Software Engineering CT:03 Understanding

Created by	Ⓑ Borhan
Last edited time	@May 12, 2025 7:41 PM
Tag	Year 3 Term 2

References:

- Design Concept : https://www.youtube.com/watch?v=3qJtbDOht5A

Risk Management

Slide 1: Risk Management and Software Testing

What's the Vibe?

This slide is just the title, setting the stage. It's like the opening scene of a movie telling us we're diving into how to handle risks in software projects and why testing matters. Risks are those sneaky things that can mess up your project, and testing helps catch them before they ruin everything.

In Bangla:

এটা শুধু টাইটল স্লাইড। বলছে আমরা সফটওয়্যার প্রজেক্টে রিস্ক কীভাবে ম্যানেজ করব আর টেস্টিং কেন জরুরি। রিস্ক মানে যেসব জিনিস প্রজেক্ট নষ্ট করতে পারে।

Slide 2: Source of Risk

What's Risk?

Risk is like when you're not sure if your *biryani* will turn out spicy enough, and there's a chance it'll be a disaster. Formally, it's an "uncertain future event" that

might happen and cause problems (loss). In software, risks come from things going wrong.

Sources of Risk (Where Trouble Starts):

1. **Misunderstanding Customer Requirements**: Like if a client wants a freelancing app like Upwork, but you think they just want a simple job board.
2. **Constantly Changing Requirements**: Client keeps saying, "Add this, change that," making your life chaotic.
3. **Unrealistic Promises**: Promising the app in 2 weeks when it needs 2 months. Oops!
4. **New Methodology Confusion**: Using a new tool (like Laravel for the first time) without fully getting it.
5. **Bad Design Assumptions**: Thinking your app's design can handle 10,000 users, but it crashes at 100.
6. **Teamwork Misjudgments**: Assuming your team will work like superheroes, but someone's slacking.
7. **Wrong Budget**: Underestimating costs, so you run out of money halfway.

Example: For your Laravel freelancing app, a risk could be misunderstanding that clients want social media logins, or promising a payment system in a week when integrating APIs takes longer.

In Bangla:

রিস্ক মানে অনিশ্চিত কিছু যা প্রজেক্টে সমস্যা করতে পারে। যেমন, তুই যদি ভাবিস ক্লায়েন্ট শুধু জব বোর্ড চায়, কিন্তু তারা ফ্রিল্যান্সিং অ্যাপ চায়। বা, তুই বললি ২ সপ্তাহে অ্যাপ দেবি, কিন্তু লাগবে ২ মাস। এগুলো রিস্কের উৎস।

Slide 3: Risk Management

What's the Deal?

Risk management is about planning ahead to avoid or handle these risks. It's like checking the weather before a picnic to bring an umbrella just in case. You identify risks, figure out how likely they are, and decide how bad they'd be if they happen.

Why It Matters:

It’s a key part of project planning to keep your freelancing app on track, so you don’t end up with a buggy app or angry clients.

In Bangla:

রিস্ক ম্যানেজমেন্ট মানে আগে থেকে প্ল্যান করা যাতে সমস্যা এড়ানো যায়। যেমন, পিকনিকে যাওয়ার আগে আবহাওয়া চেক করা। তুই রিস্ক খুঁজবি, দেখবি কতটা সম্ভাবনা, আর কতটা খারাপ হতে পারে।

Slide 4: Risk Identification

What’s This About?

This is about spotting risks early, like noticing your teammate’s not great at coding APIs before they mess up the payment system. The earlier you find risks, the better you can plan to avoid them.

Methods to Find Risks:

- **Brainstorming**: Sit with your team and list what could go wrong (e.g., “What if the client changes the design?”).
- **SWOT Analysis**: Look at Strengths, Weaknesses, Opportunities, Threats (e.g., Weakness: new to Laravel).
- **Causal Mapping & Flowcharts**: Map out how one problem (e.g., delayed API integration) could lead to others (e.g., late delivery).

Types of Risks:

1. **Technology Risks**: Using new tools like Laravel or a database that might not work as expected.
2. **People Risks**: Team members leaving or not having enough skills.
3. **Organizational Risks**: Company issues, like budget cuts or bad management.
4. **Tools Risks**: Bugs in your development software (e.g., VS Code crashing).
5. **Requirement Risks**: Client changing what they want mid-project.
6. **Estimation Risks**: Guessing wrong about time or resources needed.

Example: For your freelancing app, a technology risk could be Laravel’s authentication system being tricky to set up, or a people risk could be a teammate not knowing how to integrate payment APIs.

In Bangla:

রিস্ক আইডেন্টিফিকেশন মানে সমস্যা আগে থেকে খুঁজে বের। যেমন, তুই টিমের সাথে বসে লিস্ট কর কী কী সমস্যা হতে পারে (যেমন, ক্লায়েন্ট ডিজাইন বদলাতে পারে)। রিস্কের ধরন: টেকনোলজি (নতুন টুল), পিপল (টিমের সমস্যা), অর্গানাইজেশন (কোম্পানির ইস্যু), টুলস (সফটওয়্যার বাগ), রিকোয়ারমেন্ট (ক্লায়েন্টের চেঞ্জ), এস্টিমেশন (ভুল হিসাব)।

Slide 5: Risk Analysis and Projection (or Prioritization)

What's Happening Here?

Once you've listed risks, you analyze them to see:

1. What's the problem (e.g., delayed API integration)?
2. How likely is it to happen (probability)?
3. How bad will it be (impact)?

Probability Categories:

- **Very Low (0-10%)**: No biggie, like a minor UI glitch.
- **Low (10-25%)**: Small issue, like a short delay.
- **Moderate (25-50%)**: Affects timeline, like missing a milestone.
- **High (50-75%)**: Hurts time and budget, like major bugs.
- **Very High (75%+)**: Total disaster, like the app failing to launch.

Example: For your app, a risk like "Payment API fails" might have:

- **Probability**: High (50-75%), if you're new to APIs.
- **Impact**: Very High, as users can't pay, killing the app's purpose.

In Bangla:

রিস্ক এনালাইসিস মানে দেখা সমস্যা কী, কতটা হওয়ার সম্ভাবনা, আর কতটা খারাপ হবে। যেমন, পেমেন্ট এপিআই ফেল করার সম্ভাবনা ৫০-৭৫%, আর ইম্প্যাক্ট অনেক বেশি, কারণ ইউজার পে করতে পারবে না।

Slide 6: Risk Control

What's Risk Control?

This is about managing risks to keep your project on track, like having a backup plan if it rains during your picnic. It includes **Risk Planning** with three strategies:

1. **Avoid the Risk**: Change plans to dodge the risk (e.g., negotiate simpler requirements with the client).

2. **Transfer the Risk**: Let someone else handle it (e.g., hire a third-party for payment API integration).
3. **Reduce the Risk**: Plan for losses (e.g., hire extra developers in case someone leaves).

Example: For your app:

- **Avoid**: Promise 30 days instead of 20 to avoid late delivery.
- **Transfer**: Use a service like Stripe for payments to avoid building it yourself.
- **Reduce**: Train multiple team members on APIs so one person leaving doesn't stop progress.

In Bangla:

রিস্ক কন্ট্রোল মানে রিস্ক ম্যানেজ করে প্রজেক্ট ঠিক রাখা। তিনটা উপায়:

- এড়ানো: ক্লায়েন্টের সাথে সহজ রিকোয়ারমেন্ট ঠিক করা।
- ট্রান্সফার: পেমেন্ট এপিআই তৃতীয় পক্ষের কাছে দেওয়া।
- কমানো: একাধিক লোককে ট্রেনিং দেওয়া যাতে কেউ চলে গেলেও কাজ চলে।

Slide 7: Risk Control (Continued)

Risk Monitoring:

- Keep checking risks throughout the project, like checking the weather daily during a week-long event.
- If assumptions change (e.g., client adds new features), update your plans.

Risk Resolution:

- Fix risks when they happen to keep the project on track.
- Depends on good identification, analysis, and planning.

Example: If your API integration is delayed (a risk), monitor progress weekly. If it's behind, resolve by assigning extra developers or extending the deadline.

In Bangla:

মনিটরিং: প্রজেক্ট চলাকালীন রিস্ক চেক করা, যেমন প্রতিদিন আবহাওয়া দেখা।

রেজোলিউশন: রিস্ক হলে ঠিক করা, যেমন এপিআই দেরি হলে আরো ডেভেলপার লাগানো।

Slide 8: RMMM Plan

What’s RMMM?

RMMM stands for **Risk Mitigation, Monitoring, and Management Plan**. It’s a detailed plan the project manager uses to handle risks, included in the overall project plan. Risks are tracked using a **Risk Information Sheet (RIS)**, which logs:

- Risk ID, date, probability, impact, description, avoidance strategies, monitoring plan, management plan, current status.

The RIS is stored in a database for easy searching and prioritizing.

Example: For your app, an RIS entry might be:

- **Risk**: Payment API failure.
- **Probability**: 60%.
- **Impact**: Stops payments.
- **Avoidance**: Use Stripe.
- **Monitoring**: Weekly API tests.
- **Management**: Switch to PayPal if Stripe fails.

In Bangla:

RMMM মানে রিস্ক মিটিগেশন, মনিটরিং, ম্যানেজমেন্ট প্ল্যান। রিস্ক ইনফরমেশন শিটে রিস্কের বিস্তারিত থাকে, যেমন পেমেন্ট এপিআই ফেল করার সম্ভাবনা ৬০%, এড়াতে Stripe ব্যবহার, মনিটরিং সপ্তাহে টেস্ট।

Slide 9: Risk Mitigation

What’s Mitigation?

