

OVERVIEW The Architecture Process and Requirements Engineering

The architecture process has five steps: understand and evaluate the requirements, design the architecture, evaluate it, document it, and monitor and control its implementation. This lecture covers the first step in depth. Requirements engineering runs four activities (elicitation, analysis, specification and validation), and a requirement worth designing from must be specific, correct, complete, consistent, attainable and verifiable. Every bad-requirement example on the slides fails at least one of those six tests.

Five-step process

Understand requirements → design → evaluate → document → monitor and control implementation.

Four RE activities

Elicit (discover needs), analyse (classify, prioritise, negotiate, model), specify (write down), validate (check with stakeholders).

Six qualities

Specific, correct, complete, consistent, attainable, verifiable. "The system shall be fast" fails specific and verifiable.

Architects live in requirements

Architects go back and forth between requirements and design, and conceptual modelling is often where architecture starts.

HOW THE LECTURE IS ORGANIZED

Slides 1–3 The software architecture process

Slides 4–6 Requirements engineering and elicitation

Slides 7–11 Analysis: classification, prioritisation and negotiation, conceptual modelling

Slides 12–18 Specification and validation; the six characteristics of good requirements

Slide 19 Summary and the 4+1 views next

EXAM FOCUS

Quiz 1 261 drew seven questions from this lecture: elicitation (twice), validation, negotiation, matching the five architecture-process activities to their purposes, architecture evaluation, and whether "the system should respond quickly" is verifiable (it is not).

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, L02

NOTES

- 1 Week 3, second lecture. Lecture 1 listed what an architect does; this one gives the order in which it is done, and then studies the first step, understanding the requirements, in detail.

SLIDE 2 / 19 Lecture Outlines

Lecture Outlines

- **Software Architecture Process**
 - ✓ Understand and Evaluate Requirements
 - ✓ Design the Architecture
 - ✓ Evaluate the Architecture
 - ✓ Document the Architecture
 - ✓ Monitor and control implementation
 - **Requirements Engineering**
 - ✓ Elicitation
 - ✓ Analysis
 - ✓ Specification
 - ✓ Validation
 - **What's next..**
-

NOTES

- 1 Part 1 is the five-step **architecture process** (slide 3).
- 2 Part 2 is **requirements engineering**, the discipline behind step 1, with its four activities: elicitation, analysis, specification, validation.

SLIDE 3 / 19 Software Architecture Process

Software Architecture Process

- On a larger scale, the **process** for creating software architectures can be executed using the following tasks:
- ✓ Understand and evaluate requirements
 - ✓ Design the architecture
 - ✓ Evaluate the architecture
 - ✓ Document the architecture
 - ✓ Monitor and control implementation

NOTES

- 1 **Understand and evaluate requirements:** find the architectural drivers, meaning the requirements, quality attributes and constraints that will shape the structure.
- 2 **Design the architecture:** define the major structures (components, connectors) and the design decisions behind them.
- 3 **Evaluate the architecture:** check whether it satisfies the important requirements and stakeholder concerns before building on it.
- 4 **Document the architecture:** communicate the structures and decisions to everyone who needs them (chapter 9).
- 5 **Monitor and control implementation:** check that the code stays consistent with the architecture. This is design synchronicity from Lecture 1, and it is why architecture work doesn't end when coding starts.

Past exam · Quiz 1 · 261 · Q5 (matching)

Q1

Match each architecture process activity with its main purpose: understand and evaluate requirements; design architecture; document architecture; evaluate architecture; monitor and control implementation.

Answer: Understand and evaluate requirements → **identify architectural drivers and constraints**. Design architecture → **define major structures and design decisions**. Document architecture → **communicate architectural structures and decisions**. Evaluate architecture → **assess whether the architecture satisfies important concerns**. Monitor and control implementation → **check that implementation remains consistent with the architecture**. (Design and document are the pair most often swapped.)

Past exam · Quiz 1 · 261 · Q6 (fill in)

Q2

Fill in the blank: After designing an architecture, the activity used to determine whether it satisfies important requirements and stakeholder concerns is architecture ____.

Answer: Evaluation.

