

Syllabus Content:

12.1 Program Development Life Cycle

Candidates should be able to:

-  Show understanding of the purpose of a development life cycle
-  Describe the principles, benefits and drawbacks of each type of life cycle
-  Show understanding of the analysis, design, coding, testing and maintenance stages in the program development life cycle
-  Show understanding of the need for different development life cycles depending on the program being developed

Notes and guidance

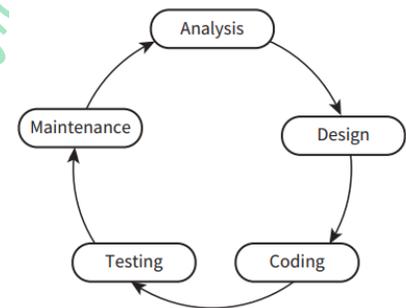
Including, waterfall, iterative, rapid application development (RAD)

12.1 Program Development Life Cycle:

Developing a program involves **different stages**.

The process of **developing a program** set out in five stages:

-  **Analysis**
-  **Design**
-  **Coding**
-  **Testing and**
-  **Maintenance**



When large software systems are required to solve big problems, these **stages are more formal**, especially when more people are involved in the development.

-  Before a solution can be designed, the **problem needs to be analysed**.
-  Then we design the solution using **Structured English**, a **flowchart** and / or **pseudocodes**.
-  Then we write the **program code** and **test it**.
-  When the program works and is being used, issues might arise that require changes. These changes are done in **maintenance stage**.

Analysis:

Analysis is the first part of the **program development lifecycle**; a process of investigation, leading to the specification of what a program is required to do.

The first step in solving a problem is to **investigate the issues** and the current system if there is one.

-  The problem **needs to be defined clearly** and precisely.
-  A '**requirements specification**' is drawn up.
-  The next step is **planning** a solution. Sometimes there is **more than one solution**. You need to decide which is the most appropriate.
-  The third step is to **decide how to solve the problem**:
 - **Bottom-up**: start with a small sub-problem and then build on this



- **Top-down:** stepwise refinement using pseudocode, flowcharts or structure charts.

Design:

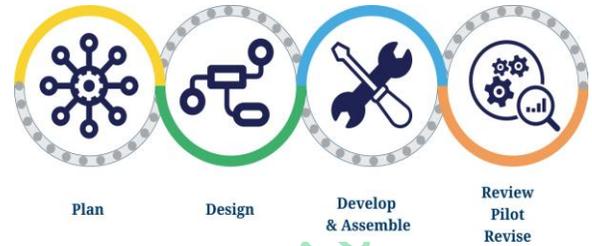
The program specification from the analysis stage is used to show **how the program should be developed**.

When the **design stage is complete**, the programmer should know what is to be done, all the tasks that need to be completed, how each task is to be performed and how the tasks work together.

This leads you to thinking about **data structures**: do you need a **1D array** or a **2D array** to store data while it is processed? Do you need a file to store data long-term?

Plan your **algorithm** by drawing a **flowchart** or writing **pseudocode**.

This can be formally documented using **structure charts**, **state-transition diagrams** and **pseudocode**.



Coding:

In this stage, the program or set of programs is written using a **programming language**.

When you have designed your solution, you might need to choose a suitable **high-level programming language**.

If you know more than one programming language, you have to weigh up the **pros and cons** of each one.

You need to decide which programming language would best suit the problem you are trying to solve and which language you are most familiar with.



Testing:

The program is run many times with different **sets of test data**, to test that it does everything it is supposed to do in the way set out in the program design.

Only thorough testing can ensure the program really works under all circumstances.

There are several **different development methodologies**.

These include:

- Waterfall model
- The iterative model
- Rapid application development model.



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 come** to light.

It also includes improving the functionality of the program, or adapting the program to meet new requirements

There are **different development life cycles models**

Each program development methodology has its **own strength**.

Different models have been developed based on the lifecycle for developers to use in practice. The models we will consider will be divided into the five stages set out above: analysis, design, coding, testing and maintenance.

We will look at three models:

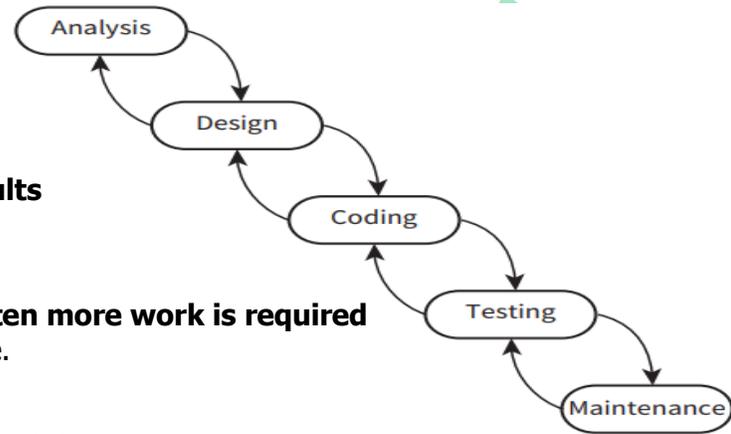
-  **The waterfall model**
-  **The iterative model**
-  **Rapid application development (RAD).**

The waterfall model:

The **arrows going down represent**: that the **results from one stage are input** into the next stage.