Mitigation is proactively avoiding risks before they happen, like wearing a helmet to avoid injury. Steps include:

1. Talk to staff to spot risks.
2. Remove causes (e.g., unclear requirements).
3. Set policies (e.g., regular team meetings).
4. Control documents (e.g., track requirement changes).
5. Conduct reviews to speed up work.

Example: For your app, mitigate the risk of requirement changes by having weekly client meetings to clarify needs and documenting every change.

In Bangla:

মিটিগেশন মানে রিস্ক হওয়ার আগে এড়ানো। যেমন, ক্লায়েন্টের সাথে সপ্তাহে মিটিং করে রিকোয়ারমেন্ট ক্লিয়ার করা, ডকুমেন্ট রাখা।

Slide 10: Risk Monitoring

What's This?

Monitoring tracks project progress to catch risks early. It's done by the project manager. The slide cuts off, but objectives include ensuring risks are managed and the project stays on track.

Example: For your app, monitor by checking weekly if the payment API integration is on schedule. If it's lagging, act fast (e.g., get help from a senior developer).

In Bangla:

মনিটরিং মানে প্রজেক্টের অগ্রগতি চেক করা। যেমন, সপ্তাহে দেখা পেমেন্ট এপিআই কাজ ঠিক চলছে কি না।

Slide 11: Risk Management

What's Risk Management (Reactive)?

This is fixing risks *after* they happen, assuming mitigation failed. The project manager steps in to solve issues.

Example: If a key developer leaves your app project:

- **Mitigation Failed**: You trained backups, but they're not ready.
- **Management**: Use documentation to onboard new developers, hire quickly, or redistribute tasks.

If you have good mitigation (e.g **sufficient staff, good documentation, team knowledge), new developers can pick up fast.

In Bangla:

রিস্ক ম্যানেজমেন্ট মানে রিস্ক হয়ে গেলে ঠিক করা। যেমন, ডেভেলপার চলে গেলে নতুন লোক নিয়ে, ডকুমেন্টেশন দিয়ে কাজ শুরু করা।

Slide 12: RMMM Plan - Example

Example Scenario: Late delivery of your freelancing app.

- **Mitigation**: Say it'll take 30 days, though you expect 20, to avoid delays.
- **Monitoring**: Set a schedule with start/end dates, track progress.
- **Management**: If delayed, negotiate with the client for extra time or add features to satisfy them.

In Bangla:

যেমন, অ্যাপ লেট হওয়ার রিস্ক:

- মিটিগেশন: ২০ দিন লাগলেও ৩০ দিন বলা।
- মনিটরিং: শিডিউল ফলো করা।
- ম্যানেজমেন্ট: লেট হলে ক্লায়েন্টের সাথে নতুন ডেট বা ফিচার দেওয়ার কথা বলা।

Slide 13: Drawbacks of RMMM

What's the Downside?

- **Cost**: RMMM takes money (e.g., hiring extra staff).
- **Time**: Planning and monitoring slow things down.
- **Complexity**: For big projects, RMMM can feel like its own project.
- **No Guarantee**: Risks can still pop up after delivery (e.g., bugs found later).

Example: Your app's RMMM plan might cost extra for backup developers, take time for weekly reviews, and still face post-launch bugs.

In Bangla:

RMMM-এর সমস্যা: খরচ বেশি, সময় লাগে, বড় প্রজেক্টে জটিল, আর লঞ্চের পরেও রিস্ক থাকতে পারে।

Why This Matters for You

Since you're working on a Laravel freelancing app (from our past chats), RMMM is super relevant. It helps you:

- Avoid delays by planning extra time.
- Handle client changes with clear communication.
- Mitigate team risks with backups or documentation.

- Monitor progress to hit deadlines.

It’s like having a playbook to keep your project smooth, so you can impress your profs tomorrow! 😎

In Bangla:

তোর ফ্রিল্যান্সিং অ্যাপের জন্য RMMM জরুরি। এটা দিয়ে তুই ডেডলাইন মিস এড়াতে পারবি, ক্লায়েন্টের চেঞ্জ ম্যানেজ করতে পারবি, টিমের সমস্যা কমাতে পারবি।

Extra Tips for Your Presentation

Since you’re showing progress tomorrow (like you mentioned for your Laravel project), here’s how to use this:

- **Highlight Key Risks**: Mention risks like API integration or requirement changes, and how you’ll mitigate them (e.g., using Stripe, weekly client calls).
- **Show an RIS**: Sketch a simple Risk Information Sheet for your app (e.g., Risk: API Failure, Probability: 60%, Mitigation: Use Stripe).
- **Be Confident**: Explain how RMMM keeps your project on track, tying it to your Laravel work.

In Bangla:

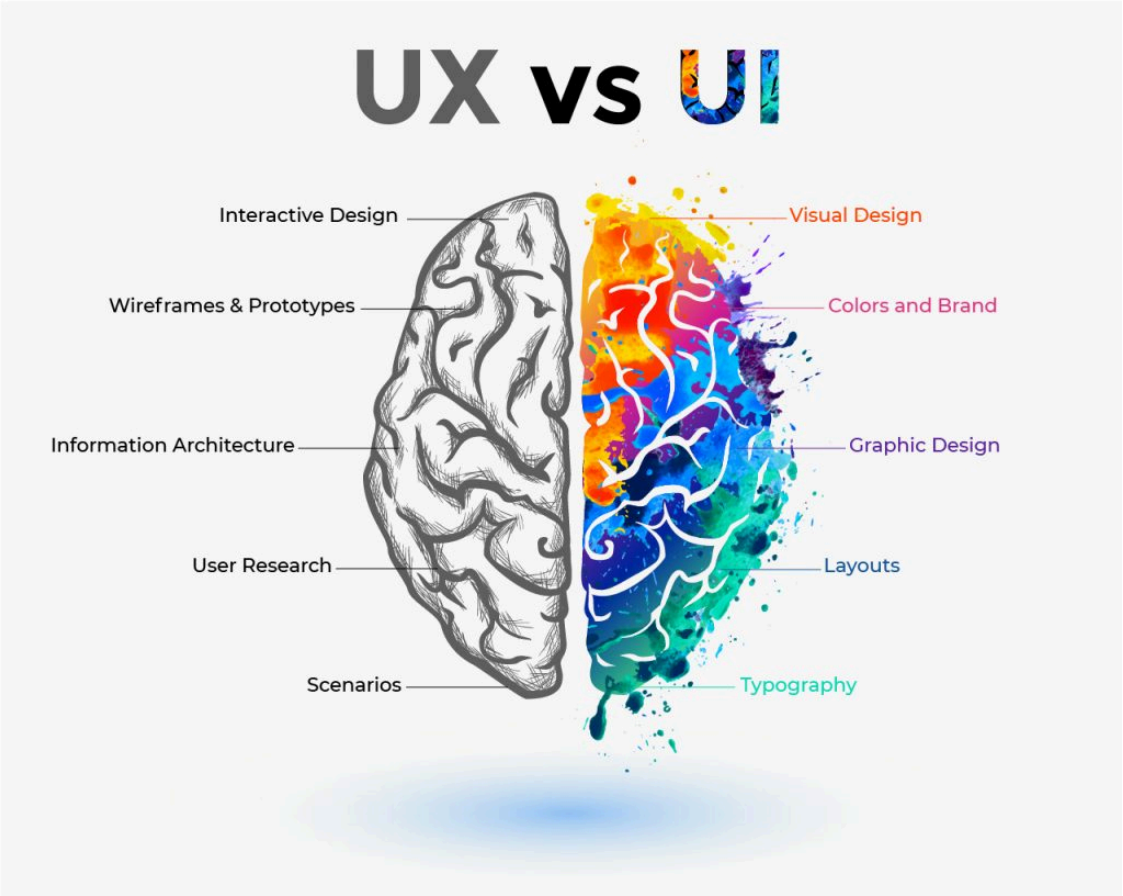
কালকের প্রেজেন্টেশনে এপিআই ফেল, রিকোয়ারমেন্ট চেঞ্জের রিস্ক বল। RIS দেখা (যেমন, এপিআই ফেল, ৬০%, মিটিগেশন: Stripe)। RMMM দিয়ে প্রজেক্ট কীভাবে ঠিক রাখবি, তা বল।

Design Concepts

Software Design

- an iterative process to transform user r**equirements into some suitable form (blueprint)** which helps the programmer in software coding and implementation
- first step in SDLC
- specify how to fulfill the requirements mentioned in SRS
- Software Design Engineer or UI/UX designer

UI (User Interface) designers focus on the **visual elements and interactions** of the interface, while UX (User Experience) designers focus on the o**verall experience** of the user, including their **emotions and perceptions.**



From : https://cole.edu.vn/wp-content/uploads/2024/05/3-1.jpg

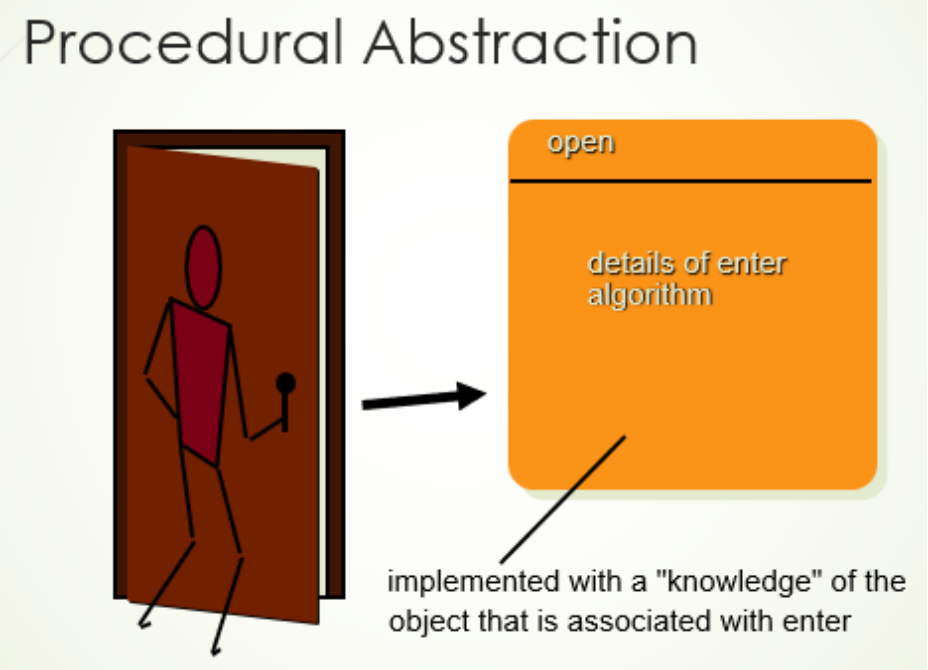
Abstraction

- Abstraction is used to **hide background details** or unnecessary implementation about the data.
- Abstraction is the **act of representing essential features without** including the **background details** or explanations
- **reduce complexity** and **allow efficient design and implementation**
- **Levels:**
    - **Highest Level of Abstraction** :a solution is stated in broad terms using the language of the problem environment.

