**ĐẠI HỌC QUỐC GIA TP. HCM**

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÀI TẬP VỀ NHÀ BUỔI 2

# MỞ ĐẦU

**Giáo viên hướng dẫn: ThS. Nguyễn Thị Thanh Trúc**

**Nhóm thực hiện:**

1. **Trương Văn Khải 21520274**
2. **Lê Yến Nhi 21520274**
3. **Ngô Phúc Danh 21521924**
4. **Lê Ngô Minh Đức 21520195**

**TP. HỒ CHÍ MINH, 2023**

## 1. *What are the fundamental activities that are common to all software processes?*

There are several fundamental activities that are common to all software processes, regardless of the specific methodology or framework being used. These activities are:

- **Requirements gathering and analysis -** This involves understanding the needs and expectations of the stakeholders, and defining the functional and non-functional requirements of the software system.
- **Design -** Based on the requirements, the software design is developed, including the architecture, interfaces, and data structures.
- **Implementation or coding -** The actual coding of the software is done in this phase, using programming languages and tools.
- **Testing -** Once the software is developed, it is tested to ensure that it meets the requirements and is free from defects.
- **Deployment -** The software is deployed or installed on the target environment, which may include hardware, operating system, and other software components.
- **Maintenance -** After the software is deployed, it may require updates, bug fixes, and other changes to keep it functioning properly.
- **Project management -** Throughout the software development process, project management activities are performed to ensure that the project is on track, within budget, and meets the required quality standards.

These activities are often iterative and overlapping, and may be performed in different orders depending on the specific software development process being used.

## 2. *List the 3 generic process models that are used in software engineering?*

There are three generic process models that are commonly used in software engineering:

- **Waterfall model -** The waterfall model is a linear sequential approach to software development, where the process flows from one phase to the next in a sequential manner. This model consists of several phases, including requirements gathering, design, implementation, testing, deployment, and maintenance.
- **Iterative model -** The iterative model is a cyclical approach to software development, where the process cycles through the phases of requirements gathering, design, implementation, and testing multiple times until the software is completed. Each

iteration results in a working product increment that can be reviewed and evaluated.

- **Agile model -** The agile model is a flexible and adaptive approach to software development that emphasizes collaboration, self-organization, and rapid iteration. The agile model is based on the principles outlined in the Agile Manifesto and consists of several frameworks such as Scrum, Kanban, and Extreme Programming (XP).

These process models have their strengths and weaknesses, and the choice of a particular model depends on the nature of the project, the requirements, and the development team's experience and preferences.

## *3. Why are iterations usually limited when the waterfall model is used?*

In the waterfall model, iterations are usually limited because the model follows a sequential and linear approach to software development, where each phase is completed before the next one can begin. The phases in the waterfall model, such as requirements gathering, design, implementation, testing, deployment, and maintenance, are performed in a specific order, and each phase is completed before moving to the next one.

Once a phase is completed, it is not revisited, and any changes or feedback from stakeholders are incorporated in the subsequent phases. This makes it difficult to incorporate new requirements or changes once the development process has moved beyond the requirements gathering phase.

In the waterfall model, iterations are usually limited because any changes or feedback that arise during development can cause delays and make it difficult to meet project timelines and budgets. As a result, the waterfall model is better suited for projects where the requirements are well understood, and there is little chance of significant changes or feedback.

On the other hand, iterative models such as Agile allow for continuous feedback and adjustment, which makes them better suited for projects where the requirements are likely to change or evolve over time.

## *4. What are the three benefits of incremental development, compared to the waterfall model?*

Incremental development offers several benefits compared to the waterfall model. The three key benefits of incremental development are:

- **Flexibility and adaptability -** Incremental development allows for flexibility and adaptability as the requirements of the project can be refined and adjusted as the development progresses. The iterative approach of incremental development enables the team to make adjustments and incorporate changes as necessary, reducing the risk of costly rework later in the development process.

- **Early delivery of working software -** Incremental development involves breaking down the software into smaller, manageable components, allowing the development team to deliver working software early in the development process. This early delivery helps to validate assumptions and identify potential issues earlier, enabling the team to take corrective action promptly.

- **Stakeholder involvement and feedback -** Incremental development encourages stakeholder involvement and feedback throughout the development process. As working software is delivered incrementally, stakeholders can provide feedback and suggest changes, which can be incorporated into subsequent iterations. This promotes a collaborative approach to development, and stakeholders feel more involved in the process, leading to a greater sense of ownership and satisfaction with the end product.

Overall, incremental development is more responsive to change, offers more opportunities for stakeholder involvement, and delivers working software earlier than the waterfall model.

## *5. What are the development stages in reuse-based development?*

Reuse-based development is an approach to software development that emphasizes the reuse of existing software components or modules to create new software systems. There are several stages involved in reuse-based development:

- **Requirements Analysis -** The first stage involves identifying the requirements for the new system. This involves determining what functionality the system should have and what constraints it must operate within.

- **Component Selection -** The second stage involves selecting the software components that will be used in the new system. These components may be existing software modules or components developed specifically for reuse.

- **Component Adaptation -** The third stage involves adapting the selected components to meet the requirements of the new system. This may involve modifying the components, integrating them with other components, or developing new components to fill any gaps.

- **System Integration -** The fourth stage involves integrating the adapted components into the new system. This may involve testing the individual components and then testing the system as a whole to ensure that it operates correctly.

