CÂU HỎI ÔN TẬP MÔN CÔNG NGHỆ PHẦN MỀM

Chương 1. Giới thiệu về Công nghệ phần mềm

1. Trình bày các khái niệm cốt lõi trong công nghệ phần mềm:
    a. Công nghệ phần mềm
    b. Khoa học máy tính
    c. Phần mềm chuyên nghiệp
    d. Tiến trình phần mềm
2. Phân biệt giữa công nghệ phần mềm và khoa học máy tính.
3. Phân biệt giữa công nghệ phần mềm và kỹ thuật hệ thống.
4. Trình bày những thách thức mà ngành công nghệ phần mềm đang đối diện.
5. Tại sao phần mềm chuyên nghiệp không chỉ là chương trình?
6. Trình bày khái niệm và đặc trưng của generic software.
7. Trình bày khái niệm và đặc trưng của custom software.
8. So sánh generic software và custom software.
9. Nêu 4 thuộc tính cốt lõi và 4 thuộc tính khác mà phần mềm chuyên nghiệp nên có.
10. Kể tên và trình bày ngắn các loại hệ thống phần mềm.
11. Trình bày stand-alone applications.
12. Trình bày interactive transaction-based applications.
13. Trình bày embedded control systems.
14. Trình bày batch processing systems.
15. Trình bày entertainment systems.
16. Trình bày systems for modeling and simulation.
17. Trình bày data collection systems.
18. Trình bày systems of systems.

Chương 2. Mô hình phát triển phần mềm

1. Trình bày các giai đoạn chính trong software life-cycle.
2. Phân biệt software life-cycle và software process model.
3. Trình bày các kiểu mô hình quy trình phần mềm.
4. Trình bày waterfall model, ưu/nhược điểm, ứng dụng.
5. Trình bày incremental model, ưu/nhược điểm, ứng dụng.
6. Trình bày reuse-oriented model, ưu/nhược điểm, ứng dụng.
7. Giải thích vì sao incremental model phù hợp business systems, không hợp real-time systems.
8. Mô hình nào phù hợp nhất cho thiết kế giao diện người dùng?
9. Thế nào là Plan-driven process? Đối tượng phù hợp?

10. Ưu điểm của reuse-oriented SE?
11. Đề xuất mô hình quy trình phù hợp với các hệ thống:
    - a) ABS trong xe hơi
    - b) Hệ thống kế toán cho đại học
    - c) Hệ thống lập kế hoạch du lịch
12. CASE là gì? Tầm quan trọng?

Chương 3. Kỹ thuật phân tích yêu cầu

1. Yêu cầu phần mềm là gì?
2. Requirements engineering là gì?
3. Phân biệt user vs. system requirements.
4. Phân biệt functional vs. non-functional requirements.
5. Các loại yêu cầu phi chức năng.
6. Product/organizational/external non-functional requirements.
7. Tài liệu SRS là gì? Cấu trúc chuẩn IEEE.
8. Cấu trúc 1 yêu cầu chức năng theo IEEE.
9. Mô tả yêu cầu chức năng tra cứu điểm.
10. Tiến trình xác định và phân tích yêu cầu.
11. Khó khăn khi tìm hiểu stakeholder requirements?
12. Phương pháp khám phá yêu cầu.
13. Requirements validation là gì? Các kỹ thuật?
14. Tại sao phải quản lý yêu cầu?

Chương 4. Mô hình hóa hệ thống và thiết kế kiến trúc

4.1. Mô hình hóa hệ thống

1. System modeling là gì? Việc mô hình hóa hệ thống mang lại lợi ích gì trong quy trình phát triển phần mềm?
2. Trình bày và so sánh bốn góc nhìn (perspectives) chính trong system modeling: external, interaction, structural và behavioral.
3. UML là gì? Kể tên và nêu ngắn gọn mục đích sử dụng của 5 loại sơ đồ UML phổ biến được sử dụng trong system modeling.
4. Context diagram dùng để làm gì? Trình bày vai trò của nó trong việc xác định ranh giới hệ thống.
5. Mô hình Use Case được sử dụng như thế nào trong việc xác định yêu cầu phần mềm? Một Use Case bao gồm những thành phần nào?
6. Sequence diagram thể hiện điều gì trong mô hình hóa hệ thống? Khi nào nên sử dụng sơ đồ này?

7. Trình bày khái niệm và mục tiêu của mô hình trạng thái (State Machine Model). Đưa ví dụ minh họa một hệ thống thực tế.
8. Thế nào là Generalization trong sơ đồ lớp UML? So sánh với Aggregation và Composition.
9. Mô tả khái niệm Model-Driven Engineering (MDE) và trình bày các loại mô hình chính trong MDA: CIM, PIM, PSM.
10. Phân tích sự khác biệt giữa mô hình hướng sự kiện (event-driven modeling) và mô hình hướng dữ liệu (data-driven modeling). Ứng dụng cụ thể của từng loại.

4.2. Thiết kế kiến trúc phần mềm

11. Trình bày sự khác biệt giữa kiến trúc phần mềm (software architecture) và thiết kế phần mềm (software design). Vai trò của mỗi giai đoạn trong quy trình phát triển phần mềm là gì?
12. Architectural design là gì? Tại sao cần thực hiện thiết kế kiến trúc trước các bước thiết kế chi tiết khác?
13. Kiến trúc phần mềm ảnh hưởng như thế nào đến các thuộc tính phi chức năng của hệ thống như: hiệu năng, bảo mật, an toàn và bảo trì? Đưa ví dụ minh họa.
14. Mô hình kiến trúc 4+1 gồm những view nào? Nêu ngắn gọn vai trò của mỗi view trong việc thiết kế và trình bày kiến trúc hệ thống.
15. Trình bày đặc điểm của mô hình kiến trúc Layered Architecture. Ưu điểm và hạn chế của mô hình này là gì? Trong trường hợp nào nên sử dụng mô hình này?
16. So sánh các mô hình kiến trúc: Client-Server, Repository, MVC và Pipe-and-Filter. Đặc điểm chính và ứng dụng phù hợp của từng mô hình là gì?
17. Trình bày các bước chính trong quá trình thiết kế lớp trong phần mềm hướng đối tượng. Sơ đồ lớp UML thể hiện thông tin gì về hệ thống?
18. Thế nào là quan hệ kế thừa (Generalization) và quan hệ thành phần (Composition) trong sơ đồ lớp UML? Cho ví dụ minh họa.
19. Mẫu thiết kế (Design Pattern) là gì? Trình bày ví dụ mẫu MVC hoặc Singleton và nêu rõ khi nào nên sử dụng mẫu đó.
20. Thiết kế giao diện người dùng cần tuân thủ những nguyên tắc nào để đảm bảo tính hiệu quả và thân thiện? Nêu ít nhất 3 nguyên tắc quan trọng.

Chương 5. Thiết kế và cài đặt phần mềm

21. Trình bày mối quan hệ giữa hoạt động thiết kế phần mềm và cài đặt phần mềm. Vì sao nói hai hoạt động này thường được thực hiện xen kẽ nhau?
22. Phân tích vai trò của mô hình UML trong thiết kế hướng đối tượng. Những loại mô hình UML nào thường được sử dụng để mô tả thiết kế hệ thống?
23. Trình bày các bước chính trong quy trình thiết kế hướng đối tượng. Đâu là vai trò của việc xác định giao tiếp (interface) của các đối tượng?

24. Mẫu thiết kế (design pattern) là gì? Trình bày các thành phần cấu thành nên một mẫu thiết kế. Cho ví dụ về mẫu Observer và mục đích sử dụng của nó.
25. So sánh ưu và nhược điểm của các kiểu kiến trúc phần mềm sau: Client/Server, Kiến trúc phân tầng (Layered), N-Tier, và Hướng dịch vụ (SOA).
26. Giải thích khái niệm và lợi ích của việc tái sử dụng phần mềm ở các mức: trừu tượng, đối tượng, thành phần, và hệ thống.
27. Cấu hình phần mềm (Configuration Management) là gì? Trình bày ba hoạt động chính trong quản lý cấu hình phần mềm.
28. Trình bày khái niệm phát triển phần mềm host-target. Phân biệt giữa nền tảng phát triển và nền tảng thực thi.
29. Open source development là gì? Nêu các lợi ích và thách thức khi phát triển phần mềm theo mô hình mã nguồn mở.
30. Trình bày các mô hình cấp phép phần mềm mã nguồn mở như: GPL, LGPL và BSD. Điểm khác biệt chính giữa các mô hình này là gì?

Chương 6. Kiểm thử và định giá phần mềm

1. Kiểm thử phần mềm là gì?
2. Mục tiêu kiểm thử?
3. Validation vs. Verification?
4. Mô hình quy trình kiểm thử.
5. Các giai đoạn kiểm thử.
6. Development testing: ai thực hiện?
7. Release testing: ai thực hiện?
8. User testing: ai thực hiện?
9. Unit/component/system testing: đặc điểm?
10. Automated testing là gì? Công cụ?
11. Test case là gì? Cấu trúc?
12. Black-box vs. white-box testing?
13. Alpha, beta, acceptance testing: khái niệm và mục đích.
14. Kiểm thử có thể đảm bảo không còn lỗi không?

CÂU HỎI ÔN TẬP MÔN CÔNG NGHỆ PHẦN MỀM

Chương 1. Giới thiệu về Công nghệ phần mềm

1. Trình bày các khái niệm cốt lõi trong công nghệ phần mềm:

a. Công nghệ phần mềm: (Slide 10 – Chương 1) (Software Engineering) or (Trang 15 – Giáo trình)

VIE:

- Công nghệ phần mềm là một lĩnh vực nghiên cứu của tin học nhằm đề xuất các nguyên lý, phương pháp, công cụ, cách tiếp cận phục vụ cho việc thiết kế, cài đặt các sản phẩm phần mềm đạt được đầy đủ các yêu cầu về chất lượng phần mềm.
- Công nghệ phần mềm là một tập các giai đoạn, thường được chia thành 7 giai đoạn chính là: Khảo sát, phân tích, thiết kế, xây dựng, kiểm thử, triển khai và bảo trì.
- Mỗi một giai đoạn lại có các quy tắc riêng, có các tài liệu đi kèm để hoàn thiện các thao tác của giai đoạn đó.
- Công nghệ phần mềm đề cập tới các hoạt động xây dựng và đưa ra một phần mềm hữu ích.
- Do vậy, các giai đoạn trong công nghệ phần mềm nếu phối hợp tốt sẽ cho ra một sản phẩm hoàn thiện, nếu chỉ một giai đoạn thực hiện chưa đúng yêu cầu sẽ có một phần mềm có chất lượng thấp và không đáp ứng được nhu cầu của người dùng.

ENG:

- Software engineering is a field of computer science that proposes principles, methods, tools, and approaches for designing and implementing software products that fully meet software quality requirements.
- Software engineering is a set of phases, typically divided into seven main stages: investigation, analysis, design, development, testing, deployment, and maintenance.
- Each stage has its own rules and is accompanied by specific documentation to complete the tasks of that phase.
- Software engineering refers to the activities involved in building and delivering a useful software product.
- Therefore, if the phases in software engineering are well-coordinated, they will result in a complete and high-quality product. Conversely, if even one phase is not properly executed, it can lead to low-quality software that fails to meet user needs.

b. Khoa học máy tính (Computer Science)

VIE:

- Khoa học máy tính là lĩnh vực nghiên cứu lý thuyết và ứng dụng về tính toán, máy tính, và các hệ thống máy tính.
- Nó tập trung vào các nền tảng lý thuyết của thông tin và tính toán, cũng như cách chúng được triển khai và ứng dụng trong các hệ thống máy tính.
- Các lĩnh vực con bao gồm thuật toán, cấu trúc dữ liệu, trí tuệ nhân tạo, đồ họa máy tính, mạng máy tính, hệ điều hành và ngôn ngữ lập trình.
- Khoa học máy tính cung cấp nền tảng kiến thức cho việc phát triển phần mềm, nhưng không phải là toàn bộ quá trình phát triển phần mềm.