SLIDE 4 / 19 Understand and Evaluate Requirements

Understand and Evaluate Requirements

➤ Requirements engineering

- ✓ The discipline within software engineering that is concerned with the systematic approach to requirements specification, mainly through the following activities:

- Elicitation
- Analysis
- Specification
- Validation

➤ Software architects spend a great deal of time working with software requirements.

- ✓ Even after requirements are specified, software architects find themselves going back and forth between requirements and design.
- ✓ In some cases, architects are completely immersed (**Deep and involved**) in the requirements phase, playing a key role in the specification of requirements.

NOTES

- 1** **Requirements engineering** is the part of software engineering concerned with a systematic approach to requirements: getting them, understanding them and writing them down properly.
- 2** It has four activities, explained on the next slides: **elicitation** (discover), **analysis** (clean up), **specification** (document) and **validation** (confirm).

- 3 Architects spend a lot of time here. Even after requirements are specified they keep going **back and forth** between requirements and design, because designing exposes gaps and conflicts. On some projects they are completely immersed in the requirements phase and help write the specification.

EXAM ANGLE

Prediction

Q1

List the four activities of requirements engineering in order.

Answer: Elicitation, analysis, specification, validation.

SLIDE 5 / 19 Understand and Evaluate Requirements (Elicitation)

Understand and Evaluate Requirements (Elicitation)

➤ Elicitation

- ✓ Activity that deals with identifying stakeholders, uncovering what the customer needs and wants, and with determining the non-functional requirements.
- ✓ Begins by identifying all sources of information that can be used to generate requirements.
- ✓ Different sources can have similar but different visions for the system.
 - Common sources of requirements include:
 - Stakeholders
 - Goals
 - Domain knowledge
 - Operational and organizational environment

NOTES

- 1 **Elicitation** is the discovery activity: identify the stakeholders, uncover what the customer needs and wants, and determine the non-functional requirements (which customers rarely state unprompted).
- 2 It starts by listing **every source of information** that can produce requirements, before collecting a single requirement.
- 3 Sources may have **similar but different visions** of the system: two departments can describe “the same” workflow differently. Spotting that early prevents conflicts later.
- 4 Common sources: **stakeholders**; **goals** (the business objectives the system serves); **domain knowledge** (how banking, medicine or education works); and the **operational and organisational environment** (existing systems, policies, hardware the new system must live with).

Fill in the blank: The requirements activity in which stakeholder needs are discovered using techniques such as interviews and observation is called ____.

Answer: Elicitation: the discovery step, before anything is documented.

SLIDE 6 / 19 Understand and Evaluate Requirements (Elicitation)

Understand and Evaluate Requirements (Elicitation)

➤ Common Elicitation Techniques

- ✓ Interviews
- ✓ Facilitated meetings
- ✓ Observation
- ✓ Scenarios

NOTES

- 1 **Interviews:** ask stakeholders directly, one at a time or in small groups. Good for detail; limited by what people remember to say.
- 2 **Facilitated meetings** (workshops): bring stakeholder groups together with a facilitator so conflicting views surface and get discussed in one place.
- 3 **Observation:** watch users do their current work. It reveals steps and workarounds that people never mention in interviews because they take them for granted.
- 4 **Scenarios:** walk through concrete stories of use ("a student registers for a full course...") to discover what the system must do and how it should respond.

An architect interviews end users, observes their current work, and runs a facilitated workshop to discover system needs. Which requirements activity is being performed?

- A. Negotiation
- B. Architecture documentation
- C. Elicitation
- D. Classification

Answer: C. Elicitation — Interviews, observation and facilitated meetings are elicitation techniques.

