

OVERVIEW **Software Architecture: Fundamentals and Key Tasks**

This lecture gives the formal definition of software architecture and then takes it apart phrase by phrase: architecture is foundational, it translates requirements into design elements, it needs a collection of diagrams rather than one, it describes major components together with the quality they provide, and it must support detailed design and construction. The second half lists the architect's key tasks: identifying stakeholder concerns (and the tactics that satisfy them), choosing views, styles and patterns, defining components and interfaces, evaluating the architecture, and keeping the code in sync with the design.

Architecture is about the parts

Which major parts exist, how they fit and communicate, and what quality each provides. It is not about algorithms or data structures.

Quality starts here

Non-functional requirements are decided at architecture time; a tactic is the design decision that achieves one.

Many views

No single diagram can show usability, performance and deployment at once, so architecture is a collection of views for different stakeholders.

Synchronicity

How closely the built system matches its design. It drops quickly unless changes are documented, reviewed and approved.

HOW THE LECTURE IS ORGANIZED

Slides 1–8 What architecture is (and is not); architecture vs. detailed design

Slides 9–16 The formal definition, phrase by phrase

Slides 17–23 Key tasks: stakeholder concerns, quality attributes and tactics

Slides 24–28 Views, styles and patterns, evaluation, design synchronicity

Slides 29–30 Problem solving during architecture and summary

EXAM FOCUS

Final 252 Q2 (5 marks) asked for the formal definition with three of its parts explained, and for five key tasks of architectural design. Midterm 252 Q1 drew four MCQs from this lecture (architecture is not about algorithms, why it is foundational, why several views are needed, design synchronicity), Midterm 261 Q1 asked for the four key tasks among distractors, and Quiz 1 261 asked about tactics and the architecture level.

HOW TO READ THIS PAGE

Every original slide is shown unchanged. **Yellow marks** on a slide point to a numbered note: hover or tap a mark to jump to its note, and hover a note to see its marks. Questions tagged **Past exam** appeared on a real paper; **Prediction** questions are likely angles, not confirmed.



Software Design and Architecture

[Software Architecture] – Week 03, L01

NOTES

- 1 Week 3, first architecture lecture. Chapter 1 named architecture as the first **major** design activity; this lecture defines it formally and lists what an architect actually does.

SLIDE 2 / 30 Lecture Outlines

Lecture Outlines

- **Fundamentals of software architecture**
 - ✓ What is software architecture?
 - ✓ Why is it important?
- **Key tasks in software architecture**
 - ✓ Stakeholders concerns
 - ✓ Identify architectural views, styles, and patterns
 - ✓ Identify major component's and interfaces
 - ✓ Evaluating and validating the Architecture
 - ✓ Introducing policies for design synchronicity
- **Problem-solving in architecture**
- **What's next...**

NOTES

- 1 First half: the fundamentals, meaning what architecture is and why it matters.
- 2 Second half: the key tasks, from identifying stakeholder concerns to keeping the implementation in sync with the design. Final 252 asked you to list five of them.
- 3 It closes by applying Chapter 1's problem-solving framework to architecture.

SLIDE 3 / 30 Fundamentals of Software Architecture

Fundamentals of Software Architecture

➤ Architecture is:

- ✓ All about communication.
- ✓ What 'parts' are there?
- ✓ How do the 'parts' fit together?

➤ Architecture is not:

- ✓ About development.
- ✓ About algorithms.
- ✓ About data structures.

Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

3

NOTES

- 1 Architecture is first a **communication** tool: it gives every stakeholder (developers, managers, clients, operators) one shared picture of the system to discuss and decide on.
- 2 Its two questions are **what parts exist** and **how they fit together**: the major components and the connections between them.
- 3 It is **not** about development details, **algorithms** or **data structures**. Those belong to detailed and construction design. A diagram that shows sorting logic is no longer an architecture diagram.

Software architecture is NOT about:

- A. Communication
- B. High-level structure
- C. Algorithms
- D. Nonfunctional requirements

Answer: C. Algorithms — Algorithms are low-level detail handled in detailed or construction design.

SLIDE 4 / 30 Fundamentals of Software Architecture

Fundamentals of Software Architecture

What is Architecture ?

- Refers to the **high level structures** of a software system, the discipline of creating such structures, and the **documentation of these structures**.
- Architecture captures three **primary dimensions**:
 - ✓ **Structure**
 - ✓ **Communication (behavior)**
 - ✓ **Nonfunctional requirements**
- Focuses on those aspects of a system that would be **difficult to change** once the system is built.

NOTES

- 1 "Architecture" has three meanings at once: the **high-level structures** of a system, the discipline of creating them, and the documentation that records them.
- 2 It captures three dimensions: **structure** (the parts), **communication** or behaviour (how the parts interact at run time), and **non-functional requirements** (the qualities the system must have: performance, security, availability...).
- 3 It focuses on the decisions that would be **difficult to change once the system is built**: the choice of components, how they talk, where data lives. That is why these decisions are made first and with care.

Prediction

Q1

Which is NOT one of the three primary dimensions captured by software architecture?

- A. Coding style
- B. Structure
- C. Communication (behavior)
- D. Nonfunctional requirements

Answer: A. Coding style

SLIDE 5 / 30 Fundamentals of Software Architecture

Fundamentals of Software Architecture

- Let's get straight to the point, formally, we define software architecture as
- (1) The **foundational software design activity** that evaluates and translates **software requirements** (both functional and non-functional) into a **collection of design elements** that specify structural and behavioral aspects of the **major components of the system**, together with their provided quality and interrelationships required to **support the detailed design and construction** of software systems.
 - (2) and the **product** resulting from such activity.
- Extra:** Software architecture (**big-picture**)
- 1-Process** is deciding how the **major components** of the system (frontend, backend, database, etc.) will be structured and interact.
- 2-Product** is the high-level diagrams, component models.
 - **Example**
 - Imagine you are building an **e-commerce website** (like Amazon).
 - **Architecture activity (process):**
You decide the system needs:
 - A **frontend** (user interface) where customers browse and order.
 - A **backend** (server logic) to process orders.
 - A **database** to store products and users.
 - A **payment service** to handle credit cards.
 - **Architecture product (blueprint):**
A diagram showing these components (Frontend → Backend → Database → Payment Service) and how they talk to each other.

NOTES

- 1 The formal definition, part (1): architecture is **the foundational software design activity**. It comes first and everything else is built on it.
- 2 It **evaluates and translates** the software requirements, both functional and non-functional, into design.
- 3 The output is a **collection of design elements** (diagrams) specifying the structural and behavioural aspects...
- 4 ...of the **major components** of the system, together with the quality they provide and their interrelationships.
- 5 All of this exists to **support detailed design and construction**: it is an input to the next phases, not an end in itself.