ENG:

- Computer Science is the study of theoretical foundations of information and computation, and of practical techniques for their implementation and application in computer systems.
- It focuses on the fundamental principles of computation, algorithms, data structures, artificial intelligence, computer graphics, computer networks, operating systems, and programming languages.
- Computer science provides the theoretical basis for software development, but it is not the entire process of software development.

c. Phần mềm chuyên nghiệp (Professional Software)

VIE:

- Phần mềm chuyên nghiệp là phần mềm được phát triển bởi các kỹ sư phần mềm hoặc nhóm phát triển chuyên nghiệp, đáp ứng các tiêu chuẩn chất lượng cao về chức năng, hiệu suất, độ tin cậy, bảo mật và khả năng bảo trì.
- Khác với phần mềm tự phát hoặc cá nhân, phần mềm chuyên nghiệp được thiết kế để sử dụng trong môi trường thực tế.
- Thường phục vụ một lượng lớn người dùng hoặc các mục đích kinh doanh, đòi hỏi quy trình phát triển có cấu trúc, kiểm thử nghiêm ngặt và tài liệu đầy đủ.

ENG:

- Professional software refers to software developed by professional software engineers or development teams that meets high quality standards regarding functionality, performance, reliability, security, and maintainability.
- Unlike self-developed or personal software, professional software is designed for real-world environments.
- Often serving a large number of users or business purposes, requiring structured development processes, rigorous testing, and comprehensive documentation.

d. Tiến trình phần mềm (Software Process) (Slide 15 – Chương 1)

- A software process is a set of activities, methods, and practices used to develop and maintain software.
- It describes how a software project is carried out from its inception to completion and delivery.
- Typical stages of a software process include:
    - Specification: where customers and engineers define the software that is to be produced and the constraints on its operation.
    - Design and implementation: where the software is designed and programmed.
    - Validation: where the software is checked to ensure that it is what the customer requires.
    - Evolution/maintenance: where the software is modified to reflect changing customer and market requirements.
- Common software process models include the waterfall model, Agile models (e.g., Scrum, Kanban), and the spiral model.

2. Phân biệt giữa công nghệ phần mềm và khoa học máy tính.

(The difference between software engineering and computer science)**(Slide 13 – Chương 1)**

3. Phân biệt giữa công nghệ phần mềm và kỹ thuật hệ thống.

(The difference between software engineering and system engineering) **(Slide 14 – Chương 1)**

4. Trình bày những thách thức mà ngành công nghệ phần mềm đang đối diện.

(Present the challenges currently facing the software engineering industry)

VIE:

- Tính phức tạp ngày càng tăng của phần mềm
- Đảm bảo chất lượng phần mềm
- An toàn và bảo mật thông tin
- Thay đổi nhanh chóng của công nghệ
- Quản lý yêu cầu và sự thay đổi của khách hàng

- Thiếu hụt nhân lực chất lượng cao

ENG:

- **Increasing complexity of software systems:** Modern software must meet complex requirements, support cross-platform integration, and ensure performance, security, and scalability.
- **Ensuring software quality:** Comprehensive testing, early bug detection, and maintaining quality in the face of continuous changes remain difficult tasks.
- **Security and data protection:** Threats from hackers, malware, and data breaches force software engineers to implement security measures from the design phase.
- **Rapid technological changes:** With constant innovation (AI, Cloud, Blockchain, etc.), professionals must continually learn and adapt to stay relevant.
- **Managing evolving user requirements:** Users frequently change their needs, making it difficult to manage project scope and timelines effectively.
- **Shortage of skilled professionals:** Recruiting and training competent developers with both technical and soft skills is a persistent challenge.

5. Tại sao phần mềm chuyên nghiệp không chỉ là chương trình?

(Why is professional software more than just a program?)

- Professional software is more than just an executable program. It is a **complete system** that includes various components and qualities to ensure it is **usable, reliable, and maintainable** in real-world environments. Specifically:
    - **Technical documentation:** Includes design documents, user manuals, and maintenance guides to support developers and end-users.
    - **Maintainability and upgradabilit:** Professional software is designed for easy error correction, updates, and adaptation to changing environments.
    - **Quality and reliability:** Thoroughly tested, the software must perform consistently and handle failures safely.
    - **Scalability**: It must be able to support large numbers of users or handle high data loads effectively.
    - **Support and maintenance services**: Professional software is accompanied by technical support, regular patches, and updates – unlike personal or experimental programs.

6. Trình bày khái niệm và đặc trưng của generic software.

(Define and describe the characteristics of generic software)

VIE:

- Generic software (phần mềm đại trà/phổ biến) là loại phần mềm được phát triển không dành riêng cho một khách hàng cụ thể, mà hướng tới một nhóm người dùng rộng lớn với các nhu cầu tương tự. Phần mềm này thường được đóng gói sẵn, bán thương mại hoặc cung cấp miễn phí, và người dùng có thể cài đặt và sử dụng mà không cần điều chỉnh lớn. Ví dụ: Microsoft Word, Google Chrome, Adobe Photoshop, ...
- Đặc trưng của generic software:
    - Phát triển dựa trên thị trường
    - Khả năng tái sử dụng cao
    - Chi phí thấp hơn so với phần mềm đặt hàng
    - Cập nhật định kỳ
    - Không linh hoạt bằng phần mềm theo yêu cầu

ENG:

- Generic software refers to software developed for a broad market rather than for a specific customer. It is intended to meet the common needs of many users and is typically packaged for commercial sale or free distribution. Users can install and use it with little or no customization.

- Characteristics of generic software:
    - Market-driven development
    - High reusability
    - Lower cost compared to custom software
    - Regular updates
    - Less flexibility than bespoke software

7. Trình bày khái niệm và đặc trưng của custom software.

(Define and describe the characteristics of custom software)

VIE:

- Custom software (phần mềm đặt hàng / phần mềm theo yêu cầu) là loại phần mềm được thiết kế và phát triển dành riêng cho một tổ chức hoặc cá nhân cụ thể, để đáp ứng nhu cầu đặc thù của họ. Nó không bán rộng rãi trên thị trường mà thường được thực hiện thông qua hợp đồng giữa khách hàng và đơn vị phát triển phần mềm. Ví dụ: Phần mềm quản lý bệnh viện cho một bệnh viện cụ thể, phần mềm ERP nội bộ cho một công ty sản xuất,...
- Đặc trưng của custom software:
    - Phù hợp với nhu cầu riêng biệt
    - Tính linh hoạt cao
    - Chi phí phát triển cao hơn
    - Thời gian phát triển dài hơn
    - Bảo trì và hỗ trợ riêng biệt

ENG:

- Custom software is software developed specifically for a particular organization or individual to meet their unique requirements. Unlike generic software, it is not distributed widely but is usually built under contract with a specific client. Examples: A hospital management system for a specific hospital, or an internal ERP system for a manufacturing company.
- Characteristics of custom software:
    - Tailored to specific needs
    - High flexibility
    - Higher development cost
    - Longer development time
    - Dedicated support and maintenance

8. So sánh generic software và custom software.

Criteria	Generic Software	Custom Software
Target Users	Designed for a broad group of users with similar needs	Designed for a specific client or organization with unique requirements
Development Goal	Based on general market demands	Based on specific client needs and requirements
Customization	Limited customization options	Highly customizable to fit specific workflows
Development Cost	Lower cost due to mass production and reuse	Higher cost due to unique development process
Deployment Time	Quick – ready to use after installation	Longer – requires requirement analysis, design, development, and testing
Updates and Maintenance	Maintained by the software vendor	Maintained by the development team or under a dedicated support contract

Examples	Microsoft Office, Adobe Photoshop, Google Chrome	Hospital Management System, Internal ERP for a manufacturing company
Scalability	Designed for general scalability, may not fit specific business growth scenarios	Scalable according to the client's needs and future business plans
Ownership	Typically owned and controlled by the vendor	Typically fully owned or licensed exclusively by the client
Flexibility	Less flexible – users must adapt to existing features	Very flexible – software is adapted to users' workflows and business logic

9. Nêu 4 thuộc tính cốt lõi và 4 thuộc tính khác mà phần mềm chuyên nghiệp nên có.

(Name 4 core attributes and 4 other important attributes that professional software should have)

* 4 core attributes of professional software:

- **Correctness:** The software performs exactly as specified and meets user requirements.
- **Reliability:** The software operates stably and can handle unexpected conditions with minimal failure.
- **Efficiency:** The software uses system resources (memory, CPU, etc.) efficiently and responds quickly.
- **Maintainability:** It is easy to fix bugs, improve performance, and update the software in the future.

* 4 other important attributes:

- **Usability:** The interface is user-friendly, intuitive, and easy to learn and use.
- **Portability:** The software can run on various platforms or operating systems.
- **Reusability:** Components of the software can be reused in other systems or projects.
- **Security:** The software ensures data protection and prevents unauthorized access.

10. Kể tên và trình bày ngắn các loại hệ thống phần mềm.

(Name and briefly describe the types of software systems)

Here are common types of software systems:

- **Information Systems:** Manage and process data for businesses or organizations (e.g., HR systems, accounting software).
- **Embedded Systems:** Integrated into hardware to control devices (e.g., in cars, washing machines, robots).
- **Real-time Systems:** Respond quickly and accurately to real-time inputs (e.g., air traffic control, medical devices).
- **Engineering/Scientific Software:** Used for analysis, simulation, or scientific calculations (e.g., CAD tools, Matlab).
- **Web-based / Distributed Systems:** Operate over the internet, distribute data and processing (e.g., cloud apps, web platforms).
- **AI-based Systems:** Can learn, analyze, and make decisions (e.g., chatbots, recommendation systems, image recognition).

11. Trình bày stand-alone applications.

- **VIE:** Stand-alone applications là những phần mềm chạy độc lập trên máy tính cá nhân mà không cần kết nối mạng. Chúng thực hiện đầy đủ chức năng trong phạm vi cục bộ. Ví dụ: Microsoft Word, Photoshop.
- **ENG:** Stand-alone applications are software programs that run independently on a personal computer without requiring a network connection. They perform complete functions locally. Example: Microsoft Word, Photoshop.

12. Trình bày interactive transaction-based applications.

- **VIE:** Là các ứng dụng có giao diện tương tác với người dùng và xử lý các giao dịch, thường thấy

trong hệ thống ngân hàng, thương mại điện tử. Chúng yêu cầu phản hồi nhanh và đảm bảo tính nhất quán của dữ liệu. Ví dụ: Hệ thống ngân hàng trực tuyến, đặt vé máy bay.

- **ENG:** These are applications with interactive user interfaces that handle transactions, commonly found in banking and e-commerce systems. They require quick responses and data consistency. Example: Online banking, airline reservation systems.