SLIDE 7 / 19 Understand and Evaluate Requirements (Analysis)

Understand and Evaluate Requirements (Analysis)

➤ Analysis

- ✓ Requirements are analyzed in their raw form to address and solve issues such as requirements that are *contradicting, incomplete, vague, or just wrong*.
- ✓ **Extra:** For example, if one requirement states the system must operate 24/7, but another says it needs daily maintenance downtime, these are contradictory and need to be resolved.
- ✓ Allows architects to **clear the air** in regard to what needs to be done before devising more detailed designs.
- ✓ The following tasks can be performed during analysis:
 - Requirement **classification**
 - Requirement **prioritization**
 - Requirement **negotiation**
 - Conceptual **modeling**

NOTES

- 1 **Analysis** takes requirements in their raw form and fixes the ones that are **contradictory, incomplete, vague or simply wrong**.
- 2 Contradiction example: one requirement says the system runs 24/7, another needs daily maintenance downtime. Both can't be true as written, so someone must decide.
- 3 Analysis lets architects **clear the air** about what really needs to be done before they invest in detailed designs.
- 4 Its four tasks: **classification** (slide 9), **prioritisation** and **negotiation** (slide 10), and **conceptual modelling** (slide 11).

Extra: Example: Real Software Example: Banking App Development

Imagine a team is developing a mobile banking app, and the initial requirements include:

Requirement	Issue Found	Resolution
R1: Secure login	Vague – “secure” not defined (could mean password, 2FA, fingerprint, etc.).	Clarify the exact login method with stakeholders.
R2: Fast & seamless user experience	May <i>conflict</i> with R1 because strong security can slow down the app.	Optimize design to balance speed and security (e.g., efficient encryption, smart caching).
R3: Multiple payment methods	Incomplete – doesn’t specify which methods (credit card, PayPal, crypto, etc.).	Define the required payment methods clearly.
R4: 24/7 availability	Contradicts the need for system updates/maintenance that may cause downtime.	Plan for rolling updates or redundant servers to keep service available.

NOTES

- 1 R1 Secure login** is *vague*: “secure” could mean a password, two-factor authentication or a fingerprint. Resolution: clarify the exact login method with the stakeholders.
- 2 R2 Fast & seamless experience** may *conflict* with R1, since strong security slows things down. Resolution: a design that balances the two (efficient encryption, smart caching).
- 3 R3 Multiple payment methods** is *incomplete*: which methods (card, PayPal, crypto)? Resolution: define them explicitly.
- 4 R4 24/7 availability** *contradicts* the need for maintenance downtime. Resolution: rolling updates or redundant servers, so updates happen without the service going down. Each row is one of slide 7’s problem types.

EXAM ANGLE

Prediction

Q1

For each banking-app requirement, name the problem analysis would find: (a) “secure login”, (b) “multiple payment methods”, (c) “24/7 availability” alongside planned maintenance downtime.

Answer: (a) Vague: “secure” is undefined. (b) Incomplete: the payment methods aren’t listed. (c) Contradictory: 24/7 conflicts with downtime; resolved with rolling updates or redundant servers.

Understand and Evaluate Requirements (Analysis)

➤ Requirement classification

- ✓ Performed for identifying the nature of each requirement
 - Functional vs. non-functional
 - Product vs. process
 - Imposed vs. Derived

Criteria	Description
Functional vs. Non-Functional	Classification that differentiates between requirements that specify the functional aspects of the system vs. the ones that place constraints on how the functional aspects are achieved.
Product vs. Process	Requirement placed on the system product vs. requirements placed on the process employed to build such product.
Imposed vs. Derived	Requirements imposed by stakeholders vs. requirements that are derived by the development team.

Extra:

1. Functional vs. Non-Functional

- **Functional** → What the system does.
Example: "The system must allow users to log in".
- **Non-Functional** → How the system behaves.
Example: "The system must load within 2 seconds."

2. Product vs. Process

- **Product requirement** → what the delivered software must have or do.
Example: "The app must support online payments."
- **Process requirement** the **way of developing** that product (the steps, methods, or standards used).
Example: "The app must be developed using Agile methodology."

3. Imposed vs. Derived

- **Imposed** → Given directly by stakeholders.
Example: "The app must run on iOS and Android."
- **Derived** → Identified by the development team to meet higher goals.
Example: "To support both platforms, the app should use Flutter framework."

