

OCR Computer Science AS Level

2.2.2 Software Development Intermediate Notes

Specification:**a) Programming methodologies**

- Waterfall lifecycle
- Agile methodologies
- Extreme programming
- Spiral model
- Rapid application development

b) Merits, drawbacks and uses of programming methodologies**c) Writing and following algorithms****d) Test strategies**

- Black box testing
- Alpha testing
- Beta testing

e) Test programs that solve problems using suitable test data and end user feedback

- Justify test strategy for given situation

Programming Methodologies

Software development life cycles (SDLCs) have some stages in common, including:

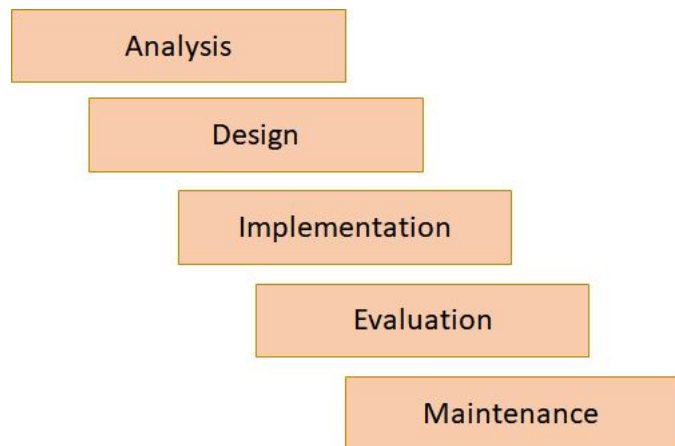
- Analysis
 - Stakeholders state what their requirements which are used to define the problem and the system requirements.
- Design
 - The different aspects of the new system are designed, such as:
 - Inputs: *volume, methods, frequency*
 - Outputs: *volume, methods, frequency*
 - Security features: *level required, access levels*
 - Hardware set-up: *compatibility*
 - User interface: *menus, accessibility, navigation*
 - A test plan may also be designed at this stage.
- Development
 - The project is split into individual, self-contained modules, which are allocated to teams for programming.
- Testing
 - The program is tested against the test plan formed in the Design stage.
- Implementation
 - Once the software has been tested and reviewed, it is installed onto the users' systems.
- Evaluation
 - The effectiveness of the software is evaluated against the system requirements. Different criteria are considered, including robustness, reliability, portability and maintainability.
- Maintenance
 - Any errors or improvements that could be made to the software are flagged up by the end-users. Programmers will regularly send out software updates to fix any bugs, security issues or make any necessary improvements.

Waterfall lifecycle

The traditional waterfall model of software development is widely recognised and is based on a series of stages which are **completed in sequence**, from start to finish.

The analysis stage includes a **feasibility study** in which designers evaluate the feasibility of the project using 'TELOS':

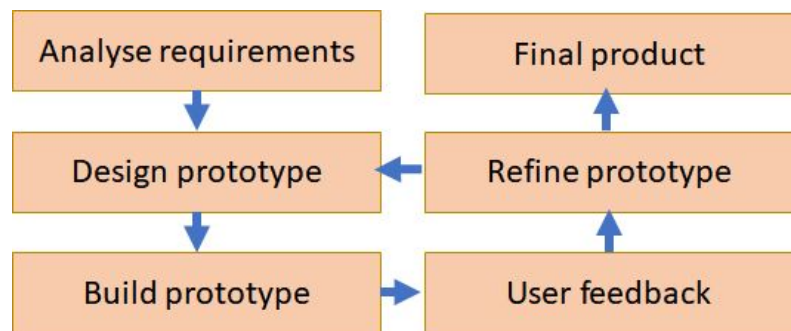
- Technical
- Economic
- Legal
- Operational
- Scheduling



If a change needs to be made within a project being developed using the waterfall model, **programmers must revisit all levels** between the current stage and the stage at which a change needs to be made. **Users have little input** as they are only involved at the very beginning and end of the waterfall lifecycle, during the analysis and evaluation stage.

Agile methodologies

This refers to a **collection of methodologies which aim to improve the flexibility of software development and adapt to changes in user requirements faster**.



The problem is broken down into **sections which are developed in parallel**. Different sections of software can be at **different stages of development**. A **working prototype is delivered early on** and prototypes are built upon and improved in an **iterative manner** so that **new prototypes are delivered regularly** throughout the course of the development cycle.

In agile development methodologies, there is **less of a focus on documentation** and more priority is given to **user satisfaction**.

