NAME: OLORUNFEMI-OJO DANIEL TOMIWA

MATRIC NO: 190805503

COURSE: CSC 316

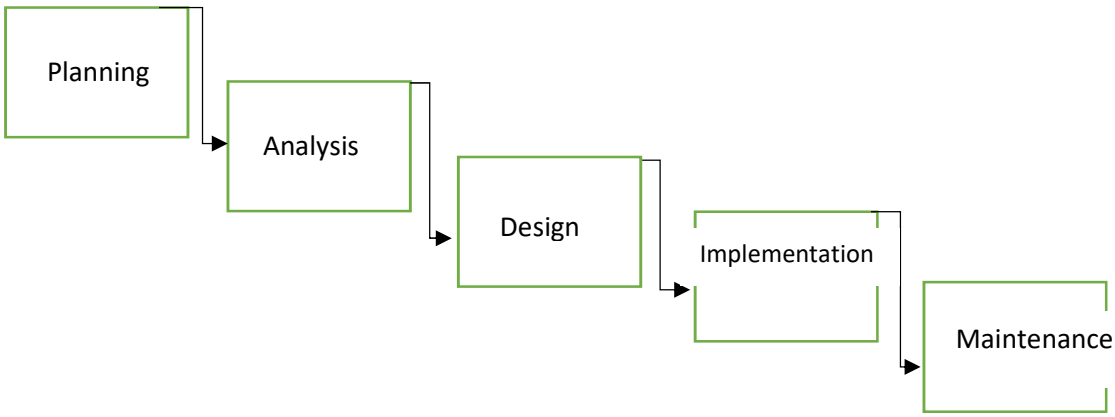
TITLE: SOFTWARE DEVELOPMENT METHODOLOGIES

SOFTWARE DEVELOPMENT METHODOLOGIES

WATERFALL METHODOLOGY

The waterfall software development approach is a downward-flowing stage model for developing software requiring substantial upfront design. It is best used if there is a stable product definition, and we pretty much know how/what needs to be done to create the software. The whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete. Waterfall is called as such because the model develop systematically from one phase to other in a downward fashion, like a waterfall and the most probable phases through which it progresses downwards are:

1. Planning
2. Analysis
3. Design
4. Implementation
5. Maintenance



As seen above, the model runs from one stage to the other, its flows downward and never backwards. This methodology works best for projects where the requirements are clear. Some advantages of this methodology are:

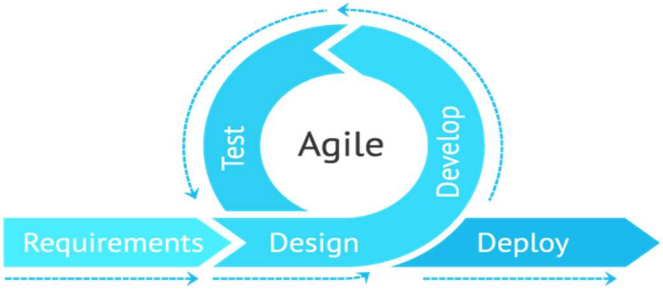
- The timeline and cost of the project can be estimated since all the requirements are predefined
- Progress is easily measure since the project scope is known
- Each phase is properly documented

Waterfall model has its downsides. They include:

- It is not flexible and therefore, unsuitable for long term projects.
- If one phase is delayed, it hinders the progress of the whole project

AGILE METHODOLOGY

The agile methodology differs greatly from the waterfall approach in two major ways; namely in regards to linear action and customer involvement. Agile is a nimble and iterative process, where the product is delivered in stages to the customer for them to review and provide feedback. An agile project is characterized by having a large number of short delivery cycles and priority is given to the feedback. Requirements are constantly evaluated and feature priorities are either upgraded or downgraded depending on customer intervention. Agile development is suitable when there is a high degree of uncertainty and risk in a project that arises from frequently changing requirements and/or the novelty of technology used.



Agile methodology lessens the risk of project failure thanks to the fact that it:

- Easily adapts to changes
- Actively engages customers to ensure user satisfaction
- Design flaws are discovered quickly

Some disadvantages of this methodology include:

- The process is time consuming
- The iterative nature of agile development may lead to a frequent refactoring if the full scope of the system is not considered in the initial architecture and design. The system can suffer from a reduction in overall quality.

