ALBUKHARY INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTING AND INFORMATICS
SEMESTER 1 2024/2025

CCC2143
SOFTWARE ENGINEERING
INDIVIDUAL ASSIGNMENT (20%)

NAME:	**Maria Iqbal**
STUDENT ID:	**AIU23102118**
GROUP:	**E**
TITLE:	**SOFTWARE PROCESS MODELS ANALYSIS AND CASE STUDY**

For Examiner's use only

RUBRIC	MARKS
SOFTWARE PROCESS MODEL (32)	
REPORT (8)	
TOTAL (40)	

Table of Contents

1.0 Introduction to Software Engineering Process Models

A “Software Development Process Models”, often referred to as Software Development Life Cycle (SDLC) is a description of the work practices, tools, and techniques used to develop software. For instance**,** Pressman (2014) defines SDLC as "a framework for planning, structuring, and controlling the process of developing an information system". It ensures methodical progress by directing the development process from the first stages of planning and requirements collection to testing and deployment. The SDLC "provides a framework for the management and control of software projects," on the other hand, according to Sommerville (2011), underscoring the significance of project management in delivering software on schedule and within budget.

These models provide all the structured frameworks for planning, organizing, and controlling the development of software system. Depending on the project size, complexity, risk, and team experience the choice of a process model is been made.

The software process models that are developed with high quality supports developers in creating software that:

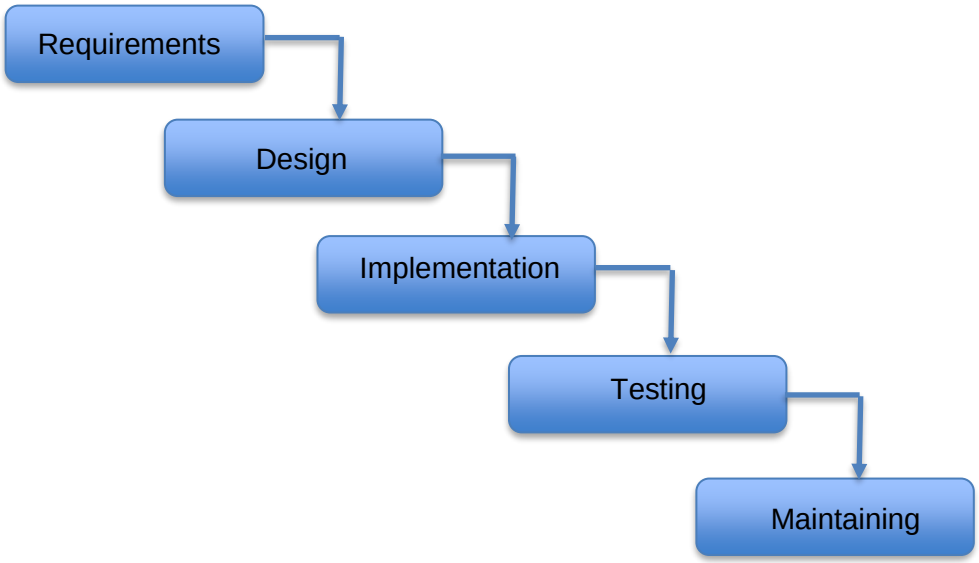
- Meets customer requirements
- Have affordable price
- And contain low numbers of defects
- Facilitate communication among stakeholders and team members

2.0 Analysis of Software Process Models

2.1 Waterfall Model

The Waterfall model is a sequential development method that moves through all project phases (analysis, design, programming, testing, and maintaining, for instance) in a waterfall fashion, with each phase finishing up entirely before the next one starts. It helps project managers to structure processes, organize tasks, set up Gantt charts and schedule, monitor project progress.

5 stages of Waterfall Model



1. Requirements

The foundation of the Waterfall methodology is the idea that all project needs may be precisely stated at the outset. Each step is described in detail, including expenses, risks, dependencies, success indicators, and schedules, guaranteeing a methodical, sequential development process.

2. Design

A technical solution to the problems outlined by product requirements including layouts, data models, scenarios, is design by software developer. Firstly logical design is formed that defines the purpose of project and integration points. After completion, logical design is transformed into a physical design that uses specific hardware and software technologies.

3. Implementation

Technical implementation begins after design is finished. As the most important phase requirements and design have been already completed, this stage of Waterfall process might be faster. Based on requirements and design programmers create applications through code.

4. Testing

To ensure product has no errors, testing needs to be done before product is released to customers ensuring good and positive user experience. To develop their test cases, the testing team will use the product manager's design documents, personas, and user case scenarios.