- **Lowest Level :** a more detailed description of the solution is provided

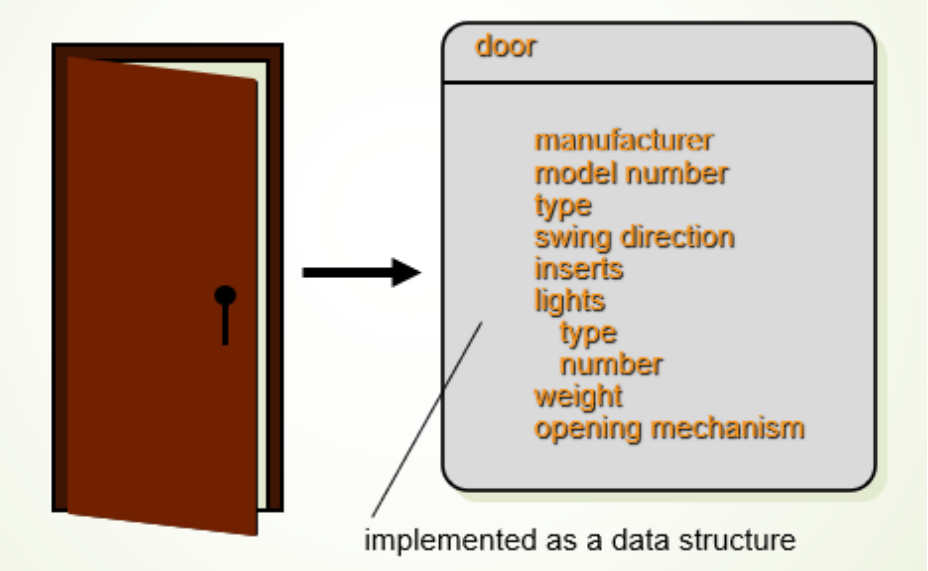
- **Procedural abstraction:** A procedural abstraction refers to a **sequence** of instructions that have a **specific and limited function.**
  - There is a collection of subprograms.
  - One is hidden group, another is visible group of functionalities.

  Example : Open for a door. Open implies a long sequence of procedural steps : Walk to the door, reach out and grasp knob, turn knob and pull door and step away from moving door, etc.



- **Data abstraction:** A data abstraction is a named **collection of data** that describes a **data object.**
  - Show representation data and hide manipulation data.

  Example: The data abstraction for **door** would encompass a set of attributes that describe the door (e.g, door type, swing direction, opening mechanism, weight, dimensions)



Architecture

- Software architecture is the structure or organization of program components (modules), the manner in which these components interact and the structure of data that are used by components
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- Marry Shaw and David Garlan (Sha95a) describe a set of properties that should be specified as part of an architecture design
    - **Structural Properties :** This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.
    - **Extra-Functional Properties:** The architectural design description should address **how the design architecture achieves requirements** for **performance, capacity, reliability, security, adaptability**, and other **system** characteristics.
    - **Families of Related Systems:** The architectural design should draw upon **repeatable patterns that are commonly** encountered in the design of families of **similar systems**.

Understand in a better way:

Software architecture is like the blueprint of a house. When you want to build a house, you first create a plan that shows how the rooms, walls, doors, and windows will be arranged. Similarly, software architecture is the plan for a software system. It defines:

- **How the software is structured**: What are the main parts (like modules or components)?
- **How these parts work together**: How do they communicate or interact?
- **What data they use**: How is the information organized and managed?

In Bangla:

সফটওয়্যার আর্কিটেকচার হলো একটা সফটওয়্যার সিস্টেমের মূল কাঠামো বা নকশা। এটা বলে দেয় সফটওয়্যারের বিভিন্ন অংশ (যেমন মডিউল) কীভাবে গঠিত হবে, তারা একে অপরের সাথে কীভাবে কাজ করবে, এবং কীভাবে ডেটা ব্যবহার করবে। এটা একটা বাড়ির নকশার মতো, যেখানে সবকিছু পরিকল্পনা করে রাখা হয়।

The quote you provided says:

> "the overall structure of the software and the ways in which that structure provides conceptual integrity for a system."

Conceptual integrity means that the software's design makes sense as a whole. All parts fit together logically, like pieces of a puzzle, to achieve the system's purpose without confusion or chaos. For example, if you're building an app like a messaging platform, the architecture ensures that the part handling messages, the part storing user data, and the part showing the user interface all work together smoothly.

Key Properties of Software Architecture

The authors Shaw and Garlan describe three important aspects that a good architectural design should cover. Let's dive into each one with examples and explanations in simple language, with Bangla translations for clarity.

1. Structural Properties

This is about **what the system is made of** and **how its parts are connected**.

- **Components**: These are the **building blocks of the software**, like modules, objects, or filters. Think of them as individual rooms in a house (e.g., kitchen, bedroom).
- **Packaging and Interaction**: This defines how these **components are grouped** and how they "talk" to each other. For example, in a web app, one component might handle user login, another might display the homepage, and they interact by passing data like user credentials.

Example: Imagine a food delivery app. The structural properties would define:

- A module for showing the menu.
- A module for processing orders.
- How the menu module sends the selected items to the order module.

In Bangla:

কাঠামোগত বৈশিষ্ট্য বলতে বোঝায় সিস্টেমের বিভিন্ন অংশ (যেমন মডিউল বা অবজেক্ট) এবং সেগুলো কীভাবে একত্রিত হয়ে কাজ করে। উদাহরণস্বরূপ, একটা ফুড ডেলিভারি অ্যাপে মেনু দেখানোর একটা অংশ থাকবে, অর্ডার প্রসেস করার আরেকটা অংশ থাকবে, এবং এরা কীভাবে ডেটা আদান-প্রদান করবে তা নির্ধারণ করা হবে।

2. Extra-Functional Properties

This is about **how well the system performs** its job, beyond just doing the basic tasks. These properties ensure the software is practical and user-friendly. They include:

- **Performance**: How fast is the software? Does it load quickly?
- **Capacity**: Can it handle many users at once?
- **Reliability**: Does it crash often, or is it stable?
- **Security**: Is user data safe from hackers?
- **Adaptability**: Can the software be updated or changed easily?

Example: For the same food delivery app:

- **Performance**: The app should load the menu in seconds, not minutes.
- **Capacity**: It should work even if 10,000 people are ordering at the same time.
- **Security**: Customer payment details must be encrypted to prevent theft.

- **Adaptability**: If the app needs to add a new feature (like live order tracking), it should be easy to include without rewriting everything.

In Bangla:

অতিরিক্ত-কার্যকরী বৈশিষ্ট্য মানে সফটওয়্যারটা কতটা ভালোভাবে কাজ করে। এটা শুধু কাজ করলেই হবে না, দ্রুত কাজ করতে হবে, অনেক ইউজারকে সামলাতে হবে, নিরাপদ হতে হবে, এবং সহজে পরিবর্তন করা যেতে হবে। যেমন, ফুড ডেলিভারি অ্যাপে মেনু তাড়াতাড়ি লোড হওয়া উচিত, অনেক মানুষ একসাথে অর্ডার করলেও কাজ করা উচিত, এবং পেমেন্টের তথ্য নিরাপদ থাকা উচিত।

3. Families of Related Systems

This is about **reusing proven designs** to save time and effort. Many software systems are similar in some ways. For example, most e-commerce apps (like Amazon or Daraz) have features like product listings, shopping carts, and payment systems. Instead of designing everything from scratch, architects use **repeatable patterns** or templates that have worked well in similar systems. These are like standard building blocks that can be customized.

Example: In the food delivery app, the “shopping cart” feature can use a design pattern already used in e-commerce apps. This pattern defines how items are added, removed, or updated in the cart. By reusing this, developers don’t need to reinvent the wheel.

In Bangla:

সম্পর্কিত সিস্টেমের পরিবার বলতে বোঝায় একই ধরনের সফটওয়্যারের জন্য পুনরাবৃত্তিমূলক ডিজাইন ব্যবহার করা। যেমন, ফুড ডেলিভারি অ্যাপের “শপিং কার্ট” ফিচারটা ই-কমার্স অ্যাপের মতো একই ধরনের ডিজাইন ব্যবহার করতে পারে। এতে নতুন করে সবকিছু ডিজাইন করতে হয় না, সময় এবং পরিশ্রম বাঁচে।

Why Does This Matter?

