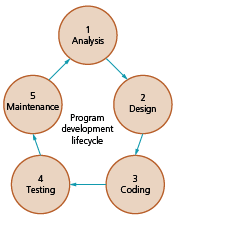
When a new software is created, the developers used a structured organized plan of how to create the program. This is called the program development life cycle

There are various types of life cycle. All these e involved the following stages

* Analysis,
* Design
* Coding
* Testing
* Maintenance

*Analysis*

Before any problem can be solved, it needs to be clearly defined and set out so everyone working on the solution understands what is needed. This is called the requirements specification. The **analysis** stage often starts with a feasibility study, followed by investigation and fact finding to identify exactly what is required from the program.

The feasibility study

This is a preliminary evaluation of the proposed software project to determine whether it is viable and feasible to proceed with the development. The feasibility study is typically conducted in the planning phase of the SDLC and is used to determine whether the project is worth pursuing from a standpoint of

* Technical-an analysis of the technical feasibility of the project, such as whether the technology required to develop the software is available and whether the project can be completed within the available time frame and budget
* Economic- an analysis of the cost-benefit of the project, return on investment (ROI), and potential revenue generation. The operational feasibility of the project is evaluated, including an analysis of the impact the software will have on existing business processes, the availability of necessary resources, and the potential for user adoption.
* operational - project is evaluated, including an analysis of the impact the software will have on existing business processes, the availability of necessary resources, and the potential for user adoption.

Based on the results of the feasibility study, the stakeholders make a decision about whether to proceed with the project, modify the project scope, or abandon the project altogether.

*Design*

In this stage, the concept is, transformed into a detailed design than can be used for manufacturing or construction.

In systems design, functions and operations are described, in detail, including screen layouts, business rules, process diagrams, and other documentation. Modular design reduces complexity and allows the outputs to describe the system as a collection of subsystems.

The design stage takes as its input the requirements already defined. For each requirement, a set of design elements is, produced.

Design documents typically include diagrams such as

* screen layouts
* business rules
* process diagrams
* pseudo-code
* a complete data model with a data dictionary.

These elements describe the system in sufficient detail that developers and engineers can develop and deliver the system with minimal additional input.

Coding

In this phase, the software design is transformed into code that can be executed by a computer. This phase involves the following

Writing code that is done according to the specification and requirement outlined in the designed documentation.

Debugging: once the code has been written, it is to be tested and debugged to ensure that it functions as intended. Developers used debugging tools to identify and fix errors in the code.

Testing

In this phase, the developers test the program to identify and correct defects, errors or issues that may affect its functionality, usability, performance and security. This usually involved different types of test. Including

* dry run
* walkthrough
* white-box
* black-box
* integration
* alpha
* beta
* acceptance
* stub

Correction of errors includes

* Syntax error
* Logic errors
* Runtime errors

A test plan is constructed in which they test the data that the system will used. These tests are

* Normal
* Abnormal
* extreme/boundary

Types of testing strategies

* Dry run also called a practice run. This is usually done in which the developer works through a program or module from a program manually and documents the results using a trace table.
* Walkthrough this is where the developers present the product to a group of reviewers. The developers will explain the software’s features, functionality and design choices. The reviewers will ask questions and provides feedback on the software design, usability and overall quality
* White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.
* Black box testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.
* Integration testing: also known as integration and testing (I&T). This testing is done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules, which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
* Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or the public. The focus of this testing is to simulate real users by using a black box and white box techniques. The aim is to carry out the tasks that a typical user might perform. Alpha testing is carried out in a lab environment and usually, the testers are internal employees of the organization. This kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing.
* Beta testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external User Acceptance Testing.
* Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.
* Acceptance testing (UAT) is a type of testing, which is done by the customer before accepting the final product. Generally, UAT is done by the customer (domain expert) for their satisfaction, and check whether the application is working according to given business scenarios, real-time scenarios.
* Stub are used during Top-down integration testing, in order to simulate the behaviour of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product. They are also used when the software needs to interact with an external system.

Errors

*Maintenance*

The program is maintained throughout its life, to ensure it continues to work effectively. This involves dealing with any problems that arise during use, including correcting any errors that become known, improving the functionality of the program, or adapting the program to meet new requirements.

The maintenance involved the following

* Perfective
* Adaptive
* Corrective

There are three kinds of maintenance needed:

Corrective maintenance: This is where problems are identified with the system after installation. Perhaps an item on the template isn't printing out correctly or maybe one of the on-screen buttons isn't linking to the correct form.

A fault report is raised, and the developers fix the problem. It is then passed over to a team of testers who check that the fault has been fixed. Once it has been passed by the testers, the fix will be released to the live system.