- **Maintenance -** The final stage involves maintaining the system over time. This may involve updating the system to incorporate new components or features, fixing bugs, or making other changes as needed.

Overall, reuse-based development can be an effective approach to software development as it can save time, reduce costs, and improve the quality of the resulting software system. However, it does require careful planning and management to ensure that the selected components are appropriate for the new system and that they are integrated correctly.

## *6. What are the principal requirements engineering activities?*

requirements engineering processes may include four high-level activities. These focus on assessing if the system is useful to the business (feasibility study), discovering requirements (elicitation and analysis), converting these requirements into some standard form (specification), and checking that the requirements actually define the system that the customer wants (validation).

- **Requirements elicitation:** This involves understanding and discovering the needs, goals, constraints, and expectations of the stakeholders who will use the software system. It involves techniques such as interviews, surveys, observations, and brainstorming to gather requirements.
- **Requirements specification:** This activity involves documenting the requirements in a clear and concise manner that can be understood by all stakeholders. The requirements specification may include use cases, user stories, functional and non-functional requirements, and acceptance criteria.
- **Requirements validation:** This activity involves ensuring that the requirements are correct, complete, and meet the needs of the stakeholders. Validation may involve reviewing the requirements with stakeholders, prototyping, simulation, and testing.
- **Requirements document:** This activity involves tracking changes to the requirements throughout the software development lifecycle and ensuring that the requirements are being met. It also involves identifying and managing any risks associated with the requirements and ensuring that they are properly documented and communicated to stakeholders.

## *7. Why is it increasingly irrelevant to distinguish between software development and evolution?*

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance). People think of software development as a creative activity in which a software system is developed from an initial

concept through to a working system. However, they sometimes think of software maintenance as dull and uninteresting. Although the costs of maintenance are often several times the initial development costs, maintenance processes are sometimes considered to be less challenging than original software development.
This distinction between development and maintenance is increasingly irrelevant. Hardly any software systems are completely new systems and it makes much more sense to see development and maintenance as a continuum. Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

- **Agile methodologies:** Agile methodologies, such as Scrum and Kanban, promote a flexible and iterative approach to software development, where the software evolves continuously over time based on feedback from users and stakeholders. In this context, the boundary between development and evolution is blurred, as new features and improvements are added to the software in an ongoing manner.
- **Continuous delivery:** The rise of continuous delivery practices has made it easier and faster to deploy changes to software systems. With continuous delivery, new features and improvements can be delivered to users on an ongoing basis, without the need for major releases or downtime. This means that software is constantly evolving, rather than being developed in discrete phases.
- **DevOps:** The rise of continuous delivery practices has made it easier and faster to deploy changes to software systems. With continuous delivery, new features and improvements can be delivered to users on an ongoing basis, without the need for major releases or downtime. This means that software is constantly evolving, rather than being developed in discrete phases..

## *8. What are the advantages of using incremental development and delivery*

Incremental development and delivery is a software development approach where the development process is broken down into smaller, more manageable iterations, with each iteration resulting in a functional and potentially shippable product. The advantages of using this approach are numerous:

- **Faster delivery:** Incremental development and delivery allows for quicker delivery of functional software. Instead of waiting for the entire project to be completed, customers can start using the software as soon as each iteration is completed.
- **Early and frequent feedback:** Since customers have access to the software early on, they can provide feedback on what is working well and what needs improvement. This allows for changes to be made early in the development process, when they are less costly to implement.
- **Reduced risk**: Incremental development and delivery reduces the risk of project failure by breaking the project down into smaller, more manageable iterations. This allows for early detection and correction of problems.

- **Improved flexibility:** This approach allows for changes to be made at any point in the development process, without disrupting the entire project. This is particularly useful when requirements change, which is common in software development.
- **Improved collaboration:** Incremental development and delivery encourages collaboration among team members, as each iteration requires coordination and cooperation among team members.
- **Cost-effective**: This approach can be more cost-effective than traditional software development approaches, as it allows for changes to be made early in the development process when they are less costly to implement.

## *9. What are the 4 sectors in each loop in Boehm's spiral model?*

- A risk-driven software process framework (the spiral model) was proposed by Boehm (1988). the software process is represented as a spiral, rather than a sequence of activities with some backtracking from one activity to another. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design, and so on. The spiral model combines change avoidance with change tolerance. It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks. Each loop in the spiral is split into four sectors:
- Objective setting Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
- Risk assessment and reduction For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.
- Development and validation After risk evaluation, a development model for the system is chosen. For example, throwaway prototyping may be the best development approach if user interface risks are dominant. If safety risks are the main consideration, development based on formal transformations may be the most appropriate process, and so on. If the main identified risk is sub-system integration, the waterfall model may be the best development model to use.
- Planning The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

## *10. What are the six fundamental best practices in the RUP?*

The practice perspective on the RUP describes good software engineering practices that are recommended for use in systems development. Six fundamental best practices are recommended:

- Develop software iteratively Plan increments of the system based on customer priorities and develop the highest-priority system features early in the development process.
- Manage requirements Explicitly document the customer's requirements and keep track of changes to these requirements. Analyze the impact of changes on the system before accepting them.
- Use component-based architectures Structure the system architecture into components, as discussed earlier in this chapter.
- Visually model software Use graphical UML models to present static and dynamic views of the software.
- Verify software quality Ensure that the software meets the organizational quality standards.
- Control changes to software Manage changes to the software using a change management system and configuration management procedures and tools.