13. Trình bày embedded control systems.

- **VIE:** Là các hệ thống phần mềm được nhúng vào phần cứng, có nhiệm vụ điều khiển thiết. Chúng có yêu cầu cao về độ tin cậy và thời gian thực. Ví dụ: Hệ thống ABS trong ô tô, điều khiển trong lò vi sóng, máy giặt
- **ENG:** Embedded control systems are software embedded in hardware devices to control their operations. They have strict reliability and real-time requirements. Example: ABS systems in cars, microwave controllers, washing machines.

14. Trình bày batch processing systems.

- **VIE:** Là hệ thống xử lý hàng loạt, thực hiện các công việc lớn không yêu cầu tương tác trực tiếp. Thường chạy vào ban đêm để tiết kiệm tài nguyên. Ví dụ: Xử lý bảng lương, thống kê dữ liệu lớn.
- **ENG:** Batch processing systems process large volumes of data without real-time user interaction. They often run overnight to optimize resource usage. Example: Payroll processing, large-scale data reporting.

15. Trình bày entertainment systems.

- **VIE:** Hệ thống giải trí bao gồm phần mềm phục vụ nhu cầu giải trí như trò chơi, truyền hình, nhạc số. Chúng ưu tiên trải nghiệm người dùng, đồ họa, âm thanh và độ trễ thấp. Ví dụ: Game console, ứng dụng streaming như Netflix.
- **ENG:** Entertainment systems are software for leisure activities such as games, TV, or digital music. They focus on user experience, graphics, sound, and low-latency performance. Example: Game consoles, streaming apps like Netflix.

16. Trình bày systems for modeling and simulation.

- **VIE:** Là hệ thống dùng để mô phỏng các hiện tượng thực tế, thường được sử dụng trong nghiên cứu khoa học và kỹ thuật. Chúng yêu cầu tính toán phức tạp và độ chính xác cao. Ví dụ: Mô phỏng bay, mô phỏng thời tiết.
- **ENG:** These systems are used to simulate real-world phenomena, often applied in scientific and engineering research. They require complex computation and high accuracy. Example: Flight simulators, weather simulation systems.

17. Trình bày data collection systems.

- **VIE:** Là các hệ thống thu thập và xử lý dữ liệu từ cảm biến hoặc nguồn bên ngoài, ví dụ: hệ thống theo dõi sức khỏe, giám sát môi trường. Yêu cầu lưu trữ và xử lý dữ liệu theo thời gian thực.
- **ENG:** These systems gather and process data from sensors or external sources, such as health monitoring or environmental surveillance systems. They often require real-time data handling and storage.

18. Trình bày systems of systems.

- **VIE:** Là tập hợp nhiều hệ thống độc lập phối hợp với nhau để đạt mục tiêu chung. Ví dụ: hệ thống điều khiển giao thông thông minh tích hợp GPS, camera, đèn giao thông. Chúng phức tạp và cần quản lý tương tác chặt chẽ.
- **ENG:** Systems of systems are collections of independent systems working together to achieve a common goal. For example, intelligent traffic control integrates GPS, cameras, and traffic lights. They are complex and require coordinated management.

Chương 2. Mô hình phát triển phần mềm

1. Trình bày các giai đoạn chính trong software life-cycle.

The software life cycle includes 7 main stages:

Stage 1: Requirements Definition Stage

- This is the initial step to define the problem or topic.
- The design team surveys the current system and interviews users to understand real-world operations.
- Software roles are identified, workload is estimated, and task schedules are created.
- Outputs: Requirements document, basic forms, and related regulations.

Stage 2: System Analysis Stage

- Use formal or informal specification languages to detail the requirements.
- Structural analysis: Use DFD (Data Flow Diagram).
- Object-oriented analysis: Use UML diagrams.

Stage 3: System Design Stage

- Design is based on two main aspects:
- Data modeling: Design tables, files, etc., for future storage.
- Program modeling: Design interfaces and input forms; commonly using pre-designed HTML templates or Windows GUI templates.

Stage 4: Implementation / Coding Stage

- Developers write code using suitable programming languages to implement the specified requirements.
- Apply layered architectures or partial code generation to increase efficiency and speed.

Stage 5: Testing Stage

- Test the program using sample data.
- Objective: Detect and fix bugs, verify correctness, stability, efficiency, and user support.

Stage 6: Deployment Stage

- Launch the software into the real environment.
- Includes: installation, user training, and Beta testing, where end-users test the software to detect and report bugs.

Stage 7: Maintenance Stage

- Fix bugs that appear after deployment.
- Adapt the software to meet new user requirements or improve performance over time.

2. Phân biệt software life-cycle và software process model.

VIE:

Tiêu chí	Software Lifecycle	Software Process Model
Khái niệm	Là tập hợp **các giai đoạn phát triển** của phần mềm, từ khi hình thành ý tưởng đến khi không còn sử dụng.	Là cách **tổ chức và triển khai** các hoạt động trong chu kỳ sống phần mềm.
Mục tiêu	Mô tả các **giai đoạn cần có** trong quá trình phát triển phần mềm.	Mô tả **quy trình cụ thể** để thực hiện các giai đoạn đó.
Tính chất	Mang tính **khái quát**, chỉ ra những gì cần làm.	Mang tính **chi tiết**, chỉ rõ cách thực hiện, trình tự và kỹ thuật sử dụng.
Ví dụ	Các giai đoạn: xác định yêu cầu, phân tích, thiết kế, cài đặt, kiểm thử, triển khai, bảo trì.	Các mô hình quy trình như: Waterfall (thác nước), Agile, Spiral, V-model, Iterative model.
Ứng dụng	Áp dụng cho **mọi loại phần mềm**, ở cấp độ tổng quan.	Lựa chọn mô hình quy trình phù hợp tùy vào **dự án cụ thể**.

ENG:

Criteria	Software Lifecycle	Software Process Model
Definition	A set of phases that a software product goes through from conception to retirement.	A structured approach or methodology to implement those phases.
Purpose	Describes what stages are involved in software development.	Describes how to carry out those stages effectively.
Nature	General and conceptual, defines what needs to be done.	More detailed and procedural, focuses on execution methods and sequence.
Examples	Phases: requirements, analysis, design, implementation, testing, deployment, maintenance.	Models like: Waterfall, Agile, Spiral, V-model, Iterative model.
Application	Applied to all software projects, regardless of method.	Chosen based on project-specific needs and constraints.

Tóm tắt ngắn gọn:
Software life cycle = "Những gì cần làm" (What to do).
Software process model = "Làm như thế nào" (How to do it).

3. Trình bày các kiểu mô hình quy trình phần mềm.
(Present types of software process models)

ENG:

There are several software process models, each suited for different types of projects. The most common ones include:

1. Waterfall Model

- Sequential process through stages: Requirements → Analysis → Design → Implementation → Testing → Deployment → Maintenance.
- Pros: Simple, easy to manage.
- Cons: Difficult to go back to previous phases.

2. Incremental Model

- The system is developed in small parts (increments).
- Each part can be deployed and used immediately.
- Suitable for projects requiring early user feedback.

3. Iterative Model

- Software is built through multiple iterations.
- Each iteration includes full development cycle: requirements → design → coding → testing.
- Helps improve product through continuous refinement.

4. Spiral Model

- Combines iterative development with risk analysis.
- Each loop includes: set objectives → risk assessment → development → evaluation.
- Suitable for large, high-risk projects.

5. Agile Model

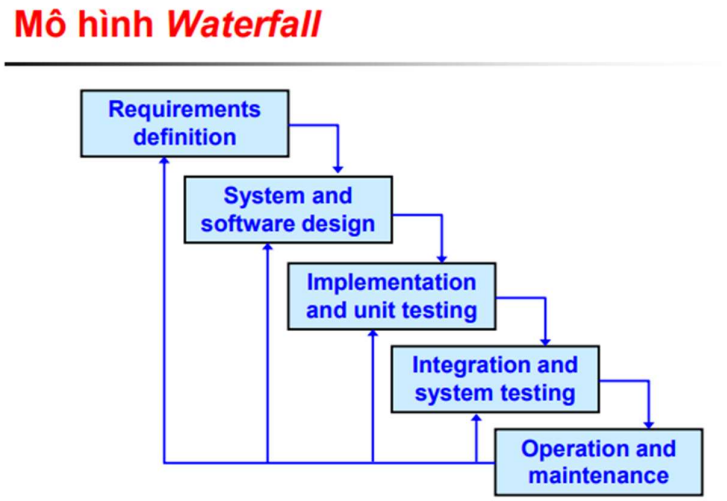
- Focuses on flexibility and quick response to changes.
- Work is divided into short cycles called Sprints (e.g., in Scrum).
- Emphasizes customer collaboration and rapid delivery.

6. V-Model (Verification and Validation Model)

- Extension of Waterfall Model, emphasizes parallel development and testing.
- Every development phase has a corresponding testing phase.
- Suitable for critical systems requiring high reliability.

4. Trình bày waterfall model, ưu/nhược điểm, ứng dụng.

(Describe the Waterfall Model, its advantages/disadvantages, and applications)



The **Waterfall Model** is a sequential software development model in which the process is divided into distinct phases:

Phase 1: Requirements Analysis: The goal is to gather all functional and non-functional requirements from customers and end-users. The output is the Software Requirements Specification (SRS) document.

- Key activities: customer interviews, surveys, requirement documentation.
- Deliverable: SRS document.

Phase 2: System and Software Design: Based on the SRS, the development team designs the overall architecture, including software modules, database design, and user interface.

- High-level design: defines system architecture and main components.
- Low-level design: details algorithms, data structures, and user interactions.

Phase 3: Implementation and Unit Testing: Developers write the source code for each module based on the design. Each module is then tested individually through unit testing.

- Each component is coded and tested independently.
- Testing tools such as JUnit, PyTest, etc., may be used.

Phase 4: Integration and System Testing: All modules are integrated to form the complete system. Testing is performed to verify interactions and overall functionality.

- Detects issues with inter-module communication.
- Ensures the system meets the original specifications.

Phase 5: Operation and Maintenance: The software is deployed in a real environment. Errors are fixed, and enhancements or updates are applied as needed.

- Corrective maintenance: fix bugs.
- Adaptive maintenance: adjust to changes in the environment.
- Perfective maintenance: improve performance or add features.

Each phase must be completed before moving to the next one, and there is no turning back to previous phases.

Advantages:

- Easy to understand and manage due to its linear structure.
- Suitable for projects with well-defined and fixed requirements.
- Progress and documentation are easily tracked.

Disadvantages:

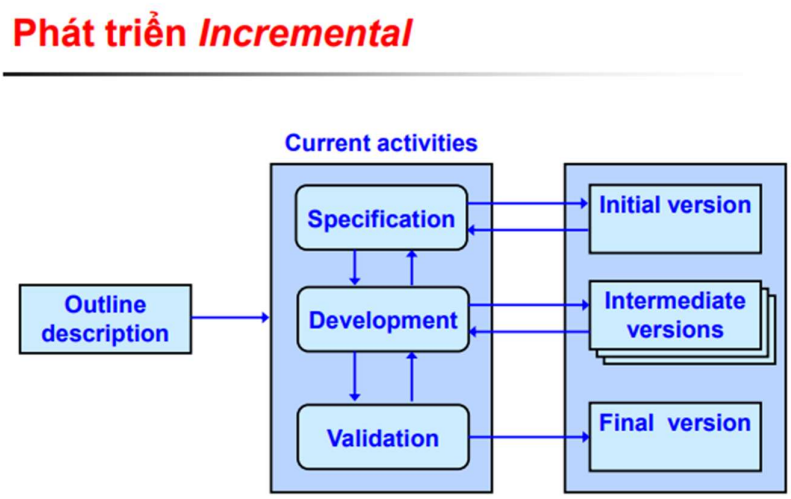
- Inflexible, hard to adapt to changes once development has started.
- Poor response to customer feedback or requirement changes.

