiral model are that it does not account for the uncertainty in the rate at which a given product might fail and for the variability in the rate at which a given product might fail.
- This model does not account for the fact that customers might change their mind about a given product and return it to the supplier.
- Spirals can continue indefinitely. As the spirals continue to increase, the project's cost also increases.
- Many intermediate stages require excessive documentation.
- Spiral models are best suited for projects rather than large complex small projects as continuous prototyping and evaluation help reduce risk.
- The number of phases is initially unknown, and frequent prototyping and risk analysis can exacerbate the situation, so project deadlines may not be met.
- This is very expensive and time-consuming as each stage involves prototyping and risk analysis.

Agile Model

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model −



The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.

- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Rapid Application Development (RAD)

Rapid Application Development, or RAD, means an adaptive software development model based on prototyping and quick feedback with less emphasis on specific planning. In general, the RAD approach prioritizes development and building a prototype rather than planning. With rapid application development, developers can quickly make multiple iterations and updates to the software without starting from scratch. This helps ensure that the final outcome is more quality-focused and aligns with the end users' requirements.

Steps in rapid application development

Here are the four basic steps of RAD:

1. Define the requirements

At the very beginning, rapid application development sets itself apart from traditional software development models. It doesn't require you to sit with end users and get a detailed list of specifications; instead, it asks for a broad requirement. The broad nature of the requirements helps you take the time to segment specific requirements at different points of the development cycle.

2. Prototype

This is where the actual development takes place. Instead of following a rigid set of requirements, developers create prototypes with different features and functions as fast as they can. These prototypes are then shown to the clients who decide what they like and what they don't.

More often than not, these prototypes are quickly made to work to showcase just the key features. This is normal, and the final product is only created during the finalization stage, where the client and developer are in alignment with the final product.

3. Construction

The construction stage is a crucial stage of development. Engineers and developers work tirelessly to flesh out a working system from a working model. Feedback and reviews are crucial at this stage, and most bugs, issues, and alterations are addressed. This stage can be particularly long, especially in cases where clients change directions or feedback is intensive.

4. Deployment

The final stage of RAD involves deploying the built system into a live production environment. The deployment phase involves intensive scale testing, technical documentation, issue tracking, final customizations, and system simulation. Teams also spend time debugging the app and running final updates and maintenance tasks before going live. In the dynamic process of RAD, especially during

the prototyping and construction stages, developers frequently need to experiment with various features quickly. Understanding ways to delete a git stash can be incredibly beneficial for managing interim development versions without cluttering the repository with unnecessary snapshots.

RAD Model - Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

The advantages of the RAD Model are as follows –

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.
- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

The disadvantages of the RAD Model are as follows –

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on Modelling skills.
- Inapplicable to cheaper projects as cost of Modelling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.

.