- 6 Part (2): architecture is also the **product** of that activity, the high-level diagrams and component models (the same process/product split as Chapter 1).
- 7 The e-commerce example: the process decides that the system needs a frontend, a backend, a database and a payment service; the product is the diagram Frontend → Backend → Database → Payment Service showing how they talk.

EXAM ANGLE

Past exam · Final 252 · Q2-1 · 2.5 marks

Q1

Define the formal software architecture and then pick any three major parts of the definition and explain them.

Answer: Definition: software architecture is the foundational design activity that evaluates and translates software requirements (functional and non-functional) into a collection of design elements that specify the structure and behaviour of the major components of the system, with their provided quality and interrelationships, to support detailed design and construction; and the product of that activity.

Three parts: *Foundational:* it is not optional; it is forward engineering where designing for quality begins. *Translates requirements:* textual requirements become graphical design elements, so they can be traced through the life cycle. *Collection of design elements:* no single diagram can describe a system, so several views are needed. (Also acceptable: *major components with their quality; supports detailed design and construction.*)

SLIDE 6 / 30 Extra: Architecture vs. Detailed Design

Extra: Comparison between Software Architecture and Detailed Design

◆ Software Architecture (big picture)

- **Process:** Deciding how the major components of the system (frontend, backend, database, etc.) will be structured and interact.
- **Product:** The blueprint (high-level diagrams, component models).
- **Example:** For an online shopping app → deciding that you need a frontend (website/app), backend (server logic), payment service, and database.

◆ Software Design (detailed picture)

- **Process:** Working out the details inside those components — the classes, functions, data structures, algorithms, and how they solve the requirements.
- **Product:** The design diagrams/documents (class diagrams, sequence diagrams, flowcharts, etc.).
- **Example:** For the same online shopping app → designing how the “ShoppingCart” class works, what methods it has (addItem(), removeItem(), checkout()), and how it communicates with the Payment class.

NOTES

- 1 **Architecture** (big picture). Process: deciding how the major components (frontend, backend, database...) are structured and interact. Product: the blueprint of high-level diagrams and component models.

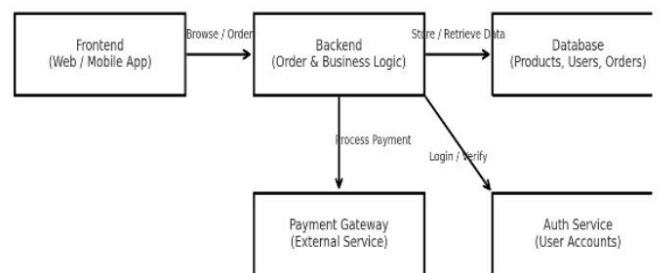
- 2 **Detailed design** (detailed picture). Process: working out what is inside those components (classes, functions, data structures, algorithms). Product: class diagrams, sequence diagrams, flowcharts. For the same shop, it designs the `ShoppingCart` class with `addItem()`, `removeItem()`, `checkout()` and how it calls the `Payment` class.

SLIDE 7 / 30 **Extra: Architecture = high-level interaction**

Extra: Comparison between Software Architecture and Detailed Design

- 1. **Architecture = High-level Interaction (Big Picture)**
- Focus:** Which components exist and how they connect.
- Shows:** Subsystems or major modules, not individual classes.
- Interaction = component-to-component.**
- Example (Online Shopping App, Architecture):**
Frontend ↔ Backend ↔ Database ↔ Payment Gateway. The diagram would show arrows like: "Frontend sends requests to Backend," "Backend stores data in Database," "Backend calls Payment Gateway."
- So here, we don't care how each part works internally — only which components exist and how they talk.**

Architecture: Online Shopping App (Amazon-like)



NOTES

- 1 Architecture is about **high-level interaction**: which components exist and how they connect. It shows subsystems or major modules, never individual classes.
- 2 The interaction it shows is **component-to-component**: "Frontend sends requests to Backend", "Backend stores data in Database", "Backend calls Payment Gateway".
- 3 At this level we deliberately **don't care how each part works inside**. Only which parts exist and how they talk.

Extra: Continue

2. Design = Low-level Interaction (Detailed Picture)

Focus: How components are implemented internally.

Shows: Classes, methods, data structures, algorithms.

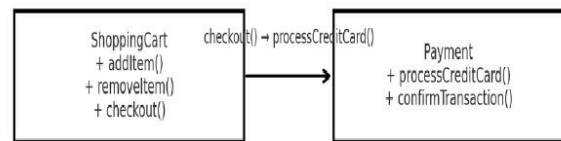
Interaction = object-to-object / class-to-class.

Example (Online Shopping App, Design):

In the **Backend**, you zoom in and design:

- **ShoppingCart** class (methods: addItem(), removeItem(), checkout())
- **Payment** class (methods: processCreditCard(), confirmTransaction())
- So, the **ShoppingCart** calls **Payment.processCreditCard()** when user checkout.
- **So here, we do care about method names, data flow, and logic.**

Detailed Design (Backend Example): Classes & Methods



Extra: you can use the **same diagram types** across both phases In **Architecture** and **Detailed design** but:

- In **Architecture**, they are **high-level** (just showing subsystems, actors, or major flows).
- In **Design**, they are **detailed** (down to classes, methods, attributes, exact interactions).

NOTES

- 1 Detailed design is about **low-level interaction**: how components are implemented inside, shown as classes, methods, data structures and algorithms.
- 2 The interaction is now **object-to-object** or class-to-class.
- 3 Zooming into the backend: a **ShoppingCart** class (`addItem()`, `removeItem()`, `checkout()`) and a **Payment** class (`processCreditCard()`, `confirmTransaction()`). At checkout, ShoppingCart calls `Payment.processCreditCard()` . Here method names, data flow and logic matter.
- 4 You can use the **same diagram types** in both phases (a sequence diagram, for instance). The difference is the level: in architecture they show subsystems, actors and major flows; in design they go down to classes, methods and attributes.

EXAM ANGLE

Prediction

Q1

Give one example of an architecture-level interaction and one of a detailed-design interaction for an online shop.

Answer: Architecture: "the Backend calls the Payment Gateway" (component to component). Detailed design: "ShoppingCart.checkout() calls Payment.processCreditCard()" (object to object, with method names).

Fundamentals of Software Architecture

- From this definition, a few things are of interest and need further explanation:
 - ✓ Foundational design
 - ✓ Transforming requirements (from textual base into a graphical base)
 - ✓ Collection of design elements (diagrams)
 - ✓ Major system components together with their provided quality
 - ✓ Support (an input to) detailed design and construction
- Let's take a more detailed look at these...