Agile has served as a model for the development of other software development methodologies like Scrum, Kanban and Extreme Programming.

SCRUM

Scrum is an Agile framework consisting of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage. The software is built using time boxed iterations called sprints. There are three main roles on a Scrum project:

- Product Owner: this role represents the product's stakeholders and the voice of the customer. The person is accountable for the backlog, and maximizing the value that team delivers to the business. The person defines the product in customer-centric terms such as user stories, adds them to the product backlog, and prioritizes them based on importance and dependencies.
- Scrum Master: this role is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The role is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences. The role is also referred to as a team facilitator or servant-leader.
- Team member: someone that does all tasks required to build the product increments (analysis, design, development, testing, documentation).

Scrum projects follow these rules:

- Each sprint starts with sprint planning done by the Scrum Master Product Owner and the rest of the team. They determine which features from the backlog they will build and the team works throughout the sprint to build all of the features in it.
- Daily meetings are held where everyone shares what they have been doing, what they are planning to do next and they should state if they have problems preventing then to do what they are planning to do.
- Each sprint is time boxed to a specific length decided during sprint planning.
- At the end of the sprint, the team holds a sprint review meeting where they demonstrate working software to users and stakeholders.
- After the sprint, the team holds a sprint retrospective meeting to find specific ways to improve how they work.

EXTREME PROGRAMMING (XP)

Extreme programming techniques are specific to programming and are aimed at addressing the most common problems that cause teams to build lousy code. Some characteristics of the Extreme Programming methodology are:

- Test-First Programming: before the code is written, an automated test is built. Since the product code hasn't been written yet, the test fails. He knows that the code is

working when the test passes. This creates a tight feedback loop that helps to prevent defects: write failing tests, build code to make them pass, find problems and learn more about how you will solve them, then write more failing tests. These automated tests are called unit tests.

- Pair Programming: two developers sit together at a single work station when writing code. In most cases, on programmer types, while the other watches—and they constantly discuss what is going on. Teams working in pairs inject far fewer bugs, because there are two sets of eyes looking to catch the bugs all the time.
- 10 minute build: the team creates an automated build for the entire codebase that runs in under 10 minutes. This build includes automatically running all of the unit tests and generating a report of which tests passed and which failed. Running the build often is very valuable to the team, because it makes problems immediately obvious. For example, the build runs the unit tests, so after the build runs, the team knows whether or not they have reached the level of quality they set up in the automated tests.
- Continuous Integration: you integrate your work to the shared main repository many times a day, triggering an automatic build of the whole system. Integrating as early and often as possible dramatically reduces the cost of the integration as it makes it less likely for merge and logical conflicts to occur. It also exposes environmental and dependency problems.

Development in XP happens in two main cycles: the weekly cycle and the quarterly cycle i.e. weekly cycle and quarterly cycle. The weekly cycle starts with a meeting where the customer chooses which stories they want to implement during the week. The team reviews their work, including last week's progress, and thinks of how to improve their process. The quarterly cycle practice is for long-term planning. Once a quarter, the team meets to take a look at the big picture. Some of the benefits of XP are

- Better communication since each team member is aware of the work every other person is doing.
- Developers are made to write the most simple and direct solutions.
- Constant tests and feedback loops keep the quality of the product in check.

KANBAN

Kanban is an agile process tool that enables project teams to work better. Specifically, Kanban gives teams planning flexibility, faster output, clear focus, and transparency throughout the development cycle. The core rules behind Kanban are:

- Visualize the workflow. Jobs are split into pieces, each item written on a card and put on a board called Kanban board.

- Limit work in progress. Limits are set on how many items can be in progress at each workflow state.
- Optimize the process to make lead time as small and predictable as possible
- Implement feedback loops

With Kanban, bottlenecks become clearly visible in real-time. This leads people to collaborate to optimize the whole value chain rather than just their part. Kanban provides a way to do agile software development without necessarily having to use time-boxed fixed-commitment iterations such as Scrum sprints.