Corrective maintenance can also involve fixing hardware faults or replacing equipment as necessary.

Adaptive maintenance: This type of maintenance often occurs as a result of external influences or strategic changes within the company. For example, the change in the tax percent of a country. This change would have meant that many organisations had to make alterations to their systems. Perhaps a bank decides to offer a new mortgage product. This will have to be included in the system so that mortgage interest and payments can be calculated.

Perfective maintenance: The system has been in place and running fine for a while. However, over time, the end user will often find tweaks or minor improvements, which could be made to improve the way the system works. Perhaps a slightly different screen or data input form. These tweaks are not major enough to prompt a completely new system, so the maintenance team adapt the system to suit.

It is important to be aware that while the system remains in use the maintenance stage will be ongoing.

Types of SDLM

**Waterfall Model (Classical Life Cycle)**

A picture containing screenshot

Description automatically generatedThe Waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. Each phase must be completed in its entirety before the next phase can begins. At the end of each phase, a review takes place to determine if the project is on the right path and whether to continue or discard the project.

The waterfall approach does not allow going back to the previous phase to handle changes in requirement

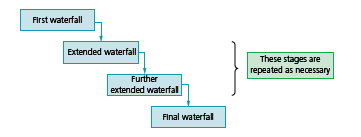
Advantages

* Simple and easy to 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
* Easy to determine the key points in the development cycle
* Easy to classify and prioritize tasks

Disadvantages

* Adjusting scope during the life cycle can kill a project
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Poor model for complex, object-oriented and long projects.
* Can be difficult to go back a step if needed.
* Poor model where requirements are at a moderate to high risk of changing.
* Software can only be used after the last stage
* Inappropriate for the long-term projects
* The progress of the stage is hard to measure while it is still in the development

The iteration model

This is a software development process that involves repeating a cycle of planning, designing developing and testing in short iteration until the software is complete. Each iteration produces a working version of the software that can be tested, and evaluated by the stakeholders, and feedback is used to inform the next iteration.

The software is designed, developed and tested in repeated cycles. With each iteration, additional features can be designed, developed and tested until there is a fully functional software application ready to be deployed to customers.

**When to use the Iterative Model?**

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. When there is a requirement of changes in future.

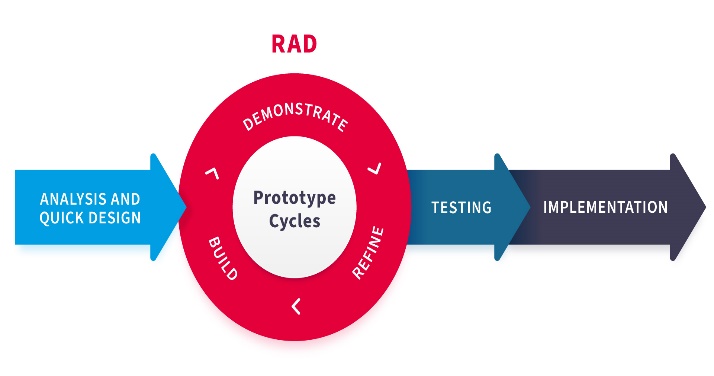
**Advantage (Pros) of Iterative Model:**

1. Testing and debugging during smaller iteration is easy.
2. A parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

**Disadvantage (Cons) of Iterative Model:**

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed repeatedly because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

Rapid Application Development

Rapid Application Development or RAD means an adaptive (easily change) software development model based on prototyping and quick feedback with less emphasis on specific planning. 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.

The development process is broken down into smaller, more manageable components or modules that can be developed independently. These modules are then integrated to create a complete software application.

The RAD model involves the use of tools and techniques that help accelerate the development process, such as

* prototyping tools
* code generators
* visual programming environments.

The RAD model also emphasizes close collaboration between developers and stakeholders, including end-users, to ensure that the software being developed, meets their needs.

The model uses iterative cycles of prototyping, testing, and feedback to refine the software and ensure that it meets the requirements.

The RAD model is known, for its speed and flexibility, as it allows for quick development and adaptation to changing requirements. However, it can also lead to less robust software if not managed properly, as the focus on speed may lead to shortcuts being taken in the development process.

Advantages

* Requirements can be changed at any time
* Encourages and priorities customer feedback
* Reviews are quick
* Development time is drastically reduced
* More productivity with fewer people
* The time between prototypes and iterations is short
* Integration isn’t a problem, since it integrates from project inception

Disadvantages

* Needs strong team collaboration
* Cannot work with large teams
* Needs highly skilled developers
* Needs user requirements throughout the life cycle of the product
* Only suitable for projects which have a small development time
* More complex to manage when compared to other models
* Only systems, which can be modularised, can be developed using Rapid application development.