NOTES

- 1 Slide 10 explains **foundational design**.
- 2 Slide 11: **transforming requirements** from a textual form into a graphical one.
- 3 Slide 12: a **collection** of design elements (diagrams), not one.
- 4 Slides 13–14: **major components together with their provided quality**.
- 5 Slide 15: architecture **supports** (is an input to) detailed design and construction. These five phrases are exactly what Final 252 Q2-1 asked you to explain (any three).

Fundamentals of Software Architecture

➤ On “ ... **foundational software design**...”

- ✓ Software architecture provides the **groundwork** **”basic”** essential for meeting requirements (**functional** + **non-functional**)
- ✓ This suggest that architecture is **not an optional activity**, it is a major activity.
- New development efforts should approach software architecture as a **forward engineering activity** that leads to the implementation of systems and **Extra :not as a reverse engineering analyzing the system after the system has been built and implemented.**
- ✓ As foundational design, it is where **designing for quality begins**
- ✓ Not considering quality attributes of the system during the software architecture activity can be a **grave mistake!**

Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

10

NOTES

- 1 **Foundational** means architecture provides the groundwork for meeting the requirements, both functional and non-functional. Without it, there is nothing for the rest of the design to stand on.
- 2 So architecture is **not optional**: it is a major activity in every new development effort.
- 3 It should be **forward engineering**: done first, leading to the implementation. The opposite, **reverse engineering**, analyses a system after it has been built to recover its architecture. That is useful for old systems, but it is not how a new one should be designed.
- 4 Because it is foundational, architecture is where **designing for quality begins**. Ignoring quality attributes here is a grave mistake: performance or security that the structure doesn't support is very hard to add later.

EXAM ANGLE

Past exam · Midterm 252 · Q1 MCQ 1

Q1

Which statement best explains why software architecture is considered a foundational design activity?

- A. It focuses on low-level coding decisions
- B. It documents completed systems
- C. It establishes the groundwork for satisfying functional and non-functional requirements
- D. It replaces detailed design activities

Answer: C. It establishes the groundwork for satisfying functional and non-functional requirements — Option b describes reverse engineering, which the slide says architecture should not be.

Fundamentals of Software Architecture

➤ On “...**translates software requirements**...”

- ✓ Software architecture provides **abstractions** for software requirements in the design domain
- ✓ Design elements (diagrams) need to be created so that there is **a mapping between** requirements and design element.
 - When we do this, we **transform requirements** from **textual form into a graphical form**
 - This allows us to provide easy means for **tracing requirements** through the development life-cycle.
- ✓ To create design elements from requirements, it is assumed that requirements are understood, however, this is **not always the case**
 - This suggest that architects must be proficient in activities related to **requirements engineering**.
 - **Extra:** This statement means that before architects can create design elements based on requirements, they need to fully understand those requirements. However, understanding the requirements isn't always straightforward. Therefore, architects need to be skilled in **requirements engineering**, which involves gathering, analyzing, and clarifying what the system needs to do. This ensures that the design aligns with what is actually needed.

NOTES

- 1 Architecture provides **abstractions** of the requirements in the design domain: each requirement is mapped to design elements (components, connectors) that will satisfy it.
- 2 That mapping turns requirements from **text into diagrams**, and it makes requirements **traceable**: for any requirement you can point to the components that implement it, and follow it through to code and tests.
- 3 The mapping assumes the requirements are understood, which is **often not the case**. Architects therefore need to be good at **requirements engineering**: gathering, analysing and clarifying what the system must do. Chapter 2 Lecture 2 covers this.

Fundamentals of Software Architecture

➤ On “...**collection of design elements**...”

- ✓ **No one structure or diagram** can fully describe the software architecture.
 - Think about this: can you evaluate a system’s **usability and performance with one diagram**?
- ✓ This suggests that architectures are composed of **multiple structures** (or diagrams).
- ✓ This is also because architecture design aims for communicating the system structure with different stakeholders. Therefore, **different diagrams are required for different stakeholders**.

Extra:

Why one diagram is not enough in Software architecture?

- ❖ A single diagram cannot show everything (usability, performance, security, data flow, deployment).
- ❖ Different stakeholders (developers, testers, managers, customers) care about different views of the system.
- ❖ That’s why multiple diagrams (structures) are used in software architecture.
- ❖ **Example: Online Shopping App**
 - **Component Diagram** – shows major parts (frontend, backend, database, payment) and their interactions (for architects).
 - **Class/Sequence Diagrams** – show detailed logic and method calls (for developers).
- ❖ In short: Software architecture = collection of diagrams. Each one highlights a different aspect of the system so the right people can understand what they need.

NOTES

- 1 **No single structure or diagram** can describe a whole architecture. Try judging both usability and performance from one picture: a screen-flow diagram says nothing about load, a deployment diagram says nothing about ease of use.
- 2 So an architecture is made of **multiple structures** (diagrams), each highlighting one aspect.
- 3 The second reason is communication: architecture serves stakeholders with different backgrounds and questions, so **different diagrams are needed for different stakeholders**.
- 4 Online shop example: a **component diagram** of frontend, backend, database and payment for architects; class and sequence diagrams of method calls for developers. Together they describe one system.

EXAM ANGLE

Past exam · Midterm 252 · Q1 MCQ 2

Q1

Why are multiple architectural views required in complex systems?

- A. To reduce documentation effort
- B. Because no single view can represent all system aspects for all stakeholders
- C. To replace requirements engineering
- D. To simplify coding

Answer: B. Because no single view can represent all system aspects for all stakeholders

Fundamentals of Software Architecture

➤ On “...**major components of the system, together with their provided quality**...”

- ✓ This means that architectural work focuses on the **major components, the quality properties,** and **services** (functionality) that these components exhibit and provide to other components.

Extra about the Figure:

software architecture, it's not enough to say “this component provides a service” — you also need to describe the quality of that service.

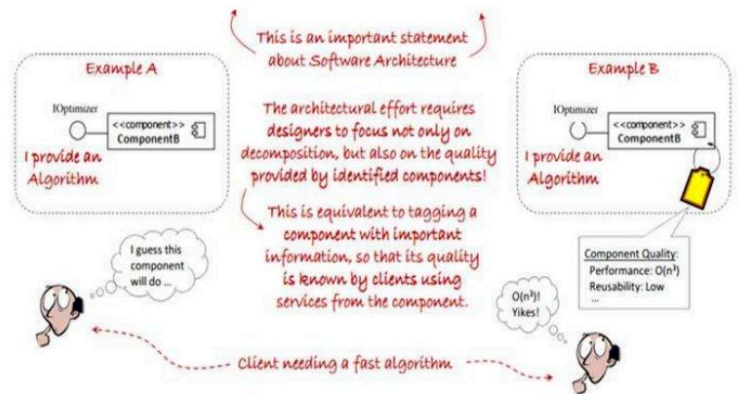
◆ Example A (Left Side)

- Component B provides an algorithm.
- But no quality info is given.
- The client just assumes it's “good enough.”
- Risk: The component might be too slow or not reusable.