- Errors in early phases may go undetected until later stages.

Applications:

- Suitable for small projects with clear, stable requirements.
- Commonly used in embedded systems, control systems, or projects with strict process requirements.

5. Trình bày incremental model, ưu/nhược điểm, ứng dụng.



VIE:

Mô hình tăng dần là một phương pháp phát triển phần mềm trong đó hệ thống được xây dựng thông qua nhiều **phiên bản nhỏ (increments)**. Mỗi phiên bản là một phần chức năng đầy đủ, được phát triển, kiểm thử và triển khai độc lập, sau đó tích hợp vào hệ thống tổng thể.

ENG:

The Incremental Model is a software development approach where the system is built and delivered in **small functional parts (increments)**. Each increment is a complete sub-functionality that is developed, tested, and deployed independently, then integrated into the overall system.

The process is iterative, and each cycle involves:

1. **Requirement analysis:** In each iteration, the development team identifies **specific features** to be implemented. These are selected from the full requirement list based on priority.
    - Criteria: implementable, high-priority, and loosely coupled.
    - Output: a feature list for the current increment.
2. **System Design of that increment:** Architectural, interface, and technical designs are made for the selected features. The design should **support smooth integration** with previous versions.
    - Includes: data flow diagrams, UI/UX mockups, relevant database structures.
    - Ensures reusability and scalability for future increments.
3. **Implementation and unit testing:** Developers code and individually test each small module of the increment's features.
    - Code must be **modular and testable in isolation**.
4. **Integration and system testing:** The new increment is integrated into the existing system. System-wide testing is performed to verify that **previous functionality is not broken**.
    - Includes: regression testing, integration testing, functional testing.
    - Ensures system consistency after the new addition.
5. **Deployment:** After successful testing, the increment is deployed to users. User feedback is collected to improve the system or guide the next increment.
    - Can be released internally (beta) or publicly.
    - Feedback helps refine future development.

Initially, a basic core system is developed. More advanced features are added in subsequent increments.

Advantages:

- Early feedback: Users can interact with early increments and provide feedback.

- Reduced risk: Problems are detected earlier in the development cycle.
- Manageable parts: The project is broken down into manageable components.
- Flexible to changes: More adaptable than the Waterfall Model.

Disadvantages:

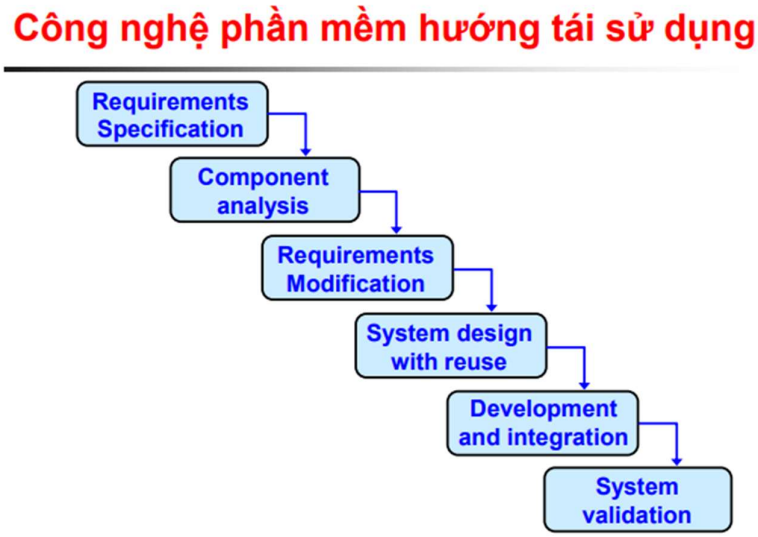
- Requires careful planning: Effective integration of increments needs good planning.
- Complex architecture: The overall design must accommodate future expansions.
- Time-consuming integration: Frequent integration may increase complexity and time.

Applications:

- Suitable for medium to large projects with partially defined or evolving requirements.
- Often used in commercial software, enterprise applications, web-based systems, and projects that require phased releases.

6. **Trình bày reuse-oriented model, ưu/nhược điểm, ứng dụng** (Công nghệ phần mềm hướng tái sử dụng)

   (Present the reuse-oriented model, its advantages/disadvantages, and applications)



The **reuse-oriented software development** model is an approach in which new systems are built by systematically integrating existing software components. These components may include libraries, modules, web services, or pre-packaged frameworks organized in platforms such as .NET or J2EE.

Main phases of the process include:

- **Component analysis**: Searching for and evaluating available components that match the requirements.
- **Requirement modification**: Adapting the system requirements to fit the capabilities of reusable components.
- **System design with reuse**: Designing the architecture based on the integration of available components.
- **Development and integration**: Developing any missing parts, integrating, and testing the full system.

Advantages:

- Reduces development time and cost.
- Increases reliability through the use of tested components.
- Makes systems easier to extend and maintain.

Disadvantages:

- Dependence on external components.
- May require adjusting requirements to fit available components.
- Potential difficulty integrating components from diverse sources.

Applications:

- Common approach in building modern business systems.
- Reuse of web services developed under service standards, accessible remotely.
- Use of component-based platforms like .NET or J2EE that bundle reusable objects and modules.

VIE:

Mô hình phát triển phần mềm dựa trên tái sử dụng (reuse-oriented model) là mô hình phát triển trong đó hệ thống mới được tích hợp từ các thành phần phần mềm đã có, thông qua việc tái sử dụng có hệ thống. Các thành phần này có thể là thư viện, module, dịch vụ web, hoặc các gói phần mềm được đóng gói theo cấu trúc như .NET hoặc J2EE.

Các giai đoạn chính của quy trình bao gồm:

- Phân tích thành phần: Tìm kiếm và đánh giá các thành phần hiện có phù hợp với yêu cầu.
- Thay đổi các yêu cầu: Điều chỉnh yêu cầu hệ thống nếu cần, để phù hợp với khả năng của các thành phần tái sử dụng.
- Thiết kế hệ thống bằng cách tái sử dụng: Xây dựng kiến trúc hệ thống dựa trên việc kết hợp các thành phần có sẵn.
- Phát triển và tích hợp: Hoàn thiện các phần còn thiếu, tích hợp và kiểm thử hệ thống tổng thể.

7. Giải thích vì sao incremental model phù hợp business systems, không hợp real-time systems.

The **Incremental Model** is suitable for **business systems** because:

- Business systems typically have **evolving requirements** and need to adapt to user feedback or market changes. The model allows for **partial feature releases**, enabling early user interaction and iterative improvement.
- Dividing the system into manageable parts helps **reduce risks**, improve maintainability, and optimize development resources in business environments.
- There's no need to build the entire system at once; essential features like user management, transactions, and reporting can be deployed early, while advanced functionalities are added incrementally.

On the other hand, this model is **not suitable for real-time systems** because:

- Real-time systems require **precise and immediate responses**, where **all system components must be tightly integrated** and function consistently.
- Incremental integration can cause **timing issues or loss of synchronization**, which is unacceptable in environments such as flight control systems, medical devices, or embedded systems.
- Due to their strict technical demands, real-time systems need a **complete and stable architecture from the beginning**, which the incremental model cannot ensure.

8. Mô hình nào phù hợp nhất cho thiết kế giao diện người dùng? (Slide 40 – Chuong 2)

VIE: Mô hình phát triển phần mềm phù hợp nhất cho thiết kế giao diện người dùng là mô hình Prototyping (mô hình tạo mẫu). Lý do là vì giao diện người dùng (UI) cần được thiết kế dựa trên phản hồi liên tục từ người dùng thực tế. Mô hình Prototyping cho phép nhà phát triển nhanh chóng xây dựng các mẫu thử (prototype) của giao diện, trình bày cho người dùng trải nghiệm và sau đó cải tiến dựa trên góp ý. Quá trình này giúp đảm bảo giao diện dễ sử dụng, trực quan và đáp ứng đúng nhu cầu của người dùng.

ENG: The most suitable software development model for user interface (UI) design is the Prototyping model. This is because UI design relies heavily on continuous user feedback. The Prototyping model allows developers to quickly build mockups (prototypes) of the interface, present them to users for testing, and then refine the design based on their input. This iterative process helps ensure the interface is user-friendly, intuitive, and meets real user needs.

9. Thế nào là Plan-driven process? Đối tượng phù hợp?

VIE:

Plan-driven process (quy trình phát triển dựa trên kế hoạch) là một mô hình phát triển phần mềm trong đó toàn bộ quá trình được lập kế hoạch trước một cách chi tiết. Các giai đoạn như phân tích yêu cầu, thiết kế, cài đặt, kiểm thử và bảo trì được xác định rõ ràng từ đầu và thực hiện tuần tự.

Plan-driven process phù hợp với các dự án có yêu cầu ổn định, rõ ràng ngay từ đầu và ít thay đổi trong quá trình phát triển. Đối tượng phù hợp gồm:

- Các hệ thống lớn, có tính phức tạp cao như hệ thống ngân hàng, quốc phòng.
- Các tổ chức yêu cầu quy trình nghiêm ngặt, có tài liệu đầy đủ và khả năng kiểm soát tốt.
- Dự án phát triển phần mềm theo hợp đồng cố định, cần lập kế hoạch chi tiết từ sớm.

ENG:

Answer:

A **Plan-driven process** is a software development model in which the entire process is planned in advance in a detailed and structured manner. Stages such as requirements analysis, design, implementation, testing, and maintenance are clearly defined from the beginning and executed sequentially.

This process is suitable for projects with stable and well-defined requirements that are unlikely to change during development. Appropriate targets include:

- Large and complex systems such as banking or defense systems.
- Organizations that require strict processes, thorough documentation, and strong control.
- Projects developed under fixed contracts that demand early and detailed planning.

10. Ưu điểm của reuse-oriented SE?

ENG:

The main advantages of reuse-oriented SE include:

1. **Reduced development time:** Since developers do not have to build everything from scratch, common functionalities can be integrated quickly using pre-existing components.
2. **Cost efficiency:** Reuse helps lower development and maintenance costs, especially in large-scale projects.
3. **Increased reliability:** Reused components are usually well-tested and widely used, leading to fewer bugs.
4. **Focus on domain-specific problems:** Development teams can allocate more effort to the unique parts of the system rather than reinventing general features.

11. Đề xuất mô hình quy trình phù hợp với các hệ thống:

a) ABS trong xe hơi

For the Anti-lock Braking System (ABS) in a car, the most suitable process model would be the **V-Model** or the **Waterfall Model**. ABS falls under **Embedded control systems** – software embedded in hardware devices to control their operations, demanding high reliability and real-time requirements.

- The **V-Model** is particularly suitable as it emphasizes rigorous testing and verification at each development stage, ensuring that all safety and functional requirements are thoroughly checked. This is crucial for a safety-critical system like ABS, where failures can have severe consequences.
- The **Waterfall Model** could also be considered because ABS requirements are generally stable and well-defined from the outset, allowing for a sequential, strict process.

b) Hệ thống kế toán cho đại học

For a university accounting system, the **Incremental Model** or the **Spiral Model** are suitable choices. This system can be categorized as **Interactive transaction-based applications** (with interactive user interfaces that handle transactions) or even a form of **Systems of systems** (integrating multiple subsystems like finance management, HR, student administration).