NOTES

- 1 **Classification** identifies the nature of each requirement, along three independent axes.
- 2 **Functional vs. non-functional:** functional says *what* the system does ("users can log in"); non-functional says *how well* it behaves ("the page loads within 2 seconds").
- 3 **Product vs. process:** a product requirement is about the delivered software ("the app supports on-line payments"); a process requirement is about how it is developed ("the app must be developed using Agile").
- 4 **Imposed vs. derived:** an imposed requirement comes straight from stakeholders ("the app runs on iOS and Android"); a derived one is worked out by the team to meet a higher goal ("to support both platforms, use Flutter").

Prediction

Q1

Classify "The app must be developed using the Agile methodology" on the product/process axis, and "To support both platforms we will use Flutter" on the imposed/derived axis.

Answer: The first is a process requirement (about how the product is developed). The second is derived: the team chose it to satisfy the imposed requirement "run on iOS and Android".

Prediction

Q2

"The system must load within 2 seconds" is a:

- A. Non-functional requirement
- B. Functional requirement
- C. Process requirement
- D. Derived requirement

Answer: A. Non-functional requirement

SLIDE 10 / 19 Understand and Evaluate Requirements (Analysis)

Understand and Evaluate Requirements (Analysis)

➤ Requirement prioritization and negotiation

- ✓ Helps identify the most important functions of the software system.
- ✓ When done properly, it can help refine the projected schedule by determining which requirements need to be processed first.
- ✓ Can be used to determine different builds of the software
- ✓ Can help during negotiation when conflicts between requirements are identified.

Extra:

1. **Identify key functions** → e.g., login security is more important than adding a theme color.
2. **Refine schedule** → do critical features first (payment, security) → less urgent features later.
3. **Plan builds/versions** → Release 1: core features, Release 2: chatbot + extras.
4. **Support negotiation** → if conflict → focus on security now, move chatbot to future update.

NOTES

- 1 **Prioritisation** identifies the most important functions: login security matters more than a theme colour.
- 2 Done well, it refines the schedule by deciding which requirements are handled **first**: payment and security early, less urgent features later.

- 3 It also defines the **builds** (releases): Release 1 has the core features, Release 2 adds the chatbot and extras.
- 4 And it supports **negotiation** when requirements conflict: “security now, chatbot in a later update” is a negotiated outcome based on priority.

EXAM ANGLE

Past exam · Quiz 1 · 261 · Q4

Q1

Two stakeholder groups request requirements that cannot both be fully satisfied. Which activity is most directly used to resolve this conflict?

- A. Requirements classification
- B. Code inspection
- C. Requirements negotiation
- D. Requirements elicitation

Answer: C. Requirements negotiation — Classification only groups requirements; negotiation reaches an acceptable trade-off.

SLIDE 11 / 19 Understand and Evaluate Requirements (Analysis)

Understand and Evaluate Requirements (Analysis)

➤ Conceptual modeling

- ✓ **Conceptual models** are created to further identify the requirements by understanding their context, discovering the **bounds of the software system**, and conceptualizing how the system interacts with its environment.
- ✓ In many projects, **this is where architectural design begins**, since system decomposition is essential to developing effective conceptual models. (**important**)

➤ **Extra: Conceptual Modeling**

➤ **Purpose:** Understand requirements, system context, and boundaries.

➤ **Boundaries:** Decide what the system will include and exclude.

✓ **Inside:** login, account balance, money transfer.

✓ **Outside:** ATM hardware, cash withdrawal.

➤ **Shows:** How the system interacts with users and its environment.

➤ **Example: Online banking system** → users log in, check balance, transfer money (but ATMs are not part of this system).

NOTES

- 1 **Conceptual models** help identify requirements by showing their context, the system’s **boundaries**, and how the system interacts with its environment.
- 2 Boundaries decide what is inside and outside the system. Online banking: login, balance and transfers are **inside**; ATM hardware and cash withdrawal are **outside**. Getting this wrong means designing (or forgetting) whole subsystems.

- 3 This is often **where architectural design begins**, because building a good conceptual model already requires decomposing the system into parts. The slide marks this as important.

SLIDE 12 / 19 Understand and Evaluate Requirements

Understand and Evaluate Requirements (Specification & validation)

➤ Specification and validation

- ✓ Activity where the results of elicitation and analysis are **formally captured and documented** in an appropriate format for the use and **review** of all stakeholders.