Software architecture is important because it:

- **Guides development**: It’s like a map that developers follow to build the software correctly.
- **Ensures quality**: A good architecture makes the software fast, secure, and easy to maintain.
- **Saves time**: By reusing patterns, developers can work faster and avoid mistakes.

In Bangla:

সফটওয়্যার আর্কিটেকচার গুরুত্বপূর্ণ কারণ এটা ডেভেলপারদের সঠিক পথ দেখায়, সফটওয়্যারকে দ্রুত, নিরাপদ এবং সহজে রক্ষণাবেক্ষণযোগ্য করে, এবং পুনরাবৃত্তিমূলক প্যাটার্ন ব্যবহার করে সময় বাঁচায়।

Design Pattern

- A **repeated form of design** in which the same shape is repeated several times to form a pattern.
- A pattern is a named nugget of insight which conveys the essence of a **proven solution to a recurring problem** within a **certain context** amidst competing concerns.
- A design pattern is a **named solution to a common problem**, tailored to a specific situation, considering trade-offs (like speed vs. complexity).

1. **Pattern name:** This is a **short, catchy** name that sums up **what the pattern does. Describes the essence of the pattern in a short but expressive name**
2. **Intent:** describes the pattern and what it does
3. **Also-Known-As:** lists any synonyms for the pattern . Lists other names the pattern might have. Sometimes, different communities call the same pattern by different names.
4. **Motivation:** provides an example of the problem. This gives a real-world example of the problem the pattern solves. It’s like showing why the recipe is needed.
5. **Applicability:** Describes when to use the pattern. It tells you the situations where the pattern fits.
6. **Structure:** describes the classes that are required to implement the pattern.
7. **Participants:** describes the responsibilities of the classes that are required to implement the pattern. Lists the roles of the classes or objects in the pattern.

8. **Collaborations:** describes how the participants collaborate to carry out their responsibilities. Describes how the participants work together.
9. **Consequences:** describes the “design forces” that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented . Explains the pros and cons of using the pattern. Every solution has trade-offs.
10. **Related Patterns:** cross-references related design patterns. Lists other patterns that are similar or often used together.

Understand in a better way

A **design pattern** is like a reusable recipe for solving a common problem in software design. Imagine you’re cooking a dish, like biryani. Every time you make biryani, you follow a similar set of steps: cook rice, marinate meat, layer them, and add spices. You don’t reinvent the process each time—you use a proven method that works. Similarly, in software, design patterns are proven solutions to problems that keep popping up in different projects. They save time and ensure the solution is reliable.

The slide defines a design pattern as:

> “A repeated form of design in which the same shape is repeated several times to form a pattern.”

This means a pattern is a standard way of organizing code to solve a specific problem, used again and again in similar situations. For example, if many apps need a way to undo actions (like undoing a text edit), a design pattern provides a ready-made solution.

Brad Appleton’s definition adds:

> “A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns.”

In simpler terms, a design pattern is a named solution to a common problem, tailored to a specific situation, considering trade-offs (like speed vs.

complexity). It’s like knowing the “trick” to solve a puzzle that others have solved before.

In Bangla:

ডিজাইন প্যাটার্ন হলো একটা সফটওয়্যার সমস্যার জন্য পুনরাবৃত্তিমূলক সমাধান, যেটা বারবার ব্যবহার করা যায়। এটা একটা রেসিপির মতো, যেমন বিরিয়ানি বানানোর নিয়ম। প্রতিবার বিরিয়ানি বানাতে একই ধাপ অনুসরণ করা হয়। ডিজাইন প্যাটার্নও তেমনি—একই ধরনের সমস্যার জন্য একটা প্রমাণিত সমাধান দেয়।

Why Are Design Patterns Important?

Design patterns are like shortcuts for developers. They:

1. **Save time**: Instead of solving a problem from scratch, you use a tested solution.
2. **Improve quality**: Patterns are based on what has worked well in the past, so they lead to reliable, maintainable code.
3. **Make communication easier**: Since patterns have standard names, developers can quickly understand each other’s designs[47] design (like saying “Factory Pattern” instead of explaining the whole concept).

In Bangla:

ডিজাইন প্যাটার্ন সময় বাঁচায়, কোডের গুণমান ভালো করে, এবং ডেভেলপারদের মধ্যে যোগাযোগ সহজ করে। এটা একটা সাধারণ নাম দিয়ে জটিল ধারণা বোঝায়।

Components of a Design Pattern

The slide lists several components that describe a design pattern in detail. Let’s go through each one with explanations and examples to make them easy to understand.

1. Pattern Name

This is a short, catchy name that sums up what the pattern does. A good name makes it easy to remember and discuss.

Example: The “Singleton Pattern” ensures only one instance of a class exists (like a single printer manager in an app). The name “Singleton” instantly tells developers it’s about one unique object.

In Bangla:

প্যাটার্নের নাম হলো একটা ছোট, সহজ নাম যা প্যাটার্নের মূল ধারণা বোঝায়। যেমন, “সিঙ্গলটন প্যাটার্ন” মানে একটা ক্লাসের শুধু একটাই অবজেক্ট থাকবে।

2. Intent

This explains **what the pattern does** and its purpose. It’s like the goal of the recipe.

Example: For the Singleton Pattern, the intent is to ensure a class has only one instance and provide a global point of access to it. This is useful for things like a logging system, where you want one log file for the whole app.

In Bangla:

ইনটেন্ট বলে প্যাটার্নটা কী করে এবং এর উদ্দেশ্য কী। যেমন, সিঙ্গলটন প্যাটার্নের উদ্দেশ্য হলো একটা ক্লাসের একটিমাত্র অবজেক্ট থাকা এবং সেটা সবাই ব্যবহার করতে পারে।

3. Also-Known-As

Lists other names the pattern might have. Sometimes, different communities call the same pattern by different names.

Example: The Singleton Pattern is sometimes called the “Single Instance Pattern.” Knowing synonyms helps avoid confusion.

In Bangla:

এটা প্যাটার্নের অন্য নামগুলো বলে। যেমন, সিঙ্গলটন প্যাটার্নকে কখনো “সিঙ্গল ইনস্ট্যান্স প্যাটার্ন” বলা হয়।

4. Motivation

This gives a real-world **example of the problem** the pattern solves. It’s like showing why the recipe is needed.

Example: For the Singleton Pattern, imagine an app where multiple parts try to write to a log file. Without a Singleton, you might end up with multiple log files or conflicts. The Singleton ensures there’s one log manager controlling access.

In Bangla:

মোটিভেশন একটা বাস্তব সমস্যার উদাহরণ দেয় যেটা প্যাটার্ন সমাধান করে। যেমন, একটা অ্যাপে অনেক জায়গা থেকে লগ ফাইলে লেখার চেষ্টা করলে সমস্যা হতে পারে। সিঙ্গলটন নিশ্চিত করে একটাই লগ ম্যানেজার থাকবে।

5. Applicability

Describes **when to use the pattern**. It tells you the situations where the pattern fits.

Example: Use the Singleton Pattern when you need exactly one instance of a class (e.g., for a configuration manager) and when that instance needs to be accessible globally.

In Bangla:

অ্যাপ্লিকেবিলিটি বলে কখন প্যাটার্ন ব্যবহার করতে হবে। যেমন, সিঙ্গলটন ব্যবহার করা হয় যখন একটা ক্লাসের একটিমাত্র অবজেক্ট দরকার এবং সেটা সবাই ব্যবহার করবে।

6. Structure

Explains the **technical design** of the pattern, like the classes or components involved. It's the blueprint of the recipe.

Example: In the Singleton Pattern, the structure includes a class with a private constructor (to prevent multiple instances) and a static method that returns the single instance.

In Bangla:

স্ট্রাকচার প্যাটার্নের টেকনিক্যাল ডিজাইন বোঝায়, যেমন কোন ক্লাস বা কম্পোনেন্ট লাগবে। সিঙ্গলটনে একটা ক্লাস থাকে যার কনস্ট্রাক্টর প্রাইভেট এবং একটা স্ট্যাটিক মেথড একমাত্র অবজেক্ট ফেরত দেয়।

7. Participants

Lists the **roles** of the classes or objects in the pattern. It's like saying who does what in the recipe.

Example: In the Singleton Pattern, the participant is the Singleton class itself, responsible for creating and managing its single instance.

In Bangla:

পার্টিসিপ্যান্ট বলে ক্লাস বা অবজেক্টের ভূমিকা কী। সিঙ্গলটনে শুধু সিঙ্গলটন ক্লাসই পার্টিসিপ্যান্ট, যেটা একমাত্র অবজেক্ট তৈরি ও ম্যানেজ করে।

8. Collaborations

Describes **how the participants work together**. It's like explaining how the chef and assistant coordinate in the kitchen.

Example: In the Singleton Pattern, the Singleton class ensures that any part of the program requesting the instance gets the same object, coordinating access

internally.

In Bangla:

কোলাবোরেশন বলে পার্টিসিপ্যান্টরা কীভাবে একসাথে কাজ করে। সিঙ্গলটনে, ক্লাস নিশ্চিত করে যে সবাই একই অবজেক্ট পায়।

9. Consequences

Explains the **pros and cons** of using the pattern. Every solution has trade-offs.

Example: For the Singleton Pattern:

- **Pros**: Ensures one instance, reduces resource waste.
- **Cons**: Can make testing harder (since it's global) and might lead to hidden dependencies.

In Bangla:

কনসিকুয়েন্স বলে প্যাটার্নের ভালো-মন্দ দিক। সিঙ্গলটন একটা অবজেক্ট নিশ্চিত করে, কিন্তু টেস্টিং কঠিন হতে পারে।

10. Related Patterns

Lists other patterns that are similar or often used together. It's like suggesting side dishes for your main recipe.

Example: The Singleton Pattern is related to the Factory Pattern, which helps create objects in a controlled way.