5. Maintaining

This is the last stage of Waterfall which begins after software is been deployed to market. A team is assigned to release new versions of software, updates. They also modify the software based on user requirements and requests.

Advantages

- Developers can identify design errors early in the analysis and design stages, reducing the chances of writing faulty code during implementation.
- Once the requirements are defined, the total project cost and timeline can be accurately estimated.
- Customers don't frequently introduce new requirements, which helps prevent delays in production.

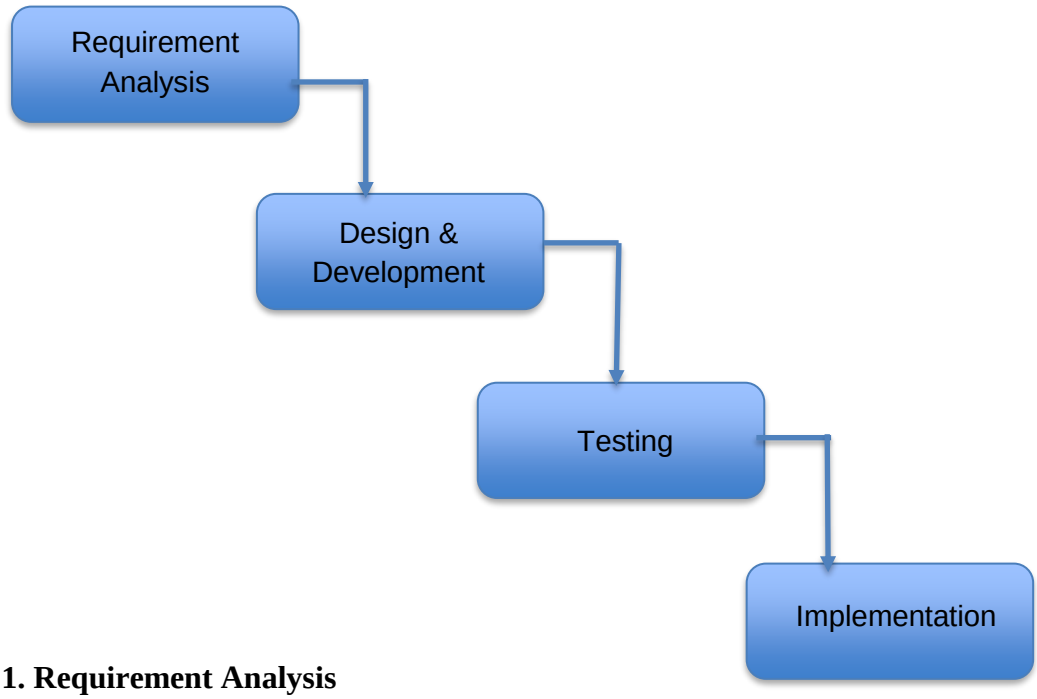
Disadvantages

- Projects may take longer to complete with this sequential approach compared to iterative methods like Agile.
- Clients often request revisions and new features later in the process, when they are harder to implement, as their initial requirements may not be fully clear.
- If one phase in the process is delay, all the other phases are delayed.

2.2 Incremental Model

It is a software development approach where project is broken down into smaller, manageable parts called increments. Firstly, a version with essential features is develop and delivered to customer. After that, through some iterations, improvements and new features are added in successive increments. Each and every increment after the development is tested and handed over to the client allowing the client to give continuously feedback at every stage. Process continues until the time system is functioning fully and it been finalized.

Phases of Incremental Model



1. Requirement Analysis

The product analysis team determines the needs in the first stage of the incremental model. Additionally, the requirement analysis team is aware of the functional needs for the system. This stage is essential to the software development process using the incremental paradigm.

2. Design & Development

The design of the system functionality and the development process are successfully completed in this stage of the SDLC's incremental model. The incremental model makes use of style and development phase when software gains new practicality.

3. Testing

The testing phase of the incremental approach examines how well each current function and new capability execute. During the testing phase, each task's behaviour is tested using a variety of techniques.

4. Implementation

The development system's coding phase is made possible by the implementation phase. In the designing and development phase, it includes the final coding, and in the testing phase, it tests the functioning. Following the conclusion of this stage, the number of functional products is improved and upgraded to the final system product.

Advantages of Incremental Model

- Errors can be recognized easily
- Testing and debugging is easy
- Flexibility
- Because it handled risk during its iteration, it is easy to manage.
- Client receives important functionality early

Disadvantages of Incremental Model

- Good planning is needed
- Costly
- Module interfaces that are well defined are needed

2.3 Integration & Configuration model

In software development, the Integration and Configuration Model concentrates on putting together pre-existing components and setting them up to function as a single, integrated system. When there are standardized, reusable components that may be combined to save down on development time and expenses, this model is frequently used. Rapid development and customization are made possible by its reliance on pre-existing modules or systems that can be modified to satisfy particular business needs.