- The **Incremental Model** allows for system development in small increments, with each increment delivering a complete functional part (e.g., a tuition fee management module, a faculty payroll module). This enables relevant departments (accounting, finance, administration) to get early access and provide feedback, helping to adjust the complex and frequently changing requirements of a large accounting system.
- The **Spiral Model** combines elements of iterative development and has strong risk management capabilities. For a complex, data-sensitive, and critical system like accounting, assessing and mitigating risks in each loop is essential to ensure data integrity and security.

c) Hệ thống lập kế hoạch du lịch

A travel planning system is an end-user-oriented application that requires a user-friendly interface and adaptability to rapidly changing requirements. It can be considered a type of **Interactive transaction-based application** (e.g., booking tickets, reserving accommodation) and might also incorporate features of

Entertainment systems (e.g., exploring destinations, reviews). Therefore, the **Agile Model**, especially methodologies like **Scrum** or **Kanban**, is the optimal choice.

- The **Agile Model** focuses on iterative development and adapting to change, which is highly suitable for applications with fluid requirements that need continuous user feedback. Releasing small, frequent versions helps integrate feedback and continuously improve the user experience, ensuring the product continuously meets the evolving demands of the travel market.

12. CASE là gì? Tầm quan trọng?

CASE stands for **Computer-Aided Software Engineering**. It's a set of software tools designed to automate, support, and improve activities throughout the software development life cycle (SDLC), from requirements analysis, design, programming, and testing, to maintenance. CASE tools help developers and software engineers work more efficiently, minimize errors, and enhance product quality.

Importance of CASE:

1. **Increased productivity:** Automates repetitive tasks, saving time and effort for developers.
2. **Improved software quality:** Ensures consistency in design and code, and helps detect errors early in the development process.
3. **Supports team collaboration:** Facilitates version control, documentation, and coordination among team members.
4. **Process standardization:** Encourages systematic and structured adherence to the software development lifecycle.

Chương 3. Kỹ thuật phân tích yêu cầu

1. Yêu cầu phần mềm là gì?

VIE: Định nghĩa **yêu cầu phần mềm** là tất cả những nhu cầu tính năng sản phẩm mà người dùng muốn, bao gồm chức năng, hiệu năng, giao diện,… Các yêu cầu thường xoay quanh 4 nhóm sau: yêu cầu về phần cứng; yêu cầu về phần mềm, yêu cầu về data (dữ liệu) và cuối cùng là những yêu cầu về con người. Khi nhận các brief từ khách hàng, đội phát triển phần mềm phải tiến hành tìm hiểu, phân tích yêu cầu phần mềm để cuối cùng lập ra một bản đặc tả chuẩn chỉnh nhất.

ENG: The definition of **software requirements** is all the product feature needs that users want, including functionality, performance, interface, etc. Requirements often revolve around the following 4 groups: hardware requirements; software requirements, data requirements and finally human requirements. When receiving briefs from customers, the software development team must research and analyze software requirements to finally create the most standard specification.

2. Requirements engineering là gì?

VIE: Requirements Engineering (Kỹ nghệ yêu cầu) là một quy trình kỹ thuật phần mềm và hệ thống, bao gồm các hoạt động khơi gợi, thu thập, phân tích, tài liệu hóa, xác minh và quản lý các yêu cầu cho một giải pháp phần mềm hoặc hệ thống. Nó còn được gọi là phân tích yêu cầu một cách có hệ thống.

ENG: Requirements Engineering is a software and systems engineering process that involves eliciting, gathering, analyzing, documenting, verifying, and managing requirements for a software solution or system. It is also known as systematic requirements analysis.

3. Phân biệt user vs. system requirements.

1. User Requirements:

- High-level descriptions of what users expect the system to do.
- Typically written in natural language that is easy for non-technical stakeholders to understand.
- Aims to communicate the system's functionalities from the user's perspective.

Example:

"The system shall allow users to log in using an email and password."

2. System Requirements:

- Detailed and precise descriptions of the system's behavior, performance, and technical constraints.
- Written for software engineers to use during system design and development.
- Includes both functional and non-functional requirements.

Example:

"The system shall validate login credentials within 2 seconds and log all failed login attempts."

4. Phân biệt functional vs. non-functional requirements.

1. Functional Requirements:

- Describe the specific functions and behaviors the system must perform.
- Focus on what the system **should do** in response to various inputs or situations.
- Answer the question: **"What does the system do?"**

Example:

"The system shall allow users to search for products by keyword."

2. Non-Functional Requirements:

- Define the **quality attributes** of the system such as performance, reliability, security, scalability, etc.
- Concerned with **how** the system performs its functions rather than what functions it performs.
- Answer the question: **"How does the system behave?"**

Example:

"The system shall be able to handle at least 1000 search requests per second."

5. Các loại yêu cầu phi chức năng.

Non-Functional Requirements describe **how** a system performs rather than **what** it does. Common types include:

a. Performance Requirements: Concern system speed, response time, throughput.

- Example: “The system must respond within 2 seconds for 95% of requests.”

b. Reliability Requirements: Define system stability and ability to recover from failures.

- Example: “System uptime must be at least 99.9% per month.”

c. Security Requirements: Include authentication, authorization, encryption, and data protection.

- Example: “User passwords must be stored using encryption.”

d. Scalability Requirements: Define how the system can grow with increased load or users.

- Example: “The system shall support twice the current user base with no performance degradation.”

e. Availability Requirements: Specify when and how often the system must be accessible.

- Example: “The system must be available 24/7, except for planned maintenance.”

f. Maintainability Requirements: Refer to ease of updating, fixing, or enhancing the system.

- Example: “Source code must be well-documented to support maintenance.”

g. Compatibility/Portability Requirements: Address the system’s ability to work across different platforms or environments.

- Example: “The web app must run smoothly on Chrome, Firefox, and Safari.”

6. **Product/organizational/external non-functional requirements.**

a. Product Requirements

- Define **operational qualities** of the software product itself.
- Focus on **how the software behaves** during execution.

Examples:

- Performance
- Reliability
- Usability
- Compatibility
- Security

Example Sentence:

"The system shall respond to 95% of user requests in less than 1 second."

b. Organizational Requirements

- Reflect **internal policies and standards** of the development organization.
- Focus on **how the software is built and maintained**.

Examples:

- Coding standards
- Development processes
- Tool or platform requirements

Example Sentence:

"The system shall be developed in Java and comply with the company’s internal coding standards."

c. External Requirements

- Arise from **outside the organization**, such as legal regulations, external stakeholders, or third-party constraints.
- Ensure compliance and operability within **real-world environments**.

Examples:

- Data protection laws (e.g., GDPR)
- Interoperability with external systems
- Industry-specific regulations

Example Sentence:

"The system shall encrypt all personal data to comply with GDPR regulations."

7. **Tài liệu SRS là gì? Cấu trúc chuẩn IEEE.**

SRS (Software Requirements Specification) is a formal document that defines all the software requirements to be developed. It serves as a **contract between stakeholders** – customers, users, analysts, and developers.

- Ensures all parties understand the **goals, functionalities**, and **constraints** of the software system.
- Forms the basis for **design, development, testing, and maintenance**.

According to IEEE standards (such as IEEE 830 or IEEE 29148), the structure of an SRS typically

includes:

I. Introduction

- Purpose
- Scope
- Definitions, Acronyms, Abbreviations
- References
- Overview

II. Overall Description

- Product Perspective
- Product Functions
- User Characteristics
- Assumptions and Dependencies

III. Specific Requirements

- Functional Requirements
- Non-Functional Requirements
- Interface Requirements (user interface, hardware, software, communication)

IV. Appendices

- May include diagrams, models, use cases, or supporting documents.

8. Cấu trúc 1 yêu cầu chức năng theo IEEE.

A **Functional Requirement** specifies what the system shall do – including the features, behavior, and interactions of the system.

According to **IEEE standards** (IEEE 830/29148), a well-written functional requirement should be clear, unique, and testable, and often includes the following elements.

Element	Description
ID	Unique identifier for the requirement
Title	Short, descriptive name of the function
Description	Detailed explanation of what the system must do
Precondition	Conditions that must be true before the function starts
Main Flow	The main sequence of steps performed by the system
Alternative Flow	Alternate or exceptional scenarios and how the system handles them
Postcondition	System state after the function is completed
Priority	Importance level (High – Medium – Low)
Testability	Criteria or method to verify the function works as specified

9. Mô tả yêu cầu chức năng tra cứu điểm.

Element	Content
ID	FR-02
Title	View Academic Grades
Description	The system shall allow students to log in and view their course grades by semester.
Precondition	The student has successfully logged into the system.
Main Flow	1. Student accesses the “View Grades” feature 2. Selects a semester 3. System displays the list of courses and corresponding grades.
Alternative Flow	If the student is not logged in → system prompts for login before continuing.
Postcondition	The system displays the student’s grades for the selected semester.
Priority	High
Testability	Verify by logging in with a student account and querying grades for a specific

	semester.

10. Tiến trình xác định và phân tích yêu cầu.

Requirements identification and analysis is the process of discovering, understanding, clarifying, and analyzing what stakeholders need from a software system.

Step	Name	Description
1	**Identify sources**	Identify all stakeholders (users, customers, managers, etc.)
2	**Requirement collection**	Use techniques like interviews, surveys, observation, document analysis
3	**Làm rõ yêu cầu(Elicitation)**	Understand and clarify requirements in depth to avoid misinterpretations
4	**Requirement analysis**	Detect conflicts, duplicates, set priorities, check feasibility & consistency
5	**Requirement modeling**	Represent requirements using Use Cases, Activity Diagrams, DFDs, etc.
6	**Requirement validation**	Ensure that requirements are correct, testable, accepted by stakeholders
7	**Requirement management**	Track and update requirements throughout the software lifecycle

11. Khó khăn khi tìm hiểu stakeholder requirements?

VIE: Stakeholder requirements là những mong muốn, mục tiêu hoặc ràng buộc được đưa ra bởi các bên liên quan như: khách hàng, người dùng cuối, nhà quản lý, kỹ sư bảo trì,... Các yêu cầu này thường phức tạp, đa dạng và không phải lúc nào cũng được diễn đạt rõ ràng.

ENG: Stakeholder requirements refer to the needs, expectations, and constraints provided by individuals or groups who have an interest in the system. These requirements are often diverse, vague, or conflicting.

Challenge	Description
1. Unclear needs	Many stakeholders are non-technical and struggle to articulate their true needs
2. Conflicting requirements	Different stakeholders may provide contradictory or inconsistent requirements
3. Changing requirements	Business context or stakeholder perspectives may evolve over time
4. Poor communication	Gaps between technical analysts and stakeholders due to jargon, culture, or assumptions
5. Limited time/resources	Inadequate resources to thoroughly gather and analyze requirements
6. Unrealistic expectations	Stakeholders expect features beyond the scope or technical feasibility of the system
7. Incomplete stakeholder representation	Only a subset of stakeholders is involved, leading to biased or incomplete requirements

12. Phương pháp khám phá yêu cầu.

(Requirement Elicitation Techniques)

Requirement elicitation is the process of gathering information from stakeholders to identify the needs and expectations for a software system.