In Bangla:

রিলেটেড প্যাটার্ন বলে অন্য প্যাটার্ন যেগুলো একই রকম বা একসাথে ব্যবহৃত হয়। সিঙ্গলটন ফ্যাক্টরি প্যাটার্নের সাথে সম্পর্কিত।

A Real-World Analogy

Imagine you're building a library system. A common problem is managing book reservations. The **Observer Pattern** (another design pattern) is often used here:

- **Problem**: When a book becomes available, all users waiting for it should be notified.
- **Pattern Solution**: The Observer Pattern lets users "subscribe" to the book's status. When the book is available, the system notifies all subscribers automatically.
- **Components**:

- **Pattern Name**: Observer
- **Intent**: Notify multiple objects when one object's state changes.
- **Motivation**: Users waiting for a book need updates without constantly checking.
- **Applicability**: Use when one object's change affects many others (e.g., notifications).
- **Consequences**: Flexible but can be complex if there are too many observers.

In Bangla:

একটা লাইব্রেরি সিস্টেমে বই রিজার্ভ করার সমস্যা হয়। অবজার্ভার প্যাটার্ন ব্যবহার করা হয়:

- সমস্যা: বই পাওয়া গেলে সব ইউজারকে জানাতে হবে।
- সমাধান: ইউজাররা বইয়ের স্ট্যাটাসে "সাবস্ক্রাইব" করে। বই পাওয়া গেলে সবাইকে জানানো হয়।
- নাম: অবজার্ভার
- উদ্দেশ্য: একটা অবজেক্টের পরিবর্তন সবাইকে জানানো।
- ভালো-মন্দ: সুবিধাজনক, কিন্তু অনেক ইউজার হলে জটিল হতে পারে।

Why Patterns Matter

Patterns help you:

- **Solve problems faster**: Use solutions that experts have already figured out.
- **Write better code**: Patterns lead to code that's easier to maintain and extend.
- **Learn from others**: Patterns are based on years of experience from other developers.

In Bangla:

প্যাটার্ন সমস্যা দ্রুত সমাধান করে, ভালো কোড লিখতে সাহায্য করে, এবং অন্যের অভিজ্ঞতা থেকে শেখায়।

Separation of Concerns

- Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently
- A *concern* is a feature or behavior that is specified as part of the requirements model for the software

Understand more

Separation of Concerns is a design principle that says: to solve a big, complicated problem, break it into smaller, independent pieces. Each piece (or "concern") focuses on one specific task or feature, making it easier to understand, build, and manage. This is like dividing a big project into smaller tasks so you don’t feel overwhelmed.

The slide explains:

> “Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently.”

In simpler terms, instead of tackling a huge problem all at once, split it into smaller parts that you can work on separately. This reduces confusion and saves time.

Example: Imagine you’re planning a wedding. The whole event is a complex problem with many tasks: arranging food, decorating the venue, managing guests, and organizing music. If one person tries to do everything, it’s chaotic. Instead, you assign:

- One team for food.
- One team for decorations.
- One team for guest management.
- One team for music.

Each team focuses on their task without worrying about the others. This is separation of concerns—each task is a “concern” handled independently.

In Bangla:

কনসার্নের বিভাজন মানে একটা জটিল সমস্যাকে ছোট ছোট অংশে ভাগ করা, যাতে প্রতিটা অংশ আলাদাভাবে সমাধান করা যায়। এটা একটা বিয়ের আয়োজনের মতো—খাবার, সাজসজ্জা, অতিথি ব্যবস্থাপনা আলাদা আলাদা দলের হাতে দেওয়া হয়, যাতে কাজ সহজ হয়।

What is a “Concern”?

A **concern** is a specific feature, behavior, or requirement of the software. It’s one piece of what the software needs to do. For example, in a food delivery app, concerns might include:

- Displaying the menu.
- Processing payments.
- Tracking delivery status.

Each concern is a distinct task that can be designed and coded separately.

In Bangla:

কনসার্ন মানে সফটওয়্যারের একটা নির্দিষ্ট ফিচার বা কাজ। যেমন, একটা ফুড ডেলিভারি অ্যাপে মেনু দেখানো, পেমেন্ট প্রসেস করা, বা ডেলিভারি ট্র্যাক করা—এগুলো আলাদা কনসার্ন।

Why is Separation of Concerns Important?

1. **Simplifies Complex Problems**: Breaking a big problem into smaller pieces makes it less overwhelming.
2. **Easier to Manage**: Each piece can be worked on by different developers or teams without conflicts.
3. **Saves Time**: Smaller tasks are quicker to solve and test.
4. **Improves Maintenance**: If something breaks, you only need to fix the specific piece, not the whole system.

Example: In the wedding analogy, if the food isn’t good, you only need to talk to the food team, not redo the entire wedding plan. Similarly, in a food delivery app, if the payment system fails, you fix just that part without touching the menu or delivery tracking.

In Bangla:

কনসার্নের বিভাজন জটিল সমস্যাকে সহজ করে, কাজ ভাগ করে দেওয়ায় সময় বাঁচে, এবং কিছু ভুল হলে শুধু সেই অংশ ঠিক করা যায়। যেমন, বিয়েতে খাবার খারাপ হলে শুধু খাবারের দলের সাথে কথা বলতে হবে, পুরো আয়োজন নয়।

How is Separation of Concerns Applied?

The slide mentions that separation of concerns is related to other design concepts: **modularity**, **aspects**, **functional independence**, and **refinement**.

Let’s explore these and the overall idea with examples.

1. Breaking Down the Problem

The core idea is to divide the software into independent pieces. Each piece handles one concern (task or feature). These pieces should ideally not overlap too much, so changes in one don’t affect others.

Example: For a food delivery app:

- **Concern 1**: Show the menu (list of restaurants and items).
- **Concern 2**: Handle user orders (add items to cart, calculate total).
- **Concern 3**: Process payments (credit card, mobile banking).
- **Concern 4**: Track delivery (show driver’s location).

Each concern is coded separately. If you want to add a new payment method, you only modify the payment concern, not the menu or delivery tracking.

In Bangla:

সফটওয়্যারকে ছোট ছোট অংশে ভাগ করা হয়। যেমন, ফুড ডেলিভারি অ্যাপে মেনু দেখানো, অর্ডার নেওয়া, পেমেন্ট করা, এবং ডেলিভারি ট্র্যাক করা আলাদা অংশ। একটা নতুন পেমেন্ট মেথড যোগ করতে শুধু পেমেন্ট অংশে কাজ করা হবে।

2. Related Design Concepts

Separation of concerns is the foundation for other important software design ideas. Here’s how it connects:

- **Modularity**: This means dividing the software into modules (like building blocks). Each module handles one concern. For example, in the food delivery app, the payment system is one module, and the menu system is another. Modules make the code organized and reusable.
  In Bangla:
  মডিউলারিটি মানে সফটওয়্যারকে আলাদা মডিউলে ভাগ করা। যেমন, পেমেন্ট সিস্টেম একটা মডিউল, মেনু সিস্টেম আরেকটা।
- **Aspects**: Some concerns “cross-cut” multiple parts of the software. For example, logging every action (like who ordered what) is a concern that applies across the app. Aspects help manage these cross-cutting concerns separately.
  In Bangla:

অ্যাসপেক্ট মানে এমন কনসার্ন যেগুলো সফটওয়্যারের অনেক অংশে ছড়িয়ে থাকে, যেমন লগিং (কারা কী অর্ডার করল তা রেকর্ড করা)।

- **Functional Independence**: Each module should do one job well and not depend too much on others. This is called low coupling (less dependency) and high cohesion (focused task). For example, the payment module shouldn't need to know how the menu is displayed.
  In Bangla:
  ফাংশনাল ইন্ডিপেন্ডেন্স মানে প্রতিটা মডিউল একটা কাজ ভালোভাবে করবে এবং অন্যের উপর কম নির্ভর করবে। যেমন, পেমেন্ট মডিউলের মেনু ডিসপ্লের সাথে সম্পর্ক থাকবে না।
- **Refinement**: This is about breaking down tasks into smaller, detailed steps. For example, the payment concern might be refined into steps like "validate card," "charge amount," and "send receipt."
  In Bangla:
  রিফাইনমেন্ট মানে কাজকে আরও ছোট ধাপে ভাগ করা। যেমন, পেমেন্ট কনসার্নকে "কার্ড চেক করা," "টাকা কাটা," "রিসিপ্ট পাঠানো" এ ভাগ করা।

A Real-World Analogy

Think of separation of concerns like organizing your schoolwork. If you have a big project due, you don't try to do everything at once. Instead:

- You write the introduction one day.
- You research the topic another day.
- You create charts or visuals separately.
- You proofread at the end.

Each task is a "concern" that you handle independently. If your teacher asks you to improve the charts, you only work on that part, not the whole project. In software, separation of concerns works the same way to keep things manageable.

In Bangla:

এটা স্কুলের প্রজেক্টের মতো। একটা বড় প্রজেক্টকে ভাগ করা হয়: ভূমিকা লেখা, রিসার্চ করা, চার্ট বানানো, প্রুফরিডিং। প্রতিটা কাজ আলাদা। সফটওয়্যারেও এভাবে কনসার্ন ভাগ করা হয়।

How It Works in Software Development

Let's apply separation of concerns to a **library management system**:

- **Concern 1**: User authentication (logging in users).
- **Concern 2**: Book catalog (searching and displaying books).
- **Concern 3**: Borrowing system (checking out books).
- **Concern 4**: Fine calculation (penalties for late returns).

Each concern is coded as a separate module:

- If you need to change how fines are calculated (e.g., increase the penalty), you only modify the fine calculation module.
- The authentication or book catalog modules remain untouched, reducing the risk of breaking other parts.