Key Elements

1. Component Reusability: The approach places a strong emphasis on reusing previously created components, which speeds up deployment and requires less work during development.

2. Configuration over Customization: Teams minimize the requirement for substantial new code by configuring pre-existing components to meet business demands rather than starting from scratch.

Advantages

- The development time is significantly reduced because of use of existing components
- Cost for maintenance and development is lower as only few resources are needed for coding and testing
- It is flexible as it can be upgrade easily without affecting the whole system.

Disadvantages

- **Complex Integration**: It can be difficult to integrate diverse systems, especially when working with legacy systems or data formats that differ.

- **Dependency on External Components:** The compatibility and quality of external components determine how reliable the system is.

3.0 Comparison between Software Process Models

Aspects	Waterfall	Incremental	Integration & Configuration
i. Structure. Flexibility and adaptability	Follows linear structure, low flexibility and adaptability	More flexible allowing gradual development in small iterative cycles, also allow changes in each increment	It is highly flexible because of using pre-existing components that can be configure to meet new requirements
ii. Customer/ user involvement	Customer is only involved at the beginning of development	Customer involvement is intermediate	Customer involvement is generally high while configuration phase, as they can customized the system
iii. Risk management	High amount of risk	Low amount of risk	Strong risk management
iv. Project management perspective	It is structured clearly but can become inflexible if changes are needed	Requires careful planning for each and every increment, remain flexible if	Focuses on coordinating components and configurations, with

		changes are needed in the middle of project	an emphasize alignment to meet goals

4.0 Case study

4.1 Adapting Software Development Processes for Small Teams: A Case Study on Mobile Game Design

Introduction

The significance of choosing the appropriate development method for software projects, especially in small teams, is emphasized in the introduction of Rob Jansen's case study. Although the majority of research focusses on big businesses, this study examines a cooperative undergraduate team that used Java 2 Micro Edition to create a Java-based mobile game. It contrasts their procedure with well-known approaches such as Extreme Programming, Spiral, Waterfall, and Unified Process. According to the study's findings, the team's strategy was too intricate for a small group, indicating that settings requiring close contact are better suited for more straightforward, flexible approaches.

System requirements

- **Project Definition:** A succinct explanation of the project's purpose and intended audience.
- **User Personas:** To accommodate various user types, create representative "artificial users" with an emphasis on meeting the principal persona.
- **User Scenarios:** Explain how users engage with the technology in authentic gaming scenarios.
- **System Use examples:** Use UML diagrams to refine scenarios into functional system flows, encompassing both typical and exceptional examples. Describe the main screen functions and user choices in the user interface.
- **Storyboarding:** Show the order of user inputs, actions, and system reactions at strategic moments in time.
- **Detailed specifications:** Enumerate the constraints, non-functional requirements, and functional requirements.

Justification

The team employed elements of multiple software development models in the game development project:

Waterfall: During the design phase, the team used a Waterfall methodology, carrying out tasks one after the other (e.g., establishing system requirements, creating user interfaces, and developing specific requirements). This approach offered a methodical procedure in which every step had to be finished before going on to the next.

Spiral: By implementing iterative development, the project acquired traits of the Spiral model during the build phase. Similar to the Spiral model's emphasis on ongoing improvement through iterations, the team ran into problems during the coding process that caused changes to the system designs. The project was able to develop because suggestions and modifications were incorporated into the development process.

Extreme Programming (XP): The emphasis on straightforward design and the flexibility of the code were characteristics of Extreme Programming. Although not mentioned directly, the small group's flexibility and emphasis on communication, as well as the ability to modify designs while the project is being built, are in line with XP values such as frequent iterations, open communication, and prompt feedback.

Unified Process: By segmenting the development into design and build phases and iterating on requirements, testing, and design, the team also illustrated elements of the Unified Process. Similar to the Unified Process's focus on incremental, iterative phases and ongoing input, they employed user scenarios and requirements to guide development.

4.2 A Study on Configuration and Integration of Sub-Systems to System-of-Systems with Rule Verification

Introduction

The introduction emphasizes how the complexity of contemporary software systems is increasing, especially as cyber-physical systems and the Internet of Things (IoT) expand in popularity. This has led to the creation of System-of-Systems (SoS). A component-oriented strategy, which separates systems into separate components with defined interfaces, is frequently used to manage this complexity. Although this approach organizes development, it does not guarantee system accuracy or appropriate component interaction, hence verification tools such as UPPAAL are required.