Technique	Description
1. Interview	One-on-one conversations to understand stakeholder needs in depth
2.Questionnaire/Survey	Use structured forms to collect information from many stakeholders efficiently
3. Workshop	Collaborative sessions with multiple stakeholders to discuss and agree on requirements
4. Observation	Watch users in their work environment to identify implicit or unspoken needs
5. Document Analysis	Study existing documentation (manuals, reports, specifications) to extract

	relevant needs
6. Prototyping	Build early models or mockups of the system to gather user feedback
7. Scenario / Use Case	Describe specific user interactions to clarify and validate requirements
8. Brainstorming	Collaborative idea generation sessions with developers and stakeholders

13. Requirements validation là gì? Các kỹ thuật?

Requirements validation is the process of ensuring that the documented requirements are complete, consistent, correct, feasible, and aligned with stakeholder needs.

The goal is to detect errors early before they become expensive to fix during implementation.

Common Issues in Requirements:

- Ambiguity (mơ hồ) or lack of clarity
- Conflicting requirements
- Unrealistic or infeasible expectations
- Unverifiable requirements (cannot be tested)

Technique	Description
Inspection / Review	Stakeholders and developers read and examine requirements documents for errors
Prototyping	Build mockups or early versions to get user feedback
Consistency Check	Detect contradictory or overlapping requirements
Completeness Check	Ensure no important requirement is missing
Use Case / Scenario Walkthrough	Simulate real-world user actions to validate requirements
Feasibility Analysis	Check whether requirements can be implemented within the given constraints (time, tech, etc.)

14. Tại sao phải quản lý yêu cầu?

Requirements Management is the process of tracking, controlling, and updating software requirements throughout the project lifecycle to ensure the system aligns with its goals.

Reason	Description
Requirements change often	Stakeholders may change their minds, or external conditions may evolve — requiring updates
Prevent misunderstanding	Proper management avoids misinterpretation, conflict, or duplication of requirements
Efficient tracking	Helps identify which requirements are completed, pending, or linked to implementation or testing
Scope control	Avoids "scope creep" where uncontrolled changes overwhelm the team
Reduce cost and risk	Catching issues early in the requirement phase saves cost and time
Clear communication	Ensures everyone shares a common understanding of what the system should do
Supports testing and maintenance	Well-managed requirements provide a base for validation, testing, and future system upgrades

Chương 4. Mô hình hóa hệ thống và thiết kế kiến trúc

4.1. Mô hình hóa hệ thống

1. System modeling là gì? Việc mô hình hóa hệ thống mang lại lợi ích gì trong quy trình phát triển phần mềm?

+ **System modeling** is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
+ **System modeling** has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
+ **System modeling** helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Benefits of system modeling in the software development process

- **Improved communication** among stakeholders by providing a shared understanding of system requirements and design.
- **Risk reduction** by identifying design flaws or incomplete requirements early in the development lifecycle.
- **System documentation** that supports future maintenance and evolution.
- **Support for analysis and design**, allowing for a clearer and more systematic approach.
- **Increased reusability** through modular and well-defined model components.
- **Facilitation of automation tools**, such as code generation or model-based testing.

2. Trình bày và so sánh bốn góc nhìn (perspectives) chính trong system modeling: external, interaction, structural và behavioral. (Slide 9 – Chuong 5)

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.
    - **External vs Interaction:** External focuses on who interacts with the system, while Interaction focuses on how components and users interact.
    - **Structural vs Behavioral:** Structural shows the static architecture, while Behavioral illustrates dynamic behaviors.
    - Combining all perspectives provides a more complete and error-resilient system design.

3. UML là gì? Kể tên và nêu ngắn gọn mục đích sử dụng của 5 loại sơ đồ UML phổ biến được sử dụng trong system modeling. (Slide 10- Chapter 5)

UML (Unified Modeling Language) is a standardized modeling language used to visualize, design, and document the architecture of software systems. UML provides a set of diagrams for representing both the static structure and dynamic behavior of systems.

Five common UML diagrams and their purposes:

- **Activity diagrams,** which show the activities involved in a process or in data processing.
- **Use case diagrams**, which show the interactions between a system and its environment.
- **Sequence diagrams,** which show interactions between actors and the system and between system components.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.

4. **Context diagram dùng để làm gì? Trình bày vai trò của nó trong việc xác định ranh giới hệ thống.**

(Slide 13-14 Chapter 5)

5. **Mô hình Use Case được sử dụng như thế nào trong việc xác định yêu cầu phần mềm? Một Use Case bao gồm những thành phần nào?**

(Slide 19-20 Chapter 5)

The Use Case model is a crucial tool during the **requirements analysis** phase. It helps to:

- **Identify and describe the key functionalities** that the system must provide.
- **Clarify interactions between users (actors)** and the system to ensure real-world needs are met.
- **Facilitate communication among stakeholders** (clients, developers, testers) through a clear visual format.
- **Serve as a basis for system design, testing, and requirement management**.

The model is typically represented using a **Use Case Diagram**, accompanied by a detailed **Use Case Description** in text.

A Use Case typically contains the following key components:

Component	Description
Use Case Name	A short name describing the main function.
Actor	The user or system interacting with the function.
Goal	The expected outcome or purpose of the use case.
Preconditions	Conditions that must be met before execution.
Main Flow	The typical sequence of interactions between the actor and system.
Alternative/Exception Flows	Variations or exceptional situations from the main flow.
Postconditions	The state of the system after the use case completes.

6. **Sequence diagram thể hiện điều gì trong mô hình hóa hệ thống? Khi nào nên sử dụng sơ đồ này?**

(Slide 19-20 Chapter 5)

A Sequence Diagram should be used when:

- You need to **model a specific Use Case** that involves multiple steps and interactions.
- You want to **visualize time-based execution flow**, especially in sequential or event-driven systems.
- You need to **clarify the roles of different objects** in handling a user request or process.
- During system design, to define the **execution logic** for complex features.

7. **Trình bày khái niệm và mục tiêu của mô hình trạng thái (State Machine Model). Đưa ví dụ minh họa một hệ thống thực tế.** *(Slide 44-45 Chapter 5)*

Ví dụ minh họa thực tế:

8. **Thế nào là Generalization trong sơ đồ lớp UML? So sánh với Aggregation và Composition.**

In UML class diagrams, Generalization represents an inheritance relationship between classes. A subclass inherits attributes and behaviors from a superclass.

- Notation: A hollow arrow pointing from the subclass to the superclass.
- Example: Bird is the superclass, Penguin is the subclass → Penguin inherits from Bird.

Comparison: Generalization vs Aggregation vs Composition

Aspect	Generalization	Aggregation	Composition
Relationship type	Inheritance (**is-a**) relationship	Loose containment (**has-a**, but with independent life cycles)	Strong containment (**has-a**, with dependent life cycles)
Lifecycle dependency	No effect on object lifespan	Part **can exist independently** of the whole	Part **cannot exist independently**; deleted with the whole
Example	`Cat` inherits from `Animal`	`Classroom` has `Students` – students still exist if classroom is gone	`House` has `Rooms` – rooms are destroyed if the house is destroyed
UML Notation	Hollow arrow (--)▶	Line with a **hollow diamond** (◇)	Line with a **filled diamond** (◆)

9. **Mô tả khái niệm Model-Driven Engineering (MDE) và trình bày các loại mô hình chính trong MDA: CIM, PIM, PSM.**

MDE is a software development methodology that emphasizes the use of models as the central artifacts throughout the entire development lifecycle.

Models are not just documentation; they are actively used for code generation, testing, and

deployment.

Main Model Types in Model-Driven Architecture:

Model Type	Full Name	Function Description
CIM	Computation-Independent Model	**Business-oriented model** that focuses on **requirements and business context**, without mentioning system internals.
PIM	Platform-Independent Model	**Abstract system model**, describing structure and behavior **independently of any technology platform**.
PSM	Platform-Specific Model	A **technology-bound model**, transforming the PIM into a concrete version **tailored for a specific platform** (e.g., Java, .NET) for code generation.

10. Phân tích sự khác biệt giữa mô hình hướng sự kiện (event-driven modeling) và mô hình hướng dữ liệu (data-driven modeling). Ứng dụng cụ thể của từng loại.

a. What is Event-Driven Modeling?

A modeling approach focused on **events** that occur and the **system's responses** to those events.

- The system is seen as a set of **actors**, **events**, and **reactions**.
- Example: A user clicks a button → the system performs an action.

Common diagrams:

- **State Machine Diagram**
- **Event Trace Diagrams**
- **Sequence Diagrams**

b. What is Data-Driven Modeling?

Focuses on **data**, its **flow**, and how the system **processes** it.

- Emphasizes **transformation and movement of information** through functions.
- Data controls the process flow.

Common diagrams:

- **Data Flow Diagram (DFD)**
- **Entity-Relationship Diagram (ERD)**

c. Practical Applications

- **Event-driven modeling**:
  - **ATM systems**: react to user input.
  - **IoT devices**: respond to sensor triggers.
  - **Mobile apps**: respond to gestures and UI events.
- **Data-driven modeling**:
  - **Student management system**: grade processing and report generation.
  - **Accounting system**: data entry, summary reports.
  - **ERP systems**: managing data flow across departments.

4.2. Thiết kế kiến trúc phần mềm

11. Trình bày sự khác biệt giữa kiến trúc phần mềm (software architecture) và thiết kế phần mềm (software design). Vai trò của mỗi giai đoạn trong quy trình phát triển phần mềm là gì?

1. What is Software Architecture?

It refers to the **high-level structure** of a software system, defining:

- Major components (modules, layers)
- Interactions among components (interfaces, communication)
- Key system-wide decisions like architectural styles (e.g., layered, microservices), technology stack, and deployment strategy.

2. What is Software Design?

It involves the **detailed design** of components identified during the architecture stage:

- Specifies algorithms, data structures, internal logic of modules.
- Applies design patterns such as Singleton, Factory, or MVC.

3. Summary Comparison

Aspect	Software Architecture	Software Design
Abstraction level	High – overall system view	Medium – internal logic of components
Focus	Structure, modules, technologies	Internal logic, data handling, algorithms
Impact	Affects performance, scalability, maintainability	Affects function implementation and efficiency
Typical diagrams	Component, Deployment, Architecture views	Class Diagram, Sequence Diagram, Design Patterns

4. Role in Software Development Lifecycle

- **Software Architecture**:
  - Sets the technical direction.
  - Enables scalability, maintainability, and integration.
- **Software Design**:
  - Provides the blueprint for developers.
  - Supports coding, testing, and performance optimization.

12. Architectural design là gì? Tại sao cần thực hiện thiết kế kiến trúc trước các bước thiết kế chi tiết khác?

Architectural Design is the **high-level design phase** that defines:

- The **overall structure** of the software system, including major components/modules.
- The **interactions** between these components (interfaces, connectors).
- The **core architectural decisions**, such as architectural style (monolithic, layered, microservices), technology stack, and system-level design principles.

Commonly represented by Component Diagrams, Deployment Diagrams, or architectural views.

There are several crucial reasons:

- Making early critical decisions
- Managing complexity
- Providing a framework for detailed design
- Promoting reusability
- Ensuring consistency and quality
- Facilitating communication among stakeholders

13. Kiến trúc phần mềm ảnh hưởng như thế nào đến các thuộc tính phi chức năng của hệ thống như: hiệu năng, bảo mật, an toàn và bảo trì? Đưa ví dụ minh họa.

Software architecture plays a crucial role in shaping and ensuring the **non-functional attributes of a system**. NFRs are requirements that are not directly related to the functions the system performs, but rather to how the system performs, such as performance, reliability, scalability, security, usability, etc. A flawed architectural decision can make it difficult or impossible to achieve the desired NFRs.