This separation makes the system easier to:

- **Develop**: Different developers can work on different modules simultaneously.
- **Test**: You can test the borrowing system without worrying about authentication.
- **Maintain**: Fixing bugs in one module doesn't affect others.

In Bangla:

একটা লাইব্রেরি সিস্টেমে:

- কনসার্ন ১: ইউজার লগইন।
- কনসার্ন ২: বইয়ের ক্যাটালগ।
- কনসার্ন ৩: বই ধার দেওয়া।
- কনসার্ন ৪: জরিমানা হিসাব।
  প্রতিটা কনসার্ন আলাদা মডিউল। জরিমানা বাড়াতে হলে শুধু জরিমানা মডিউল পরিবর্তন করা হবে, অন্য মডিউল অপরিবর্তিত থাকবে।

Benefits of Separation of Concerns

- **Less Complexity**: Smaller pieces are easier to understand than one giant system.
- **Teamwork**: Multiple developers can work on different concerns at the same time.

- **Reusability**: A module (like payment processing) can be reused in other projects.
- **Easier Updates**: Changing one concern doesn't break the whole system.

In Bangla:

- জটিলতা কমে।
- টিমে কাজ সহজ হয়।
- মডিউল পুনরায় ব্যবহার করা যায়।
- আপডেট করা সহজ।

Putting It All Together

Separation of Concerns is like organizing a big task into smaller, manageable chunks. In software, it means dividing the system into independent features or tasks (concerns) so each can be designed, coded, and fixed separately. This principle connects to modularity (building blocks), aspects (cross-cutting tasks), functional independence (focused modules), and refinement (detailed steps), all of which make software development faster and more reliable.

For example, in a messaging app like WhatsApp:

- **Concern 1**: Sending messages.
- **Concern 2**: Encrypting messages for security.
- **Concern 3**: Showing notifications.
  Each is handled separately, so adding a new notification style doesn't affect how messages are sent or encrypted.

In Bangla:

একটা মেসেজিং অ্যাপে:

- মেসেজ পাঠানো।
- মেসেজ এনক্রিপ্ট করা।
- নোটিফিকেশন দেখানো।
  প্রতিটা আলাদা কনসার্ন, তাই নতুন নোটিফিকেশন যোগ করলেও মেসেজ পাঠানোর কোড অপরিবর্তিত থাকে।

1. Modularity

What is Modularity?

Modularity means breaking a big software system into smaller, independent parts called **modules**. Each module handles one specific task or feature, making the system easier to understand, build, and maintain. It's like dividing a big book into chapters—each chapter covers one topic, so it's easier to read and edit.

The slide says:

> **"Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project."**

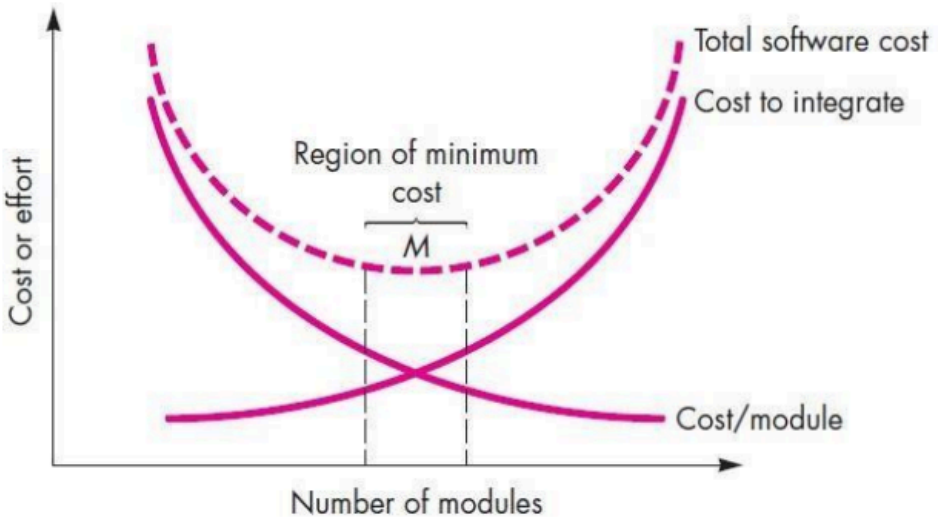
Example: Imagine building a food delivery app like Foodpanda. Instead of writing one giant program, you split it into modules:

- **Module 1**: Display the menu (restaurants and food items).
- **Module 2**: Handle orders (add items to cart, calculate total).
- **Module 3**: Process payments (credit card, mobile banking).
- **Module 4**: Track delivery (show driver's location).

Each module is developed separately and then connected to make the full app. If you need to change the payment system (e.g., add PayPal), you only modify the payment module, not the whole app.

In Bangla:

মডিউলারিটি মানে একটা বড় সফটওয়্যারকে ছোট ছোট অংশে (মডিউল) ভাগ করা। যেমন, একটা ফুড ডেলিভারি অ্যাপে মেনু দেখানো, অর্ডার নেওয়া, পেমেন্ট করা—এগুলো আলাদা মডিউল। এতে কাজ সহজ হয়, এবং একটা অংশ পরিবর্তন করলেও বাকি অংশ অপরিবর্তিত থাকে।



Why is Modularity Important?

The slide quotes:

> **"Modularity is the single attribute of software that allows a program to be intellectually manageable."**

This means modularity makes software easier to understand. A **monolithic** system (one giant program with no modules) is like a 1,000-page book with no chapters—it's overwhelming. Modularity reduces complexity by:

- **Simplifying Development**: Developers can work on different modules at the same time.
- **Easing Changes**: Update one module without affecting others.
- **Improving Testing**: Test each module separately.
- **Reducing Costs**: Less time and effort to build and maintain.

Example: In the food delivery app, if the delivery tracking module has a bug (e.g., showing the wrong location), you only debug that module, not the entire app.

In Bangla:

মডিউলারিটি সফটওয়্যারকে বোঝা সহজ করে। একটা মনোলিথিক (একক বড় প্রোগ্রাম) সিস্টেম বোঝা কঠিন। মডিউলারিটি কাজ ভাগ করে দেয়, পরিবর্তন সহজ করে, এবং খরচ কমায়। যেমন, ডেলিভারি ট্র্যাকিং-এ সমস্যা হলে শুধু সেই মডিউল ঠিক করা হবে।

2. Information Hiding

What is Information Hiding?

Information Hiding means designing modules so that their internal details (like data structures or algorithms) are hidden from other modules. Modules only share the minimum information needed to work together, through well-defined **interfaces**.

The slide explains:

> **"Modules should be specified and designed in such a way that the data structures and algorithm details of one module are not accessible to other modules."**

Example: In the food delivery app, the payment module might use a complex algorithm to process credit card payments. Other modules (like the menu or delivery tracking) don't need to know *how* the payment is processed—they just need to know *if* it was successful. The payment module hides its internal logic and shares only the result (e.g., "Payment successful") via an interface.

In Bangla:

ইনফরমেশন হাইডিং মানে মডিউলের ভেতরের বিস্তারিত (যেমন ডেটা বা অ্যালগরিদম) অন্য মডিউল থেকে লুকিয়ে রাখা। যেমন, পেমেন্ট মডিউল কীভাবে কার্ড প্রসেস করে তা মেনু মডিউল জানবে না—শুধু জানবে পেমেন্ট হয়েছে কি না।

Why is Information Hiding Important?

The slide lists benefits:

- **Reduces Side Effects**: Changes in one module don't accidentally break others.
- **Limits Global Impact**: Local changes stay local.
- **Controlled Interfaces**: Modules communicate only through specific channels.
- **Avoids Global Data**: Prevents messy dependencies.
- **Leads to Encapsulation**: A key principle of good design where data and behavior are bundled together.

- **Improves Quality**: Makes software more reliable and maintainable.

Example: If the payment module's algorithm changes (e.g., from one encryption method to another), other modules aren't affected because they never saw the algorithm—they only used the interface. This prevents errors from spreading.

In Bangla:

ইনফরমেশন হাইডিং ভুল কমায়, পরিবর্তনের প্রভাব সীমিত রাখে, এবং সফটওয়্যারের গুণমান বাড়ায়। যেমন, পেমেন্ট মডিউলের অ্যালগরিদম বদলালেও মেনু মডিউলের কিছু হবে না, কারণ তারা শুধু ইন্টারফেস ব্যবহার করে।

---

3. Functional Independence

What is Functional Independence?

Functional Independence means designing modules so each one focuses on a single task (high cohesion) and has minimal dependency on other modules (low coupling). It builds on modularity, information hiding, and separation of concerns.

The slide says:

> **"Functional independence is achieved by developing modules with 'single-minded' function and an 'aversion' to excessive interaction with other modules."**

Key Criteria:

- **Cohesion**: How well a module's internal parts work together to do one task. High cohesion means the module does one thing well.
- **Coupling**: How much a module depends on others. Low coupling means modules are independent.

Example: In the food delivery app:

- **High Cohesion**: The payment module only handles payments (validating cards, charging amounts). It doesn't display menus or track deliveries.
- **Low Coupling**: The payment module doesn't need to know how the delivery tracking works—it just passes the payment status (e.g., "Paid") when needed.