The **arrows leading back up**:

Represent to an earlier stage reflect the fact that **often more work is required at an earlier stage to complete** the current stage.



Benefits:

-  Simple to understand as the stages are clearly defined.
-  Easy to manage due to the fixed stages in the model.
-  Each stage has specific outcomes.
-  Stages are processed and completed one at a time.
-  Works well for smaller projects where requirements are very well understood.

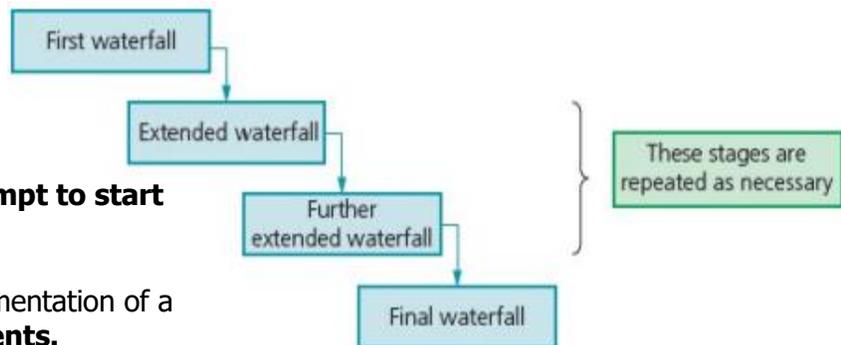
Drawbacks:

- No working software is produced** until late during the life cycle.
- Not a good model for complex** and object-oriented projects.
- Poor model for long and ongoing projects.
- Cannot accommodate changing requirements.
- It is difficult to measure progress within stages.
- Integration is done at the very end, which doesn't allow identifying potential technical or business issues early.

The iterative model:

An iterative life cycle model **does not attempt to start** with a full specification of requirements.

Instead, development starts with the implementation of a **small subset of the program requirements**.



Repeated (iterative) reviews to identify further requirements eventually result in the complete system

There is a working model of the system **at a very early stage of development**, which makes it **easier to find functional or design flaws**.



Finding issues at an early stage of development means **corrective measures can be taken more quickly**. Some **working functionality can be developed quickly** and early in the life cycle.

Benefits:

- 🛡️ **Results are obtained early** and periodically.
- 🛡️ **Parallel development** can be planned.
- 🛡️ Progress can be measured.
- 🛡️ Less costly to change the scope/requirements.
- 🛡️ Testing and debugging of a smaller subset of program is easy.
- 🛡️ Risks are identified and resolved during iteration. Easier to manage risk – **high-risk part is done first**.
- 🛡️ With every increment, **operational product** is delivered.
- 🛡️ Issues, challenges and risks identified from each increment can be utilised/applied to the next increment.
- 🛡️ Better suited for **large and mission-critical** projects.

During the life cycle, software is produced early, which facilitates **customer evaluation and feedback**.

Drawbacks:

- 🛡️ Only **large software development projects** can benefit because it is hard to break a small software system into further small serviceable modules.
- 🛡️ **More resources** may be required.
- 🛡️ Design issues might arise because not all requirements are gathered at the beginning of the entire life cycle.
- 🛡️ Defining increments may require definition of the complete system.

The Rapid Application Development (RAD) model:

RAD is a software development methodology that uses minimal planning.

Instead it **uses prototyping**.

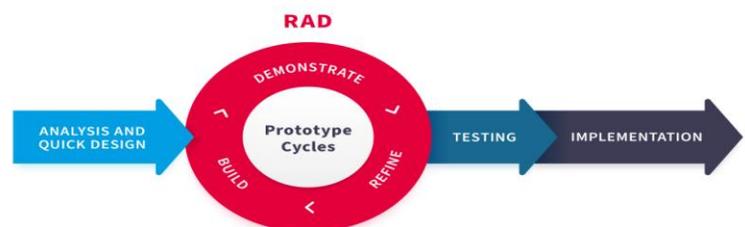
A **prototype is a working model** of part of the solution.

In the RAD model, the modules are developed **in parallel as prototypes** and are integrated to make the complete product for faster product delivery.

- 🛡️ There is no detailed preplanning.
- 🛡️ are made during the development process.
- 🛡️ The analysis, design, code and test phases are incorporated into a series of short, iterative development cycles.

(RAD) model benefits:

- 🛡️ Changing requirements can be accommodated.
- 🛡️ Progress can be measured.





-  Productivity increases with fewer people in a short time.
-  Reduces development time.
-  Increases **reusability** of components.
-  Quick initial reviews occur.
-  Encourages **customer feedback**.
-  Integration from very beginning solves a lot of integration issues.

Drawbacks: include the following.

-  Only systems that can be **modularised** can be built using RAD.
-  Requires highly skilled **developers/designers**.
-  Suitable for systems that are **component based** and **scalable**.
-  Requires user involvement throughout the life cycle.
-  Suitable for projects requiring shorter development times.

References:

-  AS & A level Course Book by Sylvia Langfield & Dave Duddell
-  A level 9608 Pastpapers

Computers(9608) with Majid Tahir at www.majidtahir.com