- **Performance:** Architecture determines the system's speed and processing capability. Example: A Microservices architecture allows individual parts to scale, boosting overall performance (e.g., order processing service can scale independently).
- **Security:** Architecture establishes security boundaries and protection mechanisms. Example: Using a DMZ to isolate public-facing servers from the internal network enhances security.
- **Safety:** Architecture ensures reliable system operation, especially during failures. Example: In control systems, redundancy in the architecture helps maintain operation even if a component fails.
- **Maintainability:** Architecture influences how easily the system can be modified and evolved. Example: A modular architecture simplifies bug fixes or feature additions because components are less interdependent.

14. Mô hình kiến trúc 4+1 gồm những view nào? Nêu ngắn gọn vai trò của mỗi view trong việc thiết kế và trình bày kiến trúc hệ thống.

The **4+1 Architectural View Model** consists of 5 views:

1. **Logical View:** Describes **functionality** (objects, classes) for end-users.
2. **Process View:** Describes **concurrent/distributed interactions** (performance, reliability).
3. **Development View:** Describes **module/code structure** for developers.
4. **Physical View:** Describes **mapping to hardware** (deployment, performance).
5. **Scenario View:** Describes **use cases/operational flows** to validate other views.

15. Trình bày đặc điểm của mô hình kiến trúc Layered Architecture. Ưu điểm và hạn chế của mô hình này là gì? Trong trường hợp nào nên sử dụng mô hình này?

Layered architecture is one of the most common architectural patterns, organizing a system into stacked layers. Each layer has a specific responsibility and is typically only allowed to interact with the layer immediately below it (downward communication) or both above and below depending on the specific design (strict vs. relaxed layering).

Common layers often include:

- **Presentation Layer (UI Layer):** Responsible for displaying the user interface and handling user interactions.
- **Application Layer (Business Logic Layer):** Contains the main business logic of the system, coordinating activities and processes.
- **Business/Domain Layer:** (Sometimes merged with Application Layer or separated) Holds core business rules and entities.
- **Persistence Layer (Data Access Layer):** Handles data storage and retrieval from databases or other data sources.
- **Database Layer:** The physical database or Database Management System (DBMS).

2. Advantages:

- Easier Maintenance/Testing: Changes in one layer have minimal impact on others.
- Reusability: Lower layers can be reused.
- Clear Separation: Easier to manage complexity.

3. Disadvantages:

- Performance Overhead: Cross-layer communication can be slow.
- Rigidity: Small changes in lower layers might cascade upwards.
- Initial Cost: Requires careful interface design between layers.

4. When to Use:

Suitable for traditional enterprise (monolithic) applications, systems needing clear functional separation, and where ease of maintenance/testing is crucial.

16. So sánh các mô hình kiến trúc: Client-Server, Repository, MVC và Pipe-and-Filter. Đặc điểm chính và ứng dụng phù hợp của từng mô hình là gì?

1. Client-Server

Main characteristics: The system is divided into two parts: clients send requests and servers handle these requests and return results.

Suitable applications: Networked systems such as the web (browser-server), online banking applications, email services.

2. Repository

Main characteristics: System components communicate through a shared central data repository. Data is centrally stored and managed.

Suitable applications: Compiler systems, central data management systems, CASE tools (Computer-Aided Software Engineering).

3. MVC (Model - View - Controller)

Main characteristics: Separates the system into three components: Model (logic and data), View (user interface), Controller (interaction and coordination). Enhances flexibility and maintainability.

Suitable applications: Web applications, GUI applications, systems with rich user interaction.

4. Pipe-and-Filter

Main characteristics: Data is processed through a sequence of processing steps (filters), each with a specific function, and passed through pipes.

Suitable applications: Signal processing systems, sequential data processing like compilers, audio/image processing systems.

17. Trình bày các bước chính trong quá trình thiết kế lớp trong phần mềm hướng đối tượng. Sơ đồ lớp UML thể hiện thông tin gì về hệ thống?

(Describe the main steps in class design in object-oriented software. What information does a UML class diagram convey about the system?)

The main steps in class design in object-oriented software include:

- **Identify classes:** Based on system requirements, analyze key entities and behaviors to identify necessary classes.
- **Identify attributes and methods:** Define the data (attributes) and behaviors (methods) specific to each class.
- **Define relationships between classes:** Including associations, inheritance, dependencies, and aggregations.
- **Refine and optimize design:** Review abstraction, encapsulation, and separation of responsibilities among classes.
- **Represent with UML class diagram:** Model the classes and their relationships to illustrate the system architecture.

A UML class diagram represents:

- The **classes** in the system.
- **Attributes** and **methods** of each class.
- **Relationships** among classes: inheritance, association, dependency, etc.
- **Visibility** levels: public, private, protected.
- **Roles and constraints** in relationships, if applicable.

18. Thế nào là quan hệ kế thừa (Generalization) và quan hệ thành phần (Composition) trong sơ đồ lớp UML? Cho ví dụ minh họa.

1. Generalization: Generalization represents an "is-a" relationship between classes. A subclass inherits attributes and methods from a superclass. This models inheritance in object-oriented programming.

- **UML Notation:** A hollow triangle arrow pointing from the subclass to the superclass.
- **Example**: Class Dog inherits from class Animal. → Dog is a type of Animal.

2. Composition: Composition represents a strong "has-a" or "part-of" relationship. One class owns another class exclusively; if the container class is destroyed, so are its parts.

- **UML Notation:** A filled diamond pointing from the whole to the part.
- **Example:** Class House has a component Room. → If the House is deleted, the Room ceases to exist as well.

19. Mẫu thiết kế (Design Pattern) là gì? Trình bày ví dụ mẫu MVC hoặc Singleton và nêu rõ khi nào nên sử dụng mẫu đó. (Slide 29, 30 – Chương 7)

Design Pattern is:

- A design pattern is a way of reusing abstract knowledge about a problem and its solution.
- A pattern is a description of the problem and the essence of its solution.
- It should be sufficiently abstract to be reused in different settings.
- Pattern descriptions usually make use of object-oriented characteristics such as inheritance and polymorphism.

Example 1: Singleton Pattern

- **Purpose:** Ensure that a class has only one instance and provide a global point of access to it.
- **When to use:** When a single shared resource is needed, such as a database connection, logger, or configuration manager.
- **Example:**

```java
public class Database {
    private static Database instance;
    private Database() {} // private constructor
    public static Database getInstance() {
        if (instance == null) {
            instance = new Database();
        }
        return instance;
    }
}
```

Example 2: MVC Pattern (Model – View – Controller)

- **Purpose:** Separate concerns by dividing the application into Model (data), View (UI), and Controller (logic) to enhance maintainability and scalability.
- **When to use:** When developing user interface applications, especially web or desktop applications.
- **Example:**
  - Model: processes student data.
  - View: displays student list.
  - Controller: handles user input and connects Model with View.

20. Thiết kế giao diện người dùng cần tuân thủ những nguyên tắc nào để đảm bảo tính hiệu quả và thân thiện? Nêu ít nhất 3 nguyên tắc quan trọng.

An effective and user-friendly UI design helps users interact smoothly, make fewer errors, and enjoy a better experience. Here are 3 key principles:

- **Consistency:** Keep the layout, colors, icons, and interactions consistent across screens so users can learn and predict the system's behavior easily.
- **Clear Feedback:** The system should give immediate and clear responses to user actions (e.g., button changes color when clicked, success message after saving).
- **Simplicity and Minimalism:** Remove unnecessary elements, present content clearly, and support

intuitive interactions to reduce users' cognitive load.

Chương 5. Thiết kế và cài đặt phần mềm

21. Trình bày mối quan hệ giữa hoạt động thiết kế phần mềm và cài đặt phần mềm. Vì sao nói hai hoạt động này thường được thực hiện xen kẽ nhau?

Software design is the phase in which the structure and behavior of the software system are defined. It includes architectural design, module design, user interface design, and data structure design. Meanwhile, software implementation (or coding) is the process of translating the design into executable source code.

These two activities are closely related: design serves as the blueprint for coding, while coding often provides feedback that leads to design modifications. In practice, when developers begin implementing, they may encounter issues or inefficiencies in the original design. This necessitates updates or revisions to the design to align with real-world constraints.

Therefore, design and implementation are often carried out in an interleaved, iterative manner rather than as strictly sequential steps. This approach helps detect issues early and allows for more flexible and efficient software development.

22. Phân tích vai trò của mô hình UML trong thiết kế hướng đối tượng. Những loại mô hình UML nào thường được sử dụng để mô tả thiết kế hệ thống?

UML (Unified Modeling Language) plays a vital role in object-oriented design by providing a standardized and visual way to model software system components. It helps clearly represent the structure, behavior, and interactions of objects within the system, facilitating effective communication among development team members.

In system design, UML bridges the gap between requirement analysis and detailed design. It also aids in validating, maintaining, and evolving the system throughout its lifecycle.

Commonly used UML diagrams to describe system design include:

- **Class diagram:** Describes classes, attributes, methods, and relationships between classes.
- **Sequence diagram:** Shows the order of message exchanges between objects over time.
- **Activity diagram:** Represents workflows or business processes.
- **State diagram:** Describes changes in an object's state in response to events.
- **Component diagram:** Models software components and their dependencies.

23. Trình bày các bước chính trong quy trình thiết kế hướng đối tượng. Đâu là vai trò của việc xác định giao tiếp (interface) của các đối tượng?

The object-oriented design process typically includes the following main steps:

Identify classes and objects: Based on the analysis model, determine the classes that represent entities in the system.

- **Identify relationships between classes**: Include inheritance, association, aggregation, and dependency relationships.
- **Define attributes and methods for each class:** Specify the data (attributes) and behaviors (methods) for each class.
- **Define object interfaces:** Design the public methods that objects expose to interact with other objects.
- **Design object interactions:** Use sequence or collaboration diagrams to describe how objects interact in specific scenarios.
- **Optimize and review the design:** Evaluate the design's clarity, scalability, and efficiency, and refine as needed.

24. Mẫu thiết kế (design pattern) là gì? Trình bày các thành phần cấu thành nên một mẫu thiết kế. Cho ví dụ về mẫu Observer và mục đích sử dụng của nó.

(Slide 29 -35 Chương 7)

25. So sánh ưu và nhược điểm của các kiểu kiến trúc phần mềm sau: Client/Server, Kiến trúc phân tầng (Layered), N-Tier, và Hướng dịch vụ (SOA).

Architecture Style	Advantages	Disadvantages
Client/Server	- Clear separation between client and server. - Easy to deploy and manage.	- Difficult to scale if the server is overloaded. - High dependency on the central server.
Layered	- Enhances modularity, maintainability, and replaceability of layers. - Facilitates testing and extension.	- May reduce performance due to intermediary layers. - Layer communication can be complex.
N-Tier	- Supports flexible distributed deployment. - Each tier can be independently managed.	- More complex configuration and management. - Requires robust infrastructure and security.
Service-Oriented (SOA)	- Facilitates integration across heterogeneous systems. - High service reusability.	- Complex to design and standardize. - Potentially high operational costs.

26. Giải thích khái niệm và lợi ích của việc tái sử dụng phần mềm ở các mức: trừu tượng, đối tượng, thành phần, và hệ thống.