In Bangla:

ফাংশনাল ইন্ডিপেন্ডেন্স মানে মডিউল একটা কাজে ফোকাস করবে (হাই কোহেসন) এবং অন্য মডিউলের উপর কম নির্ভর করবে (লো কাপলিং)। যেমন, পেমেন্ট মডিউল শুধু পেমেন্ট করে, ডেলিভারি ট্র্যাকিং-এর সাথে তার সম্পর্ক কম।

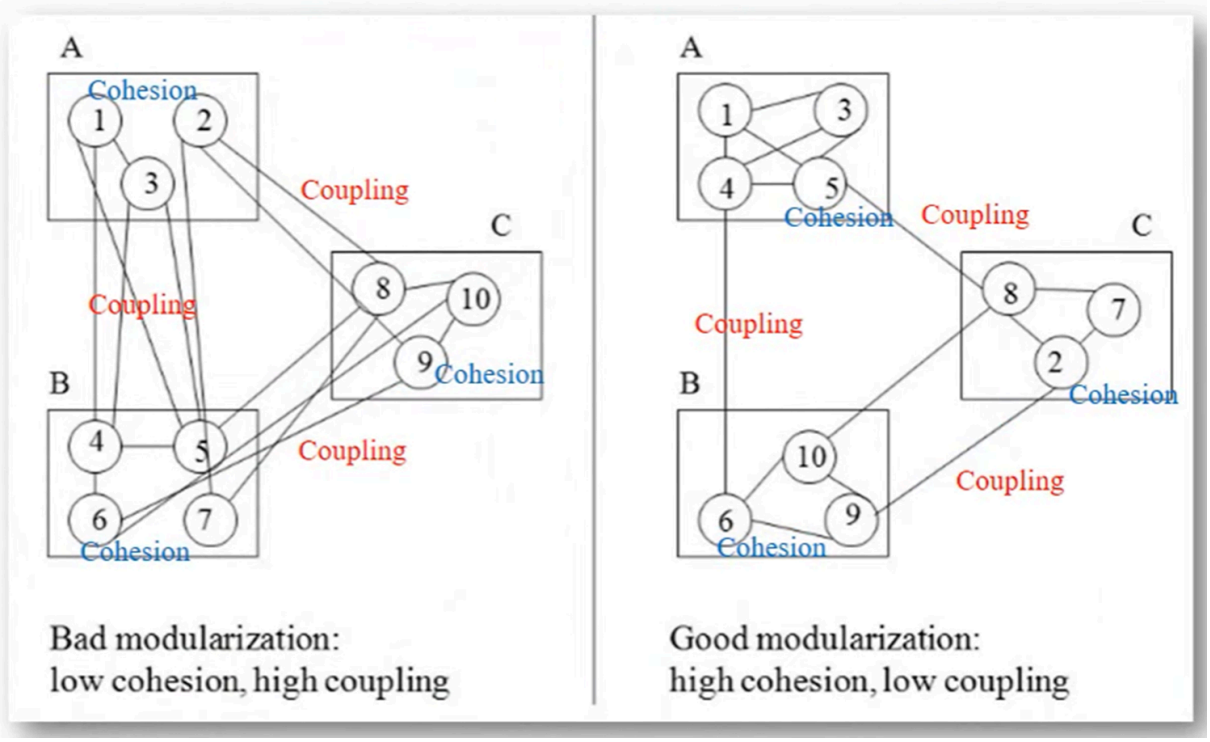
Why is Functional Independence Important?

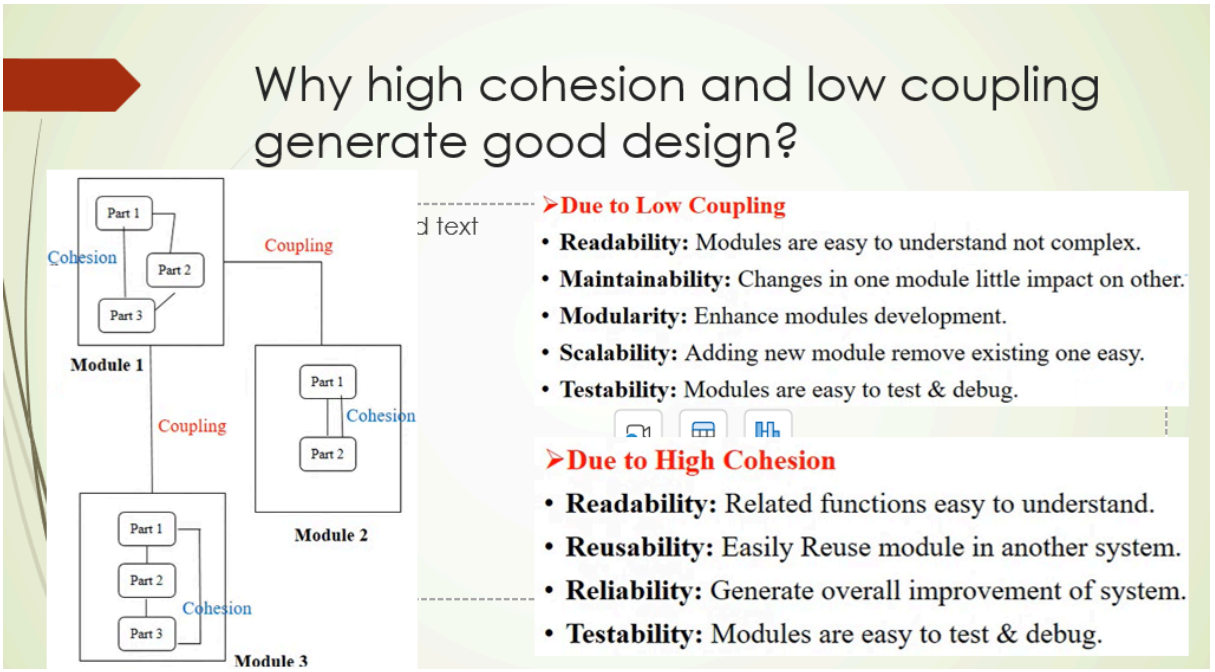
- **High Cohesion**: Makes modules easier to understand and reuse because they focus on one task.
- **Low Coupling**: Reduces the risk of changes in one module breaking others, making maintenance easier.

Example: If the delivery tracking module needs an update (e.g., to show real-time GPS), it won't affect the payment module because they're independent. This saves time and reduces errors.

In Bangla:

হাই কোহেসন মডিউলকে একটা কাজে পারফেক্ট করে, লো কাপলিং পরিবর্তনের প্রভাব কমায়। যেমন, ডেলিভারি ট্র্যাকিং আপডেট করলেও পেমেন্ট মডিউল অপরিবর্তিত থাকবে।





4. Refinement

What is Refinement?

Refinement is a **top-down** design process where you start with a high-level idea and break it into smaller, detailed steps until you reach actual code. It's like planning a trip: first, you decide the destination, then the route, then specific stops, and finally the exact roads to take.

The slide explains:

> **"A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statements are reached."**

Example: Suppose you're designing the payment module for the food delivery app. Refinement works like this:

1. **High-Level**: "Process a payment."
2. **Break Down**:
    - Validate the credit card.
    - Charge the amount.
    - Send a receipt.

3. **Further Details**:
    - Validate card: Check card number, expiry date, and CVV.
    - Charge amount: Connect to bank API, deduct funds.
    - Send receipt: Email the user a confirmation.
4. **Code Level**: Write the actual programming statements for each step.

In Bangla:

রিফাইনমেন্ট মানে একটা বড় ধারণাকে ছোট ছোট ধাপে ভাগ করা, যতক্ষণ না কোড লেখার পর্যায়ে পৌঁছানো যায়। যেমন, "পেমেন্ট প্রসেস করা" ভাগ করা হয়: কার্ড চেক করা, টাকা কাটা, রিসিপ্ট পাঠানো—তারপর কোড লেখা হয়।

Why is Refinement Important?

- **Clarity**: Breaks complex tasks into manageable steps.
- **Organization**: Creates a clear plan before coding starts.
- **Ease of Implementation**: Developers know exactly what to code at each step.

In Bangla:

রিফাইনমেন্ট জটিল কাজকে সহজ করে, পরিকল্পনা স্পষ্ট করে, এবং কোডিং সহজ করে।

---

5. Aspects

What are Aspects?

Aspects deal with **cross-cutting concerns**—features or behaviors that affect multiple parts of the software. A cross-cutting concern is something that "cuts across" different modules, like a rule that applies everywhere.

The slide defines:

> **"An aspect is a representation of a cross-cutting concern."**

Example: In a home security app (like SafeHomeAssured.com), consider two requirements:

- **Requirement A**: Let users view camera footage over the internet.
- **Requirement B**: Validate users (e.g., check login credentials) before they access any feature.

Requirement B (user validation) is a cross-cutting concern because it applies to *all* features, including viewing camera footage. You can't satisfy Requirement A without first satisfying Requirement B (validating the user). The validation logic is an **aspect** because it affects multiple modules.

In Bangla:

অ্যাসপেক্ট মানে এমন কনসার্ন যেগুলো সফটওয়্যারের অনেক অংশে ছড়িয়ে থাকে। যেমন, একটা হোম সিকিউরিটি অ্যাপে ইউজার ভ্যালিডেশন (লগইন চেক) সব ফিচারের জন্য লাগে—এটা একটা অ্যাসপেক্ট।

Why are Aspects Important?

- **Manage Cross-Cutting Concerns**: Aspects keep features like logging, security, or validation separate from the main logic, making the code cleaner.
- **Reusability**: The same aspect (e.g., validation logic) can be applied across modules.

Example: In the security app, the validation aspect ensures every feature (camera access, alarm settings) checks the user's login first. If you update the validation process (e.g., add two-factor authentication), you modify the aspect, not every feature.

In Bangla:

অ্যাসপেক্ট ক্রস-কাটিং কনসার্ন (যেমন সিকিউরিটি) আলাদা রাখে, কোড পরিষ্কার করে, এবং পুনরায় ব্যবহার করা যায়। যেমন, ভ্যালিডেশন পরিবর্তন করতে শুধু অ্যাসপেক্ট বদলানো হবে।

How These Concepts Connect to Separation of Concerns

All these concepts (**Modularity**, **Information Hiding**, **Functional Independence**, **Refinement**, and **Aspects**) are ways to apply **Separation of Concerns**, the idea of dividing a complex problem into manageable pieces:

- **Modularity** creates the pieces (modules).
- **Information Hiding** ensures each piece keeps its details private.
- **Functional Independence** makes each piece focused (high cohesion) and independent (low coupling).
- **Refinement** breaks each piece into detailed steps.
- **Aspects** handle concerns that apply across multiple pieces.