Projects where changes are difficult to deploy to production on a regular basis will have a hard time using Kanban. This is because Kanban requires you to have the ability to move small items of work into production in order to pull in new items into the work in progress column.

LEAN

Lean is different. Unlike Scrum and XP, Lean does not include a set of practices. Lean is a mindset, and just like with the mindset for Scrum or XP, Lean comes with values and principles. The mindset of lean is called Lean Thinking. It may also be referred to as Minimum Viable Product (MVP) strategy as these ways of thinking are very much alike since both intend to speed up development by focusing on new deliverables. Some of the lean values are:

- Find the work that you're doing that doesn't directly help to create valuable software and remove it from the project.
- Use feedback from your project to improve how you build software.
- Understand the cost of delay, and minimize it using pull systems and queues.
- Make every important decision for your project when you have the most information about it.
- Build software that intuitively makes sense to the users, and which forms a coherent whole.

EVOLUTIONARY DEVELOPMENT MODEL

It is a model whose stages consist of expanding increments of an operational software product, with the directions of evolution being determined by operational experience. It is a combination of Iterative and Incremental model of software development life cycle.The Evolutionary Development Model is best suited to fourth generation languages because it gives users a rapid initial operational capability and provides a realistic operational basis for determining subsequent product improvements.

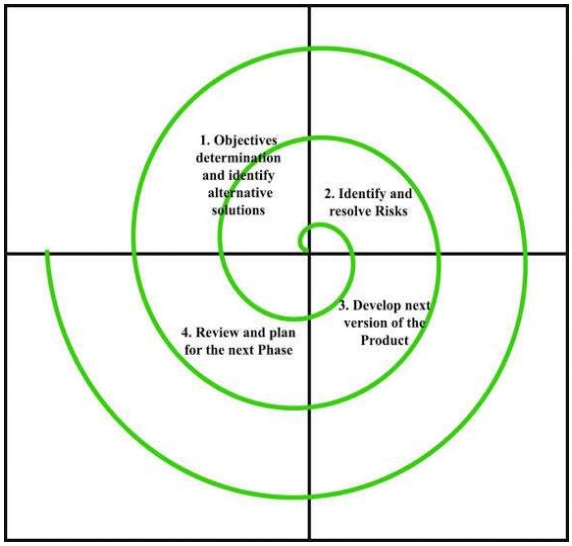
Nonetheless, evolutionary development also has its difficulties. It is generally difficult to distinguish it from the old code-and-fix model. It is also based on the often-unrealistic assumption that the user's operational system will be flexible enough to accommodate unplanned evolution paths. This assumption is unjustified in three primary circumstances:

- Circumstances in which several independently evolved applications must subsequently be closely integrated.
- Temporary workarounds for software deficiencies increasingly solidify into unchangeable constraints on evolution.

Under such conditions, evolutionary development projects have come to grief by pursuing stages in the wrong order: evolving a lot of hard-to-change code before addressing long-range architectural and usage considerations.

SPIRAL

It creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. The angular dimension represents the progress made in completing each cycle of the spiral. The model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its levels of elaboration, from an overall concept of operation document down to the coding of each individual program. Each typical spiral begins with identifying the objectives, alternate means of implementing that portion and the constraints imposed on the application.



The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

RATIONAL UNIFIED PROCESS (RUP)

It is a software development process for object-oriented models. It is designed and documented using UML (Unified Modeling Language). Tne Rational Unified Process is designed, developed, delivered, and maintained like any software tool. The Rational Unified Process shares many characteristics with software products:

- Rational Software releases regular upgrades
- It can be tailored and configured to suit the specific needs of a development organization.

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML gives you a standard means of writing the system's blueprints, covering conceptual items such as business processes and system functions as well as concrete items such as classes written in a specific programming language, database schemas, and reusable software components. The UML is a common language to express the various models, but it does not tell you how to develop software. It provides the vocabulary, but it doesn't tell you how to write the book. The Rational Unified Process is a guide to the effective use of the UML for modeling. It describes which models you need, why you need them, and how to construct them.

**UML Diagram:*