Extra: Stakeholders review the classified requirements to ensure accuracy, completeness, and feasibility.

- ✓ When specifying requirements, it is important that each requirement exhibit certain characteristics desired for designing successful systems. **Requirements must be:**

Specific

Consistent

Correct

Attainable= "Achievable", Possible"

Complete

Verifiable

NOTES

- 1 **Specification** formally captures the results of elicitation and analysis in a document format that every stakeholder can use and review (the SRS).
- 2 **Validation:** stakeholders review the documented requirements to confirm they are accurate, complete and feasible, i.e. that they really represent their needs.
- 3 Each requirement must have six characteristics: **specific, correct, complete, consistent, attainable, verifiable**. Slides 13–18 take them one at a time.

EXAM ANGLE

Past exam · Quiz 1 · 261 · Q3 (fill in).

Q1

Fill in the blank: Checking whether documented requirements correctly represent stakeholder needs is called requirements ____.

Answer: Validation. (Analysis and correctness were the common wrong answers.)

Prediction

Q2

List the six characteristics a well-specified requirement should have.

Answer: Specific, correct, complete, consistent, attainable, verifiable.

Understand and Evaluate Requirements

➤ On being **specific**,

- ✓ Requirements need to be specified in a **clear, concise** "short and brief," and **exclusive** manner.
- ✓ **Clear requirements** are not open to interpretation; unclear or ambiguous requirements lead to incorrect designs, incorrect implementations, and **deceptive validation** during test.
- ✓ **Concise requirements** are brief and to the point and are therefore easier to understand.
- ✓ **Exclusive requirements** specify one, and **only one thing**, making them easier to verify.

Search the database for what?

Specific	Requirement
No	The software shall search the database.
Yes	The software shall search for a product using the product ID.
	The software shall be secure.
	The software shall authenticate users with user ID and password.
No	The software shall be secure and fast.
Yes	The software shall authenticate users with user ID and password.
	Server acknowledgment message shall be sent within 1/2 second from the time a request is received.

Secure and fast?

What does this mean?

These two requirements are specific and provide the information required to determine what secure and fast mean!

Example of specific and nonspecific requirements:

NOTES

- 1 **Specific** has three parts: the requirement is clear, concise and exclusive.
- 2 **Clear**: not open to interpretation. Ambiguity leads to wrong designs, wrong code, and *deceptive validation*: a test can "pass" against one reading of the requirement while the stakeholder meant another.
- 3 **Concise**: short and to the point, so it is easy to understand.
- 4 **Exclusive**: each requirement states one and only one thing, which makes it easy to verify; a requirement that bundles two things can be half met.

➤ Extra: Specific Requirements Non-Specific Requirements:

1. “The software shall **search for a product using the product ID.**”
✓ **Specific:** Yes, because it clearly states what needs to be searched (product) and how (using product ID).
2. “The software shall **authenticate users with user ID and password.**”
✓ **Specific:** Yes, as it specifies the authentication mechanism (user ID and password).
3. “Server acknowledgment message shall be sent **within 1/2 second** from the time a request is received.”
✓ **Specific:** Yes, it provides a measurable performance requirement (response time within 1/2 second).
4. “The software shall **search the database.**”
✓ **specific:** No, This is vague because it doesn't specify what to search for in the database. It lacks context and detail about the search criteria or the expected outcome.
5. “The software shall **be secure.**”
✓ **specific:** No, Security is a broad concept. This requirement doesn't specify what aspects of security are important, such as whether it refers to data encryption, secure user authentication, or another security measure.
6. “The software shall be **secure and fast.**”
✓ **specific:** No, This requirement combines two qualitative attributes—security and speed—without detailing what “secure” or “fast” means in measurable terms.

NOTES

- 1 Specific examples: searching **by product ID**, authenticating **with user ID and password**, acknowledging **within ½ second**. Each says exactly what, how, or how much.
- 2 “The software shall search the database” is not specific: search for what, by which criteria, returning what?
- 3 “The software shall be secure” is not specific: security covers encryption, authentication, access control... which one is required?
- 4 “Secure and fast” fails twice: two vague qualities in one sentence (not exclusive), with no measurable meaning for either.