Extreme programming

This is an **agile model** in which the development team consists of a **pair of programmers alongside a representative end-user**. The model is built on 'user stories': system requirements are specified by the end-user. Paired programming produces **high-quality code** and programmers work **no longer than forty hours a week**.

The iterative development means that it is hard to produce high quality documentation.

Spiral model

The spiral model is built on **four key stages** with the focus of **effectively managing risk-heavy projects**:

- Analysing system requirements
- Pinpointing and mitigating risks
- Development, testing and implementation
- Evaluating to inform the next iteration



If the project is found to be too risky at any point, the project is terminated. However hiring risk assessors to analyse the risks involved can be **expensive**, which makes this methodology **suited to only very large-scale projects**.

Rapid application development

RAD is an **iterative methodology** which uses **partially functioning prototypes** which are **continually built-upon**. User requirements are **initially gathered using focus groups** and used to develop an '**incomplete**' version of the solution which is given to the user to trial.

User feedback is then used to generate the next, **improved prototype** and this continues until the prototype matches the requirements of the end-users at which point it becomes the final product.

This is commonly used where **user requirements are incomplete or unclear at the start**. However, as requirements change over the course of the project, additions and changes made to the code may be **inefficient**.

Writing and following algorithms

Algorithms are a **set of instructions used to solve a problem**. They are core to computer science and can be used to tackle a wide range of problems. Regardless of the problem, all good algorithms have certain **key qualities** which are highlighted below

:

- Inputs must be **clearly defined** - what is valid and what is invalid?
- Must **always** produce a **valid output for any defined input**
- Must be able to **deal with invalid inputs**
- Must always reach a **stopping condition**
- Must be **well-documented** for reference
- Must be **well-commented** so modifications can easily be made

Merits, drawbacks and uses of programming methodologies

	Merits	Drawbacks	Uses
Waterfall	<ul style="list-style-type: none"> - Straightforward to manage - Clearly documented 	<ul style="list-style-type: none"> - Lack of flexibility - No risk analysis - Limited user involvement 	Static, low-risk projects which need little user input, such as a piece of general-purpose software
Agile	<ul style="list-style-type: none"> - Produces high quality code - Flexible to changing requirements - Regular user input 	<ul style="list-style-type: none"> - Poor documentation - Requires consistent interaction between user and programmer 	Small to medium projects with unclear initial requirements.
Extreme Programming	<ul style="list-style-type: none"> - Produces high quality code - Constant user involvement means high usability 	<ul style="list-style-type: none"> - High cost of two people working on one project - Teamwork is essential - End-user may not be able to be present 	Small to medium projects with unclear initial requirements requiring excellent usability.

<p>Spiral</p>	<ul style="list-style-type: none"> - Thorough risk-analysis and mitigation - Caters to changing user needs - Produces prototypes throughout 	<ul style="list-style-type: none"> - Expensive to hire risk assessors - Lack of focus on code efficiency - High costs due to constant prototyping 	<p>Large, risk-intensive projects with a high budget.</p>
<p>Rapid Application Development</p>	<ul style="list-style-type: none"> - Caters to changing user requirements - Highly usable finished product - Focus on core features, reducing development time 	<ul style="list-style-type: none"> - Poorer quality documentation - Fast pace may reduce code quality 	<p>Small to medium, low-budget projects with short time-frames.</p>

Test Strategies

There are various types of testing that can be carried out:

- Alpha testing
 - Alpha testing is **carried out in-house** by the software development teams within the company. Bugs are pinpointed and fixed.
- Beta testing
 - Beta testing is **carried out by end-users** after alpha testing has been completed. **Feedback from users** is used to inform the next stage of development.
- White box testing
 - This is **carried out by software development teams** in which the **internal structure of the program** is known. All of the **possible routes through the program** are tested.
- Black box testing
 - Software is tested **without testers being aware of the internal structure** of the software. The test plan traces through **inputs and outputs** within the software.

The testing stage of software development is designed to pinpoint any flaws in the software. Depending on the type of program, programmers must select the appropriate test strategy. Software should produce the correct output for any given input and should produce a designated error if the data entered is invalid.

When testing, programmers should record the purpose of the test, type of test, the expected result and the actual result of the test. There are three special types of data that programmers consider when testing:

- Normal
 - Data within the range of the data type considered as valid.
- Boundary
 - Data that falls at either of the ends of the valid data range.
- Erroneous
 - Data that falls outside of the valid data range

Another test strategy is performing a [dry-run](#) of a program, which is when programmers manually work through the code. A [trace table](#) is used to note when and which variables are updated. If an unexpected result is produced, the cause of this can be traced and fixed.