The integration and configuration paradigm is used in this study, with an emphasis on putting together separately created parts into a coherent system and setting them up to guarantee proper operation. The study investigates how specifications that are valid in separate sub-systems might not remain valid when these are integrated into a SoS by employing UPPAAL for real-time verification.

System-of-Systems (SoS) Integration Process

1. Collecting and Adjusting Components

- **Component removal:** First, the sub-systems are cleared of all dummy parts and assumed environments.
- **Channel adjustment:** To guarantee proper communication, interface channels must be changed because disconnected channels can cause some components to stop working.
- **Manual synchronization:** To provide seamless communication between components, synchronization instructions are introduced to connect the edges correctly. To avoid inconsistent data within the SoS, this involves eliminating any unnecessary choose commands.

2. Managing Constraints

- The set of limitations that were first disregarded must be reinstated after the components have been connected. This guarantees that the integrated system operates under the desired circumstances.
- To guarantee uniformity throughout the system, a constraint pertaining to the CMDB component is validated in this case.

Justification

Component-Oriented Development: The integration and configuration model is characterized by the system's division into separate parts with clearly defined interfaces.

Verification of Interaction: A major concern in this development approach, the research focusses on confirming the interactions between different parts when they are incorporated into a bigger system.

UPPAAL Use: The technique seeks to guarantee that system configurations satisfy particular requirements by using UPPAAL to validate both individual components and their integration into a System-of-Systems (SoS).

5.0 Recommendation & Conclusion

5.1 Recommendation

The following suggestions are put forth to improve software development procedures in diverse contexts based on the examination of various software development models and the case studies that were discussed:

1. Choose the Right Model Based on Project Needs:

- **Waterfall:** Ideal for projects with clear specifications and few anticipated modifications. Perfect for projects that require a lot of paperwork and have distinct, well-organized phases.
- **Incremental:** It is advised for projects that need early delivery of essential features, regular user input, and iterative improvement. Particularly advantageous in settings where needs change throughout time.
- **Integration & Configuration:** Perfect for applications that make use of legacy systems or pre-existing components. This methodology is especially well-suited for complex systems that need to be deployed and customized quickly, such System-of-Systems (SoS) development in cyber-physical and Internet of Things environments.

2. Emphasize verification & validation:

- Employ verification tools like UPPAAL to ensure real-time system correctness and effective interaction between components in complex SoS environments. This helps in identifying inconsistencies early and improves overall system reliability.

3. Enhance risk management practices:

- Put into practice strong risk management plans customized for each model. For instance, integration models should concentrate on reducing reliance risks from external components, but incremental approaches should include frequent risk assessments at every iteration.

4. Strengthen communication and stakeholder involvement:

- Success requires regular stakeholder involvement and open lines of communication, particularly in iterative models. Constant feedback loops guarantee that the finished product fulfils user requirements and changes to suit changing demands.

5. Adopt component-Oriented development:

- Adopt modular design and component reusability to cut down on development time and expense. This method also increases system flexibility and makes future updates easier.

5.2 Conclusion

Any project's success depends on selecting an appropriate software development process model. Each of the three models which are Waterfall, Incremental, and Integration & Configuration has special advantages and disadvantages that make them suitable for certain project kinds and organizational requirements.

The Integration & Configuration paradigm is notable for its focus on component reuse and quick development in the context of contemporary software systems, especially in System-of-Systems (SoS) and Internet of Things contexts. To guarantee system integrity and dependability, robust verification techniques like UPPAAL are necessary due to the complexity of integrating disparate components.

In the end, producing high-quality software that effectively satisfies both functional and non-functional criteria requires matching the development model with project objectives, stakeholder expectations, and risk management techniques.

6.0 References

Adobe. (n.d.). *What is the Waterfall methodology?* Adobe Blog. https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20%E2%80%94%20also%20known,before%20the%20next%20phase%20begins

Javatpoint. (n.d.). *Software engineering - Incremental model.* Javatpoint. https://www.javatpoint.com/software-engineering-incremental-modelPMI. (n.d.). *Configuration management: Controlling changes to maintain integrity*. Project Management Institute. https://www.pmi.org/learning/library/configuration-management-control-changes-4980

Pressman, R. S. (2014). *Software engineering: A practitioner's approach* (8th ed.). McGraw-Hill Education.

Sommerville, I. (2011). *Software engineering* (9th ed.). Addison-Wesley.

Wikipedia. (2023). *System integration*. In *Wikipedia*. https://en.wikipedia.org/wiki/System_integration

https://www.scribd.com/doc/117927198/waterfall-model-case-study

https://www.scirp.org/journal/paperinformation?paperid=60588