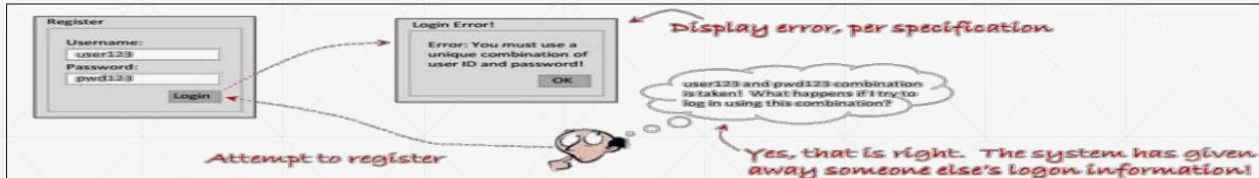
EXAM ANGLE**Prediction****Q1**

Rewrite “The software shall be secure and fast” as specific requirements.

Answer: Split it and make each measurable, e.g. “The system shall authenticate users with a user ID and password and lock the account after 5 failed attempts” and “The system shall return search results within 2 seconds for 95% of requests.”

Understand and Evaluate Requirements

- On being **correct**,
 - ✓ Requirements need to be correct in the sense that they must **accurately describe a desired system function**.
 - ✓ Incorrect requirements can lead to **incorrect or undesired behavior**.
- The diagram you're describing seems to illustrate a sequence of interactions around user registration and login for a software system, highlighting an error handling scenario where a username and password combination is not unique. Here's a detailed breakdown based on the described elements:

**Extra: Scenario Overview:****1. Attempt to Register:**

A user tries to register with a username and password via a registration form.

2. Error Display:

The system checks if the username and password combination is unique.

Since the combination is already taken, the system generates an error message stating, "Error! You must use a **unique combination of user ID and password!**"

3. User's Reaction:

This prompts the user to either choose a different username or password to ensure uniqueness.

NOTES

- 1 A requirement is **correct** when it accurately describes a function the stakeholders actually want. A wrong requirement, implemented perfectly, still gives wrong or undesired behaviour.
- 2 The cartoon is an incorrect requirement in action. The specification said each **user ID + password combination must be unique**, so registering as user123/pwd123 is rejected with "you must use a unique combination". But that error tells the new user that someone else already has exactly that username and password: the system has just given away another person's login. The code did what the requirement said; the requirement was wrong. (Usernames should be unique; passwords should never be compared across users.)

EXAM ANGLE

Prediction

Q1

Explain why "each username and password combination must be unique" is an incorrect requirement.

Answer: Enforcing it means telling a registering user that their chosen combination already exists, which discloses another user's valid credentials. The requirement describes undesired behaviour; the correct one is that usernames are unique.

Understand and Evaluate Requirements

➤ On being **complete**,

- ✓ Requirements must be complete both **individually** and as **collective set**.
 - This means that each requirement should be specified thoroughly so that it absolutely describes the functions required to meet some need.
 - Collectively, requirements need to provide complete specification of the software's required functionality in the **software requirements specification (SRS)**.
 - **Extra:**
 - Individually → each requirement should be fully detailed and clear.
 - Collectively → together, they should describe the entire functionality in the Software Requirements Specification (SRS).
- ✓ **Incomplete requirements lead to incomplete designs**, which in turn leads to incomplete construction of the software system.

Notice how the original requirement is broken into two complete requirements!

Complete	Requirement
No	The software shall generate product reports.
Yes	The software shall generate product reports consisting of product description, picture, and price.
	Product reports shall be in PDF format.

This is a good requirement in the sense that it is specific and correct.

However, if we know the requirements for product reports, then a complete version of this requirement specifies this information. This help disambiguate the notion of a product report so that it can be implemented easily in code.

NOTES

- 1 **Complete** applies at two levels. Individually, each requirement is specified thoroughly enough to describe fully the function it asks for.
- 2 Collectively, the whole set in the **SRS** (software requirements specification) must cover all the required functionality: nothing missing, everything specified. A perfect login requirement doesn't make the SRS complete if account recovery was never written down.
- 3 Incompleteness propagates: incomplete requirements → incomplete design → incomplete construction.