◆ Example B (Right Side)

- Same component B, but now tagged with quality properties.
- Example:
 - **Performance: $O(n^2)$** (slow for large inputs).
 - **Reusability: Low.**
- Now the client knows the limitations and may decide not to use it.

◆ Why this matters Architecture is not just about structure, but also about qualities (performance, scalability, security, reliability). Adding quality tags makes components transparent → clients can make enlightened choices.



Extra: When we design software architecture, we don't just list the big parts (components).

We also describe:

Major components – the building blocks (e.g., Frontend, Backend, Database).

Quality properties – how they behave in terms of performance, security, scalability, etc.

Services (functionality) – what they actually provide to other components.

13

NOTES

- 1 Architecture records three things about each major component: the component itself, the **services** (functionality) it provides to others, and the **quality properties** of those services.
- 2 Example A (left): component B provides an algorithm, but nothing is said about its quality. The client just assumes it is “good enough”, and may discover too late that it is too slow or can't be reused.
- 3 Example B (right): the same component tagged with its qualities, e.g. performance $O(n^2)$ (slow for large inputs) and low reusability. Now the client knows the limits and can decide whether to use it. Quality tags make components **transparent** and choices informed.

Fundamentals of Software Architecture

- More on “...**major components of the system, together with their provided quality**...”

✓ Expected quality requirements identified during architecture **trickles down** all the way to the **implementation** of components.

Extra:

1. Assign quality to components

- In the architecture, each component is tagged with expected qualities.
- Example:** Component B must be fast (performance), reliable, and secure.

2. Design & Coding phase

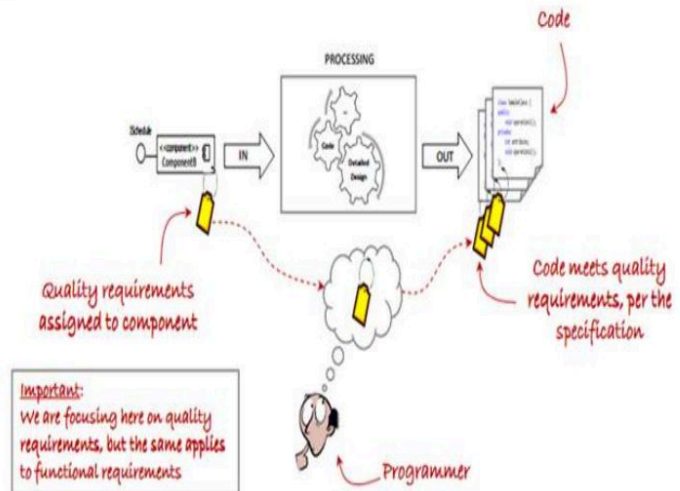
- Programmers take these components into detailed design and coding.
- The quality tags act like requirements guiding how the code should be written.

3. Output (code)

- The produced code should meet the quality requirements (as specified in the architecture).
- Example:** If performance was required, the code should use efficient algorithms.

4. Important note

- The same approach applies not only to quality requirements (e.g., performance, security) but also to **functional requirements** (what the system should do).

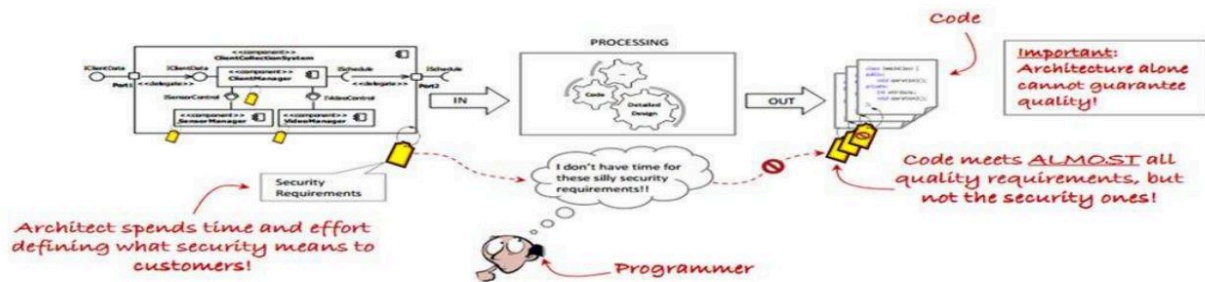


NOTES

- Quality requirements identified at architecture time **trickle down** all the way to the implementation of the components.
- Step 1: in the architecture, each component is **tagged** with its expected qualities (component B must be fast, reliable and secure).
- Steps 2–3: programmers take each component into detailed design and code, using the quality tags as requirements. If performance was required, the code must use efficient algorithms.
- The same flow applies to **functional** requirements: what a component must do also passes from the architecture down to the code.

Fundamentals of Software Architecture

- On “...support detailed design and construction...”
- ✓ Although **architecture** focuses on the quality properties of the system, it must also result in a design that **supports efficient detailed design and construction** of the system.
 - Architects benefit greatly from having **proficiency in general programming design concepts**.
 - ✓ This suggests that architecture alone **cannot guarantee the quality of the system!**



- **Extra:**
- Architect defines **quality requirements** (e.g., security).
- Programmer ignores some (e.g., “no time for security”).
- Final code meets **most qualities** but **misses security**.
- **Result:** Architecture alone ≠ guarantee → qualities must be **implemented in code**.

15

NOTES

- 1 Architecture focuses on quality, but it must also produce a design that **supports efficient detailed design and construction**. A structure that is elegant on paper but awkward to implement fails this test.
- 2 That is why architects benefit from knowing **programming and design concepts** well: they must foresee how their components will be built.
- 3 And architecture alone **cannot guarantee quality**. Example: the architect requires security, a programmer skips it for lack of time, and the final code meets every quality except security. Qualities must be carried through into the code.

EXAM ANGLE

Prediction

Q1

True or false: a well-designed architecture guarantees the quality of the final system.

Answer: False — The qualities must also be implemented in detailed design and code; a programmer can still omit them.

Fundamentals of Software Architecture

➤ To wrap up ...