Software reuse refers to the process of using previously developed software assets (such as code, designs, or systems) to build new software, aiming to save time, reduce cost, and improve quality. Reuse can occur at various **levels of abstraction**, each offering distinct benefits:

1. **Abstraction level:**
   - **Concept:** Reusing general models, designs, or architectures, such as design patterns or architectural frameworks.
   - **Benefits:** Promotes standardized thinking; improves design consistency and efficiency across projects.
2. **Object level:**
   - **Concept:** Reusing pre-written classes, methods, or object instances.
   - **Benefits:** Reduces coding time; improves reliability and maintainability with proven code.
3. **Component level:**
   - **Concept:** Reusing self-contained software modules with well-defined interfaces (e.g., libraries, APIs, software packages).
   - **Benefits:** Easy integration; enhances modularity; supports reuse across different systems.
4. **System level:**
   - **Concept:** Reusing entire systems or large-scale services (e.g., database management systems, payment processing systems).
   - **Benefits:** Major cost savings; accelerates deployment; builds on already tested and validated systems.

27. Cấu hình phần mềm (Configuration Management) là gì? Trình bày ba hoạt động chính trong quản lý cấu hình phần mềm.

Software Configuration Management (SCM) is a systematic process for controlling and tracking changes to software throughout its development lifecycle. Its primary goal is to ensure the software remains consistent, complete, and manageable when modifications occur.

The three main activities in SCM include:

- **Version Control:** Tracks and stores different versions of source code, documents, and

software components. Common tools: Git, SVN.

- **Change Control:** Manages change requests by evaluating their impact, approving changes before implementation, and ensuring all changes are documented and controlled.
- **Configuration Auditing and Reporting:** Verifies the completeness and consistency of the system configuration; ensures that the software is built from approved components and correct versions.
- **Key benefits:** Enables controlled software development, minimizes errors from uncontrolled changes, supports team collaboration, and facilitates recovery in case of failures.

28. Trình bày khái niệm phát triển phần mềm host-target. Phân biệt giữa nền tảng phát triển và nền tảng thực thi.

Host-target software development refers to the approach where the software is developed (coded, compiled, tested) on a **host** system (development machine), while it is intended to run on a different system called the **target** (execution device).
This model is common in embedded systems, mobile apps, or software for custom hardware.

Differences between development platform and execution platform:

Aspect	Development Platform (Host)	Execution Platform (Target)
Function	Used to write, compile, test, and debug software	Used to run the software in the actual operational environment
Typical Examples	PC running Windows/Linux/MacOS	Microcontrollers, embedded systems, mobile phones, IoT devices
Supporting Tools	IDEs, compilers, debuggers	Runtime environments, firmware, device drivers
Hardware Requirements	Powerful hardware, supports various development tools	Optimized for execution, may have limited resources

29. Open source development là gì? Nêu các lợi ích và thách thức khi phát triển phần mềm theo mô hình mã nguồn mở.

(Slide 54 – Chapter 7)

Open source development is a software development model in which the source code is made publicly available and can be freely viewed, modified, distributed, and contributed to by anyone.

Open source software is usually released under licenses like GPL, MIT, or Apache, and often developed collaboratively by a global community or organizations.

Benefits of open source development:

1. **Transparency and trust:** Users can examine the source code for security and quality assurance.
2. **Low cost:** Often free or significantly cheaper than proprietary software.
3. **Strong community support:** Large developer communities help improve the software and fix bugs quickly.
4. **Flexibility and customization:** Source code can be modified to meet specific needs.
5. **Learning and innovation:** Great opportunity for students and developers to learn, practice, and contribute.

30. Trình bày các mô hình cấp phép phần mềm mã nguồn mở như: GPL, LGPL và BSD. Điểm khác biệt chính giữa các mô hình này là gì?

(Slide 61, 62 – Chapter 7)

1. GPL (General Public License):

- Requires that any software using GPL code or derived from GPL code must also be released under the GPL license.
- This is a strong copyleft license – derived works must remain open source.

2. LGPL (Lesser General Public License):

- More permissive than GPL.
- Allows non-open source software to use LGPL libraries without requiring the entire software to be open source.
- Suitable when sharing libraries but still allowing commercial use.

3. BSD (Berkeley Software Distribution License):

- A very permissive license with minimal restrictions.
- Allows code reuse, modification, redistribution, and inclusion in proprietary software.
- The only requirement is to retain the original copyright notice.

Feature	GPL	LGPL	BSD
Source code disclosure	Mandatory (strong copyleft)	Partially required (weak copyleft)	Not required
Commercial software integration	Difficult	Possible	Freely allowed
Keep original copyright	Required	Required	Required

Chương 6. Kiểm thử và định giá phần mềm

1. **Kiểm thử phần mềm là gì?** *(Slide 5-6- Chapter 8)*

   Software testing is the process of evaluating a software system to identify defects and ensure that it functions correctly according to specified requirements. This process involves executing the software with the intention of discovering bugs or errors in order to improve product quality. Software testing can be done manually or through automation, and it includes various levels such as unit testing, integration testing, system testing, and acceptance testing.

2. **Mục tiêu kiểm thử?** *(slide 7-8 Chapter 8)*

   The primary objective of software testing is to detect defects as early as possible to ensure the final product's quality. Beyond identifying bugs, testing also verifies that the software meets both functional and non-functional requirements. Key objectives include:

   - Detecting and eliminating software defects
   - Ensuring the software operates reliably and accurately
   - Validating that the software meets user requirements and design specifications
   - Improving the reliability and performance of the system
   - Supporting future maintenance and upgrades of the software

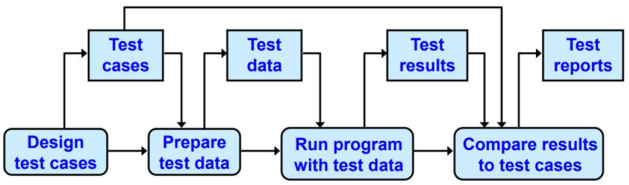
3. **Validation vs. Verification?** *(slide 12-13 chapter 8)*

Verification and Validation are two essential activities in software testing, each with distinct goals and methods:

- **Verification**: This is the process of checking whether the software product is being built correctly according to the specified requirements and design. It answers the question: "Are we building the product right?"
  → Typically involves reviews of documents, source code, design, and development processes.
  → Performed in the early stages of the software development life cycle.
- **Validation:** This is the process of evaluating whether the software meets the actual needs and expectations of the end users. It answers the question: "Are we building the right product?"
  → Typically involves executing the software and checking the output.
  → Performed after the product is developed or ready for testing.

4. **Mô hình quy trình kiểm thử.** (Slide 19 – chapter 8)

A model of the software testing process



5. **Các giai đoạn kiểm thử. (Stages of testing)** *(Slide 20 – Chapter 8)*

Stages of testing

- **Development testing, where the system is tested during development to discover bugs and defects.**
- **Release testing, where a separate testing team test a complete version of the system before it is released to users.**
- **User testing, where users or potential users of a system test the system in their own environment.**

6. **Development testing: ai thực hiện? (Slide 21-22 – Chapter 8)**

Development testing

- **Development testing includes all testing activities that are carried out by the team developing the system.**

Development testing includes activities such as:

- Unit testing
- Module-level integration testing
- Static analysis
- Automated code checks
- Debugging during development

7. **Release testing: ai thực hiện? (Slide 55 – Chapter 8)**

Release testing is the process of evaluating the complete software system before it is officially released to users. It is typically conducted by an **independent testing team** or a **Quality Assurance (QA)** team, rather than the developers who built the software.

The objectives of release testing are:

- Ensure the software meets all specified requirements
- Assess overall quality and operational stability
- Identify any remaining defects not found during development
- Evaluate the software's readiness for deployment

8. **User testing: ai thực hiện? (Slide 63- Chapter 8)**

User testing is conducted by **end users** or **customers** in a real-world or near-real environment. The goal is to determine whether the software truly meets user needs, expectations, and delivers a satisfactory user experience.

Common forms of user testing include:

- **Alpha testing:** Performed by users at the developer's site, typically early versions of the product.
- **Beta testing:** Performed by users in their own environment, usually before the official release.

9. **Unit/component/system testing: đặc điểm?**

Component: Slide 37 chapter 8

Unit: Slide 23 Chapter 8

System: Slide 43 Chapter 8

10. **Automated testing là gì? Công cụ?** (Slide 27 - Chapter 8)

Automated testing is the process of using software tools to automatically execute test cases, compare actual outcomes with expected results, and report any defects. It saves time, reduces manual effort, and increases accuracy, especially effective for repetitive or large-scale testing.

Some popular automated testing tools include:

- Selenium: For web application testing, supports multiple programming languages.
- JUnit / TestNG: For unit testing in Java.
- Postman / SoapUI: For API and web service testing.
- Appium: For mobile app testing (Android/iOS).
- Cypress: For modern web interface testing.
- Jest / Mocha: For JavaScript/Node.js testing.

11. **Test case là gì? Cấu trúc?**

A test case is a document that describes a specific testing scenario, including execution steps, input data, test conditions, and the expected result, to verify a particular functionality or behavior of the software.

The basic structure of a test case typically includes:

1. **Test Case ID** – Unique identifier
2. **Test Case Name** – Brief purpose of the test
3. **Description** – Detailed explanation of the test scenario
4. **Preconditions** – Conditions that must be met before execution
5. **Test Steps** – Specific steps to follow
6. **Test Data** – Input data required
7. **Expected Result** – The expected system behavior if it works correctly
8. **Actual Result** – The outcome observed during execution
9. **Status** – Pass / Fail based on comparison of actual vs. expected result
10. **Remarks** – Additional notes if any

12. Black-box vs. white-box testing?

Black-box testing and White-box testing are two distinct software testing approaches based on how much knowledge the tester has about the internal structure of the software.

Black-box testing:

- **Tester does not need to know the internal code or structure of the system.**
- Focuses on **inputs and outputs** to validate functionality.
- Used to verify whether the software meets user and functional requirements.
- **Example:** testing login functionality with invalid credentials to check system response.

Pros: easy to apply, ideal for functional and user-level testing.
Cons: cannot detect internal logic or hidden code errors.

White-box testing:

- **Tester needs to understand and access the source code.**
- Focuses on **control flow, internal logic, and code structure.**
- Used to verify detailed implementation like functions, branches, and loops.
- **Example:** testing all "if" conditions or "for" loops to ensure all paths execute correctly.

Pros: helps identify logic bugs and optimize code.
Cons: requires strong programming knowledge and is less suited for overall functional testing.

13. Alpha, beta, acceptance testing: khái niệm và mục đích.

- **Alpha Testing:**

This is an internal testing phase performed by the development or QA team before releasing the software to external users. Its purpose is to identify major bugs, verify core functionalities, and ensure the system performs stably in a simulated real-world environment.

- **Beta Testing:**

This is an external testing phase carried out by actual users (potential customers) in real-world settings. The purpose is to gather user feedback, uncover issues not found during alpha testing, and evaluate user satisfaction.

- **Acceptance Testing:**

This is the final verification step to ensure that the software meets the agreed-upon business and technical requirements. Its purpose is to decide whether the software is ready for official deployment. It is usually conducted by the customer or end-user representatives.

14. Kiểm thử có thể đảm bảo không còn lỗi không?

No. Software testing **cannot guarantee** that a system is completely free of bugs.

While testing helps detect and remove many defects during development, it has limitations:

- It's impossible to test **all possible inputs and scenarios**.
- Some bugs only appear in **rare or special conditions**.
- Test design and execution are done by humans, who can make mistakes.

Therefore, testing helps to **reduce risks** and **improve software quality**, but **cannot prove the absence of all defects**.