Understand and Evaluate Requirements

- On being **consistent**,
 - ✓ Requirements are consistent when they **do not preclude** “prevent” the design or construction of other requirements.
- On being **attainable**,
 - ✓ Requirements that are unattainable **serve no purpose**.
 - ✓ Attainability can be determined for both product and process.
 - ✓ Requirements are attainable if they can realistically be achieved within the project constraints, which include **time, technology, budget, and resources**.

Extra:

Consistent

- No conflicts between requirements.
- ✓ Example: “System must support online payments” & “System must support credit card payments” (work together).
- ✗ Conflict: “System must be offline only” & “System must support online payments.”

Attainable

- Realistic within time, budget, technology, resources.
- ✓ Example: “App must handle 1,000 users at once.”
- ✗ Not attainable: “App must handle 1 billion users on a \$500 budget.”

💡 **Tip:** Good requirements = **clear, non-conflicting, and realistic**.

Attainable	Requirement
No	The software shall execute on all future operating systems.
Yes	The software shall execute on the Microsoft Windows 7 platform.

NOTES

- 1 **Consistent:** requirements don’t preclude (prevent) the design or construction of other requirements, i.e. no conflicts.
- 2 Consistent pair: “support online payments” and “support credit-card payments” work together. Conflicting pair: “the system must be offline only” and “the system must support online payments”.
- 3 **Attainable:** unattainable requirements serve no purpose. A requirement is attainable if it can realistically be achieved within the project’s constraints of time, technology, budget and resources, and this applies to both product and process requirements.
- 4 Attainable: “handle 1,000 simultaneous users”. Not attainable: “handle 1 billion users on a \$500 budget”.

Understand and Evaluate Requirements

Extra: A requirement is verifiable if there is a suitable way to measure or test it to confirm that the requirement has been met. This could be through tests, demonstrations

- On being **verifiable**,
 - ✓ Perhaps the **most obvious desirable** characteristic of requirements.
 - ✓ Requirements that cannot be verified **cannot be claimed as met**.
 - ✓ Inability to verify requirements point to a **serious flaw early on** in the development process

Verifiable	Requirement
No	The system shall maximize communication speed.
Yes	The system's data rate shall be no less than 1Mbps.

•**Extra:**

-  **Verifiable Requirements**
- **Good requirement** = measurable, testable, clear.
- **Bad requirement** = vague, not testable.
- **Examples:**
 -  **Not verifiable:** "The system shall maximize speed."
 -  **Verifiable:** "The system's data rate shall be at least **1 Mbps**."
-  **Tip:** Always make requirements measurable → use **numbers, limits, or conditions**.

NOTES

- 1 **Verifiable** is perhaps the most obvious desirable characteristic: there must be a suitable way (test, measurement, demonstration) to confirm the requirement has been met.
- 2 A requirement that can't be verified can't be claimed as met, and being unable to verify a requirement points to a serious flaw early in development: nobody knows what "done" means.
- 3 "The system shall maximise speed" is not verifiable (maximise compared with what?). "The system's data rate shall be at least 1Mbps" is. Rule of thumb: use numbers, limits or conditions.

EXAM ANGLE

Past exam · Quiz 1 · 261 · Q15 (true/false)

Q1

The requirement "The system should respond quickly" is sufficiently specific and verifiable for architectural evaluation.

Answer: False — "Quickly" is vague unless replaced by a measurable criterion such as "within 2 seconds for 95% of requests".

Summary

- In this session, we presented fundamentals concepts of **requirements engineering**, and provided enough information to successfully **create good** requirements.
 - In the next session, we'll discuss **how to design the software architecture using the popular 4+1 View Model**, which includes:
 - ✓ Logical view
 - ✓ Development View
 - ✓ Process View
 - ✓ Physical View
 - ✓ User View
-

NOTES

- 1 This lecture gave the basics of requirements engineering, enough to write good requirements: elicit, analyse, specify and validate, then check each requirement against the six characteristics.
- 2 Next: designing the architecture with the **4+1 view model**: logical, development, process and physical views, plus the user (scenario) view that ties them together.

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.