- ✓ Architectural design should be an **early stage** of the system design process
- ✓ Represents the **link between specification and design processes**
- ✓ Often carried out **in parallel** with some specification activities
- ✓ It involves identifying **major system components** and their **communications** with required **quality attributes**

NOTES

- 1 Architectural design is an **early stage** of system design.
 - 2 It is the **link** between specification (requirements) and design.
 - 3 It often runs **in parallel** with some specification activities: sketching an architecture helps discover and check requirements.
 - 4 It identifies the **major components**, their communication and the quality attributes they must provide.
-

Key Tasks in Architectural Design

➤ Formally, key tasks that need to be performed during the software architecture design effort include:

- Identifying **stakeholders concern** (requirements & quality attributes)
- Identifying appropriate **architectural views**
- Identifying **architectural styles and patterns**
- Identifying **influences** of architectural decisions in organizations
- Identifying the system's **major components** and interfaces
- **Evaluating** and **validating** the architecture
- Establishing policies for ensuring architectural design **synchronicity**.
- **Extra:** **synchronicity means** It means making sure **all parts of the system follow the same design rules** so everything fits together.
- **Example:** If the rule is "use one database system (MySQL)", then all teams must use **MySQL**, not mix MySQL in one module and MongoDB in another.

Extra:

1. Architectural Views

•**Definition:** Different perspectives to describe the system for different stakeholders.

•**Purpose:** Communicate structure and behavior clearly to users, developers, testers, managers, etc.

Example:

Physical (deployment) view: servers, nodes, network.

2. Architectural Styles

Definition: A general template/approach for organizing system structure.

Purpose: Provides a **high-level solution shape**.

Example: **Distributed System**.

3. Architectural Patterns

Definition: Reusable solutions to recurring and repeated architectural problems (more specific than styles).

Purpose: Guide detailed design decisions inside the style.

Examples:

Client –Server, Filters and pipes

17

NOTES

- 1 **Identify stakeholders' concerns:** the requirements and quality attributes each stakeholder cares about (slides 18–23).
- 2 **Identify appropriate architectural views:** the different perspectives used to describe the system to different stakeholders, e.g. the physical (deployment) view of servers and network (slides 24–25).
- 3 **Identify architectural styles and patterns.** A *style* is a general template for organising a system (e.g. distributed system); a *pattern* is a more specific reusable solution to a recurring problem (client-server, pipes and filters).
- 4 **Identify the influences of architectural decisions in organisations:** their effect on budget, schedule and resources.
- 5 **Identify the major components and interfaces.**
- 6 **Evaluate and validate** the architecture (slide 27).
- 7 **Establish policies for design synchronicity**, so every part follows the same design rules. Example: if the rule is "one database system, MySQL", no team quietly uses MongoDB in its module (slide 28).

EXAM ANGLE

Past exam · Final 252 · Q2-2 · 2.5 marks

Q1

List five of the key tasks in Architectural Design (the order is not important).

Answer: Any five of: identify stakeholders' concerns (requirements and quality attributes); identify appropriate architectural views; identify architectural styles and patterns; identify organisational influences of architectural decisions (budget, schedule, resources); identify the system's major components and interfaces; evaluate and validate the architecture; establish policies for design synchronicity.

Past exam · Midterm 261 · Q1

Q2

Which FOUR of the following are key tasks performed during the Software Architecture Design effort? (Options included designing algorithms, unit testing, installing software, writing source code, budgeting and debugging.)

Answer: Identifying architectural styles and patterns; identifying appropriate architectural views; identifying the system's major components and interfaces; identifying stakeholders' concerns and quality attributes. Algorithms, unit tests, installation, maintenance, training, coding, budgets and debugging belong to other phases.

Past exam · Midterm 252 · Q1 MCQ 6

Q3

Which task is part of architectural design?

- A. Writing source code
- B. Identifying stakeholders' concerns
- C. Debugging
- D. Unit testing

Answer: B. Identifying stakeholders' concerns

Key Tasks in Architectural Design

➤ Identifying stakeholders concerns

- ✓ Stakeholders are persons, groups, or organizations that have a **direct** or **indirect** stake or interest in the system.
 - They include systems engineers, software engineers, hardware engineers, project management, customers, testing teams, quality assurance teams, members of the **configuration management team**, etc.
- ✓ A stakeholder's **concern** provides **high-level information about desired characteristics** of the software system.
 - The software architect must ensure that the software to be developed addresses the concerns of all stakeholders." **it considers the needs and interests of everyone involved.**"
 - Stakeholders' concerns serve as **driving force behind architectural decisions (important)**

Extra:

Direct Stakeholders	Indirect Stakeholders
Directly use or operate the system	Do not use it directly, but are affected by it
Drive functional requirements	Drive non-functional / external requirements
Examples: End Users, Admins, Developers, Testers	Examples: Regulators, Investors, Marketing Team, Managers

NOTES

- 1 A **stakeholder** is any person, group or organisation with a direct or indirect stake in the system: systems, software and hardware engineers, project management, customers, testers, QA and configuration management, among others.
- 2 A stakeholder's **concern** is high-level information about a characteristic they want the system to have ("it must never lose a payment", "it must be easy to learn"). The architect must make sure the design addresses the concerns of all stakeholders.
- 3 Concerns are the **driving force behind architectural decisions**: the components and styles are chosen to satisfy them. The slide marks this as important.

Key Tasks in Architectural Design

➤ Identifying stakeholders concerns (cont.)

- ✓ The software architect must identify and understand the different ways stakeholders influence the system.
 - These need to be elicited before any design effort can begin.
- **Extra:** Imagine you're designing a new mobile app for a university. The stakeholders include students, professors, and administrators.
 - **Students** want an easy way to view their schedules and grades.
 - **Professors** need a tool to upload grades and communicate with students.
 - **Administrators** require features to manage courses and student records.
- Before you start designing the app, you need to talk to each group to understand their specific needs and concerns.
- **For example:** Students might want a user-friendly interface, while Administrators are more concerned with security and data management.

NOTES

- 1 The architect must find out how each group of stakeholders influences the system, and those concerns must be **elicited before any design begins**. Designing first and asking later produces the wrong system.
- 2 University app example: **students** want to see schedules and grades easily, **professors** need to upload grades and message students, **administrators** need to manage courses and records.
- 3 Their concerns differ in kind, not just in features: students care about a user-friendly interface (usability), administrators about security and data management. Each concern points the architecture in a different direction.

Key Tasks in Architectural Design

➤ Some important quality attributes of software systems that might concern the stakeholders .

Extra:

None functional Requirements (NFR) Versus Quality Attributes

We can say they are closely related but not exactly the same.

1. **NFR (system-level):**
Describes *how well* the system should perform , so it is a global quality goal (e.g., performance, security).
 2. **Quality Tag (component-level):**
Assigns that goal to a specific component so it's clear what part of the system must meet it.
- **Example** (Online Shopping App)System
 - **NFR (Performance):**
 - ✓ "System must process 1000 orders per second."
 - **Quality Tags:**
 - ✓ **Backend Order Service Component** → Performance ≤ 100 ms per order.
 - ✓ **Database Component** → Performance ≥ 5000 queries/sec.

20

Quality	Description
Usability	The degree of complexity involved when learning or using the system.
Modifiability	The degree of complexity involved when changing the system to fit current or future needs.
Security	The system's ability to protect and defend its information or information system.
Performance	The system's capacity to accomplish useful work under time and resource constraints.
Reliability	The system's failure rate.
Portability	The degree of complexity involved when adapting the system to other software or hardware environments.
Testability	The degree of complexity involved when verifying and validating the system's required functions.
Availability	The system's uptime.
Interoperability	The system's ability to collaborate with other software or hardware systems. Extra: interoperability is when different computer programs or devices can easily work together. For example, Microsoft Office and Google Workspace both can open, edit, and save the same types of files like .docx for documents. This means you can make a file in Microsoft Word and then open and edit it in Google Docs without any problems. This is really helpful for people who use different programs but need to work on the same files.

NOTES

- 1 Quality attributes stakeholders worry about, read as "the degree of complexity involved" or "the system's ability". **Usability:** how hard it is to learn or use. **Modifiability:** how hard it is to change. **Security:** protecting information. **Performance:** useful work under time and resource limits. **Reliability:** the failure rate. **Portability:** how hard it is to move to other environments. **Testability:** how hard it is to verify.
- 2 **Availability** is the system's uptime. **Interoperability** is its ability to work with other systems, e.g. Microsoft Office and Google Docs both opening the same .docx file.
- 3 **NFR vs. quality attribute** (Extra box): closely related but not identical. A non-functional requirement is a system-level goal ("the system must process 1000 orders per second").
- 4 A **quality tag** assigns that goal to a specific component so it is clear which part must meet it ("back-end order service ≤ 100 ms per order", "database ≥ 5000 queries per second").

EXAM ANGLE

Past exam · Final 252 · Q3-1 · 3 marks

Q1

List and define three of the major quality attributes of software architecture.

Answer: Any three, e.g. **Availability:** the system is ready to perform its task when needed (uptime). **Performance:** how quickly it responds to events, its capacity to do useful work under time and resource limits. **Security:** its ability to protect information and resist unauthorised access while serving legitimate users. **Modifiability:** the cost of changing it. **Usability:** how easy it is to learn and use. **Testability:** how easily its faults can be revealed by testing. (Chapter 3 treats these in depth.)

Key Tasks in Architectural Design

- Notice that these quality attributes also describe **high-level information** about desired characteristics of the software system.
 - ✓ In their current form, they are **insufficient to develop** the system.
 - ✓ For a system to exhibit any of these qualities, **design decisions (tactics)** must be made to **support** the achievement of these qualities.

NOTES

- 1 Quality attributes are, like concerns, **high-level** descriptions. "The system must be secure" is true and useless to a developer: in this form they are not enough to build anything.
 - 2 To make a system actually exhibit a quality, the architect makes specific **design decisions, called tactics**: "authenticate every user", "encrypt stored passwords", "run two servers". Tactics turn a wish into something buildable and checkable.
-

Key Tasks in Architectural Design

- The term **tactic** is a design decision that affects the control of a quality attribute response.
- In many cases, these quality attributes are identified through the requirement phase, leaving the architect responsible for specifying how to meet these quality attributes. During this process, **tactics** are identified for each desired quality attribute. (important)

Extra: **Tactic:** A design decision the architect makes to achieve that quality attribute.

In short:

Quality attribute = goal (what we want).

Tactic = design decision (how we achieve it).

Examples

1. **Quality Attribute:** Performance

Tactic: Add caching to reduce response time.

2. **Quality Attribute:** Security

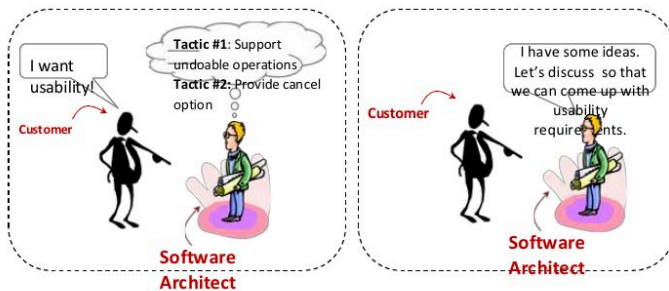
Tactic: Encrypt all sensitive data.

So you can say:

A tactic **supports** a quality attribute.

A tactic **controls** the response of a quality attribute.

A tactic is the way we **achieve** a quality attribute.



22

NOTES

- 1 A **tactic** is a design decision that affects the control of a quality attribute **response**: it changes how the system responds in situations that matter for that quality (attack, overload, failure, change request).
- 2 Quality attributes are usually identified in the requirements phase. The architect's job is to specify **how** to meet them, by choosing tactics for each desired quality. The slide marks this as important.
- 3 In short: the **quality attribute is the goal** (what we want); the **tactic is the design decision** (how we achieve it). A tactic supports, controls and achieves a quality attribute.
- 4 Examples: performance → add caching to reduce response time; security → encrypt all sensitive data; usability (from the cartoon) → support undo and provide a cancel option.

EXAM ANGLE

Past exam · Quiz 1 · 261 · Q16 (fill in)

Q1

Fill in the blank: A design decision intended to influence or control the response associated with a quality attribute is called an architectural ____.

Answer: **Tactic** (singular; "tactics" was marked wrong).

Key Tasks in Architectural Design

Examples on some tactics that relate to the following qualities:

- Tactics for **Security**
 - ✓ **Resisting Attacks**: authenticating users, Limit exposure, Limit access, only on need-to-know basis, etc.
 - ✓ **Detecting Attacks**: intrusion detection
- Tactics for **Testability**
 - ✓ **Event logging**: log data and operations throughout the system. Allow testers to enable/disable this feature so that when enabled, events are displayed in the console to give insight into the system's operations and data.
- Tactics for **Modifiability**
 - ✓ **Localize changes**: modularization, abstraction, encapsulation
 - ✓ **Prevention of ripple effects**: encapsulation, reduce coupling
- Tactics for **Availability**
 - ✓ **Redundancy**, Task monitor, Watchdog timer , etc.
- Tactics for **Performance**
 - ✓ **Increase computation efficiency**, reduce computational overhead, introduce concurrency, etc

Extra: Tactics are specific design decisions that help ensure the software meets certain quality attributes.

•**Security**: To protect the system, we might require user authentication (like passwords) and limit access to sensitive data. **For example**, only admins can access student records.

•**Testability**: By logging events, testers can see what the system is doing. **For example**, if a bug occurs, the logs can help trace what went wrong.

•**Modifiability**: We organize code into modules so that changes in one part don't break others. **For example**, if we update the login feature, it won't affect the grade submission module.

•**Availability**: We use backups and monitoring to ensure the system is always up. **For example**, if one server fails, another can take over without downtime.

•**Performance**: We optimize code to run faster, like reducing unnecessary calculations. **For example**, processing grades quickly even when many students are online.

23

NOTES

- 1 **Security** tactics: *resist attacks* (authenticate users, limit exposure, limit access on a need-to-know basis, e.g. only admins see student records) and *detect attacks* (intrusion detection).
- 2 **Testability** tactic: *event logging*. Log data and operations throughout the system, and let testers switch the logging on to see what the system is doing; when a bug appears, the log shows what led to it.
- 3 **Modifiability** tactics: *localise changes* (modularization, abstraction, encapsulation) and *prevent ripple effects* (encapsulation, reduce coupling). Updating the login feature then doesn't touch grade submission. These are Chapter 1's principles used as tactics.
- 4 **Availability** tactics: *redundancy* (a backup server takes over when one fails), task monitors, watchdog timers.
- 5 **Performance** tactics: increase computational efficiency, reduce computational overhead, introduce concurrency, so grades are processed quickly even when many students are online.

EXAM ANGLE

Prediction

Q1

Name the quality attribute each tactic supports: (a) redundancy, (b) event logging, (c) intrusion detection, (d) reducing coupling.

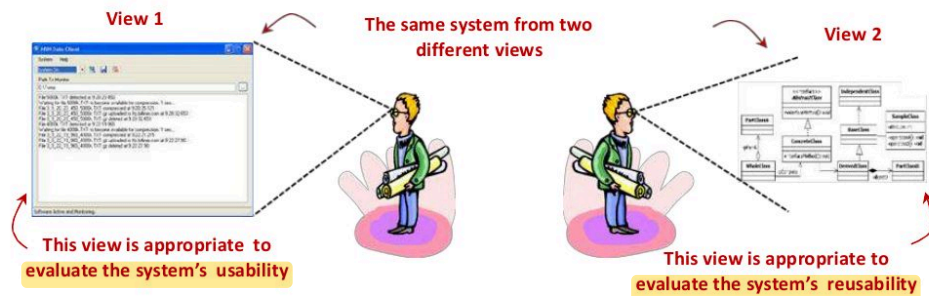
Answer: (a) Availability. (b) Testability. (c) Security (detecting attacks). (d) Modifiability (preventing ripple effects).

Key Tasks in Architectural Design

➤ Identifying appropriate *architectural views*

✓ In complex software systems, there can be **numerous stakeholders** with **different backgrounds**.

- These stakeholders have **different perception** about the system, which influence the way they evaluate the system's design



Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

24

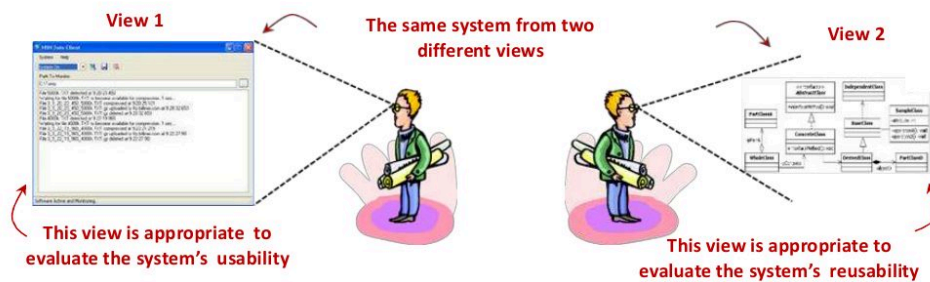
NOTES

- 1 Complex systems have **many stakeholders** with different backgrounds.
- 2 Each has a **different perception** of the system, and that perception shapes how they evaluate its design.
- 3 The two pictures show the same system from two views: View 1 (the screens a user sees) is the right one to judge **usability**; View 2 (the internal components) is the right one to judge **reusability**. Neither view can answer the other's question.

Key Tasks in Architectural Design

➤ Identifying appropriate *architectural views*

- ✓ For this reason, architectural designs must support **different architectural views** used to evaluate the design from a particular **stakeholder's perspective**.
- ✓ An architectural **view** is a **representation of the system**.
 - Different representations are required to **evaluate certain properties** of the system.



Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

25

NOTES

- 1 Hence architectural designs must support **different architectural views**, each used to evaluate the design from one stakeholder's perspective.
- 2 An **architectural view** is a **representation of the system**: one focused picture of it. Different representations are needed to evaluate different properties (performance from a process view, deployment from a physical view). Lecture 3 introduces Kruchten's 4+1 views.

EXAM ANGLE

Prediction

Q1

Define an architectural view and explain why several are needed.

Answer: A view is a representation of the system from a particular stakeholder's perspective. Several are needed because different properties (usability, reusability, performance, deployment) can only be evaluated from the representation that shows them, and stakeholders have different concerns.

Key Tasks in Architectural Design

➤ Identifying architectural styles and patterns

- ✓ They provide an **overall strategy** for designing a family of software systems. (e.g. distributed systems family)
- ✓ They provide **reusable architectural solutions**, documented in a way that is easily understood and applied.
- ✓ Decisions based on styles and patterns benefit from **years of documented experience**.
- ✓ The concept of architectural styles and patterns are fundamental to the **efficient creation** of software architectures.
- ✓ Today, numerous styles and patterns exist so architects must be aware of these so that they can identify and determine the **appropriateness** of a particular style or pattern for their system's design
- ✓ We will discuss major design patterns in more details later in the course
- ✓ Architects must understand the **effect of architectural decisions** on **customers, budget, schedule, and resource availability**

Extra:

- ✓ Architectural styles and patterns are reusable design strategies that guide architects in building efficient systems while balancing customer needs, cost, and schedule.

26

NOTES

- 1 Architectural **styles and patterns** give an overall strategy for designing a whole **family** of systems (e.g. the family of distributed systems).
- 2 They are **reusable solutions**, documented so they are easy to understand and apply, and decisions based on them inherit **years of documented experience** instead of being invented from scratch.
- 3 Many exist, so the architect must know them well enough to judge the **appropriateness** of a style or pattern for a particular system. Chapter 4 covers the major ones.
- 4 Architects must also understand the effect of their choices on **customers, budget, schedule and resource availability**: a pattern can be technically right and still unaffordable.

Key Tasks in Architectural Design

➤ Evaluating and Validating the Architecture

- ✓ Long and iterative process
- ✓ Failure to do so can have significant impact in effort and cost incurred to develop the system.
 - It is well known that defects found earlier on in the development process take much less effort to correct than if found at later stages.
- ✓ The result should determine if the software architecture is sufficiently complete to support the development of the system.
- ✓ We'll have more to say about this later on in the course...

Extra: Evaluating and Validating the Architecture is crucial because it ensures the design is solid before moving forward.

Key Point: Fixing problems early in the process is cheaper and easier than addressing them later.

Outcome: The evaluation confirms whether the architecture is ready to support the system's development.

This process, though lengthy, helps avoid costly mistakes down the line:

Extra Example: Consider a large-scale e-commerce platform like Amazon.

•**Early Evaluation:** During the design phase, architects evaluate the system's ability to handle millions of users simultaneously, ensuring the architecture supports scalability and performance.

•**Result:** If they identify that the database design won't handle peak traffic, they can rework the design early on.

•**Impact:** Catching this issue early avoids major performance problems during high-traffic events like Black Friday. Fixing it later, after the system is live, would be far more costly and disruptive.

27

NOTES

- 1 Evaluating and validating the architecture is a **long and iterative** process: evaluate, fix, evaluate again.
- 2 Skipping it has a large impact on effort and cost, because **defects found early take much less effort to correct** than the same defects found in code or after release.
- 3 The evaluation decides whether the architecture is **sufficiently complete** to support the development of the system.
- 4 Amazon-style example: during design, architects check whether the system can handle millions of simultaneous users. If the database design won't survive peak traffic, they rework it now, not during Black Friday. Chapter 8 covers evaluation methods (ATAM).

Key Tasks in Architectural Design

➤ Introduce policies for design synchronicity

- ✓ **Design synchronicity** is a measurement of the degree of how well the software implementation reflects its design, both in architectural form and detailed design form.
- ✓ Obviously, we want high synchronicity, but this is not always the case.
 - It is very easy to **deviate from design**, especially on **multi-year efforts**.
- ✓ For any software architecture effort to result in successful implementation, all subsequent phases and activities must be synchronized with the architecture.
 - To do this, a **well-defined and understood** process must be in place.
 - This includes the **maintenance phase**, which can go on for years after a software has been deployed!

Extra: Design synchronicity

Meaning: How closely the final software matches the original design.

Example: Initial plan: Banking app with secure login + fund transfer.

During development: Team adds extra features (themes, chat) and changes security without updating design docs.

Result: Final product ≠ original design → low synchronicity.

Best Practice: **Document, review, and approve all changes** → ensures high synchronicity.

28

NOTES

- 1 **Design synchronicity** measures how well the software implementation reflects its design, at both the architectural and the detailed level.
- 2 High synchronicity is the goal, but it is easy to **deviate** from the design, especially in **multi-year** projects where people, priorities and deadlines change.
- 3 Every phase after architecture must stay synchronised with it, which needs a **well-defined, understood process**. That includes **maintenance**, which can last for years after deployment.
- 4 Example: a banking app is designed with secure login and transfers; during development the team adds themes and chat and changes security without updating the design. The product no longer matches the design: low synchronicity. The cure is to **document, review and approve every change**.

EXAM ANGLE

Past exam · Midterm 252 · Q1 MCQ 4

Q1

Design synchronicity ensures that:

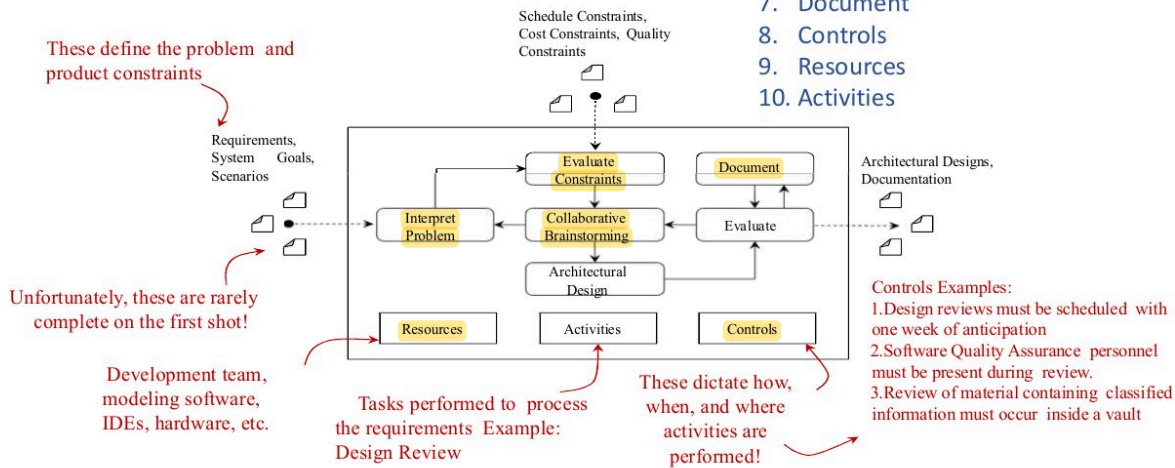
- A. Code runs faster
- B. All phases stay aligned with architecture, especially implementation
- C. Testing is skipped
- D. Documentation is minimized

Answer: B. All phases stay aligned with architecture, especially implementation

Problem Solving During Architecture

Extra:

1. Requirements, System Goals, Scenarios
2. Interpret Problem
3. Evaluate Constraints
4. Collaborative Brainstorming
5. Architectural Design
6. Evaluate
7. Document
8. Controls
9. Resources
10. Activities



Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

29

NOTES

- 1 Chapter 1's holistic problem-solving framework applied to architecture. Inputs (requirements, system goals, scenarios) define the problem, and the architect first **interprets** them. Requirements are rarely complete on the first try, so this step repeats.
- 2 **Evaluate constraints:** schedule, cost and quality constraints bound the solution.
- 3 **Collaborative brainstorming** produces candidate architectures, which are then **evaluated**.
- 4 The output is **documented**: architectural designs and their documentation.
- 5 The process runs on **resources** (the team, modelling software, IDEs, hardware) and is governed by **controls**, rules for how, when and where activities happen. Examples: design reviews scheduled a week ahead, QA staff present at reviews, classified material reviewed inside a vault.

Summary

- This session, we presented **fundamentals concepts** of software architecture and **key tasks** that need to be performed during software architecture, including:
 - ✓ **Identifying stakeholders concerns** (requirements)
 - ✓ Identify architectural views, styles, and patterns
 - ✓ Identify major component's and interfaces
 - ✓ Evaluating and validating the Architecture
 - ✓ Introducing policies for design synchronicity
- We've also mentioned issues with software quality and requirements (but not in details)
- Next session will focus on **requirements engineering**, providing enough information to successfully create good requirements.

Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

30

NOTES

- 1 Summary of the fundamentals: architecture is the foundational activity that turns requirements into a collection of design elements describing the major components and their quality, to support detailed design.
- 2 The key tasks: identify stakeholder concerns; identify views, styles and patterns; identify major components and interfaces; evaluate and validate; introduce policies for design synchronicity.
- 3 The next lecture covers **requirements engineering**, because (slide 11) an architect can only translate requirements that are understood.

Original slides from the SE322 lecture deck, with study notes. Check exact wording against the [original deck](#).
Notes and questions are study aids. Made by Shoug Alomran.