Together, they make software easier to design, build, test, and maintain.

In Bangla:

এই সব ধারণা (**মডিউলারিটি**, **ইনফরমেশন হাইডিং**, **ফাংশনাল ইন্ডিপেন্ডেন্স**, **রিফাইনমেন্ট**, **অ্যাসপেক্ট**) কনসার্নের বিভাজনের অংশ। মডিউলারিটি অংশ তৈরি করে, ইনফরমেশন হাইডিং গোপন রাখে, ফাংশনাল ইন্ডিপেন্ডেন্স ফোকাস এবং স্বাধীনতা দেয়, রিফাইনমেন্ট ধাপে ভাগ করে, এবং অ্যাসপেক্ট সবার জন্য কনসার্ন ম্যানেজ করে।

A Real-World Analogy

Imagine organizing a school event:

- **Modularity**: Divide tasks into teams (food, decorations, music).
- **Information Hiding**: The food team doesn't share their recipe details with the music team—they just deliver the food.
- **Functional Independence**: The music team focuses only on music (high cohesion) and doesn't need to coordinate much with decorations (low coupling).
- **Refinement**: The decoration team plans: "Decorate the hall" → "Choose colors" → "Hang lights" → specific tasks.
- **Aspects**: A rule like "Ensure everyone has an entry ticket" applies to all teams (a cross-cutting concern).

In Bangla:

একটা স্কুল ইভেন্টের মতো:

- মডিউলারিটি: খাবার, সাজসজ্জা, মিউজিক দল।
- ইনফরমেশন হাইডিং: খাবার দল রেসিপি শেয়ার করে না।
- ফাংশনাল ইন্ডিপেন্ডেন্স: মিউজিক দল শুধু মিউজিক করে।
- রিফাইনমেন্ট: সাজসজ্জা দল ধাপে ধাপে প্ল্যান করে।
- অ্যাসপেক্ট: "সবাই টিকিট দেখাবে" সব দলের জন্য।

Why These Concepts Matter

- **Easier Development**: Modularity and refinement make planning and coding simpler.

- **Fewer Errors**: Information hiding and low coupling prevent mistakes from spreading.
- **Better Maintenance**: Independent modules and aspects make updates and fixes easier.
- **Scalability**: Add new features by creating new modules or aspects without disrupting the system.

In Bangla:

এই ধারণাগুলো কাজ সহজ করে, ভুল কমায়, রক্ষণাবেক্ষণ সহজ করে, এবং নতুন ফিচার যোগ করা সহজ করে।

1. Refactoring

What is Refactoring?

Refactoring is the process of improving the internal structure of a software system's code or design without changing what it does externally. It's like reorganizing a messy kitchen to make it more efficient—same food gets cooked, but the process is smoother and cleaner.

The slide quotes Fowler:

> "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure."

Example: Imagine a food delivery app where the payment processing code is messy—full of repeated logic, unclear variable names, and inefficient loops. Refactoring involves:

- Removing duplicate code.
- Renaming variables to be clearer (e.g., amt to totalAmount).
- Simplifying complex algorithms.
  The app still processes payments the same way from the user's perspective, but the code is now easier to understand and maintain.

In Bangla:

রিফ্যাক্টরিং মানে কোড বা ডিজাইনের ভেতরের গঠন উন্নত করা, কিন্তু বাইরের কাজ অপরিবর্তিত রাখা। যেমন, ফুড ডেলিভারি অ্যাপের পেমেন্ট কোড জটিল হলে, কোড পরিষ্কার করা, নাম স্পষ্ট

করা, বা অপ্রয়োজনীয় অংশ সরানো হয়—কিন্তু ইউজারের জন্য পেমেন্ট একইভাবে কাজ করে।

Why Refactor?

Refactoring fixes design flaws like:

- **Redundancy**: Repeated code that can be consolidated.
- **Unused Elements**: Code or designs that serve no purpose.
- **Inefficient Algorithms**: Slow or complex processes that can be optimized.
- **Poor Data Structures**: Inappropriate ways of storing data.

Example: If the payment module stores user data in a slow, flat file instead of a database, refactoring might switch to a database for faster access, without changing how users make payments.

Benefits of Refactoring

- Makes code easier to read and maintain.
- Reduces bugs by simplifying logic.
- Improves performance by optimizing algorithms.
- Prepares the system for future changes.

In Bangla:

রিফ্যাক্টরিং কোড পরিষ্কার করে, ভুল কমায়, দ্রুত করে, এবং ভবিষ্যতে পরিবর্তন সহজ করে।

---

2. Object-Oriented Design Concepts

What is Object-Oriented (OO) Design?

Object-Oriented Design is a way of designing software by modeling it as a collection of **objects** that interact with each other. Each object represents a real-world thing or concept, with its own data (attributes) and behavior (methods). OO design is popular because it makes software flexible, reusable, and easier to maintain.

Key OO concepts mentioned in the slide:

- **Classes and Objects**: A **class** is a blueprint (e.g., “Car”), and an **object** is an instance of that blueprint (e.g., “My red Toyota”).

- **Inheritance**: A class can inherit properties and behaviors from another class (e.g., a “SportsCar” class inherits from “Car”).
- **Messages**: Objects communicate by sending messages (e.g., one object tells another to perform an action).
- **Polymorphism**: Different objects can respond to the same message in their own way (e.g., both “Car” and “Bike” can respond to “startEngine” differently).

Example: In a food delivery app:

- **Class**: Order (blueprint for orders).
- **Object**: A specific order (e.g., “Order #123 for pizza”).
- **Inheritance**: A SpecialOrder class (e.g., for discounted orders) inherits from Order.
- **Messages**: The Order object sends a message to the Payment object to process payment.
- **Polymorphism**: CreditCardPayment and CashPayment both respond to “processPayment” but handle it differently.

In Bangla:

অবজেক্ট-ওরিয়েন্টেড ডিজাইন মানে সফটওয়্যারকে অবজেক্টের সমষ্টি হিসেবে ডিজাইন করা। যেমন, ফুড ডেলিভারি অ্যাপে:

- ক্লাস: Order (অর্ডারের নকশা)।
- অবজেক্ট: একটা নির্দিষ্ট অর্ডার।
- ইনহেরিটেন্স: SpecialOrder ক্লাস Order থেকে বৈশিষ্ট্য পায়।
- মেসেজ: Order পেমেন্ট প্রসেস করতে বলে।
- পলিমরফিজম: ক্রেডিট কার্ড আর ক্যাশ পেমেন্ট আলাদাভাবে প্রসেস হয়।

Why OO Design?

- **Reusability**: Classes can be reused across projects.
- **Flexibility**: Inheritance and polymorphism make it easy to add new features.
- **Maintainability**: Objects encapsulate data and behavior, reducing complexity.

In Bangla:

ওও ডিজাইন পুনরায় ব্যবহার, নতুন ফিচার যোগ করা, এবং রক্ষণাবেক্ষণ সহজ করে।

---

3. Design Classes

What are Design Classes?

Design Classes are detailed blueprints of the software components that are created during the design phase. They refine the **analysis classes** (which come from the requirements phase and describe what the software needs to do) by adding technical details to make them ready for coding. Think of analysis classes as a rough sketch of a house, and design classes as the detailed architectural plans that specify materials, measurements, and wiring.

The slide focuses on three types of design classes:

1. **Entity Classes**: Represent the core data and behavior of the system.
2. **Boundary Classes**: Handle the user interface and how data is shown to users.
3. **Controller Classes**: Manage interactions and coordination between other classes.

These classes work together to turn the software requirements into a working system, ensuring that each part has a clear role (aligned with **Separation of Concerns**).

In Bangla:

ডিজাইন ক্লাস হলো সফটওয়্যারের অংশগুলোর বিস্তারিত নকশা, যা এনালিসিস ক্লাস থেকে তৈরি হয়। এগুলো কোডিংয়ের জন্য প্রস্তুত করে। তিন ধরনের ক্লাস:

- **এনটিটি ক্লাস**: মূল ডেটা এবং কাজ।
- **বাউন্ডারি ক্লাস**: ইউজার ইন্টারফেস।
- **কন্ট্রোলার ক্লাস**: কাজের সমন্বয়।

---

1. Entity Classes

What are Entity Classes?

Entity Classes are refined versions of analysis classes that represent the core objects or concepts in the system, focusing on the data and behavior related to the business or problem domain. They store information and define the actions those objects can perform.

The slide says:

> “Analysis classes are refined during design to become entity classes.”

In the requirements phase, analysis classes describe what the system needs (e.g., “we need to track orders”). In the design phase, entity classes add details like specific attributes (data) and methods (actions) to make them ready for coding.

Example: In a food delivery app:

- **Analysis Class**: “Order” (describes that the system needs to manage orders).
- **Entity Class**: Order (refined with details):
  - **Attributes**: orderId, items, totalAmount, customer.
  - **Methods**: addItem(), calculateTotal(), confirmOrder().

The Order entity class represents a real-world order and handles its data and behavior, like adding items to the cart or calculating the total cost.

Why Entity Classes?

- They model the core business logic (e.g., what an order is and what it can do).
- They ensure the system accurately represents the problem domain.

In Bangla:

এনটিটি ক্লাস হলো এনালিসিস ক্লাসের বিস্তারিত সংস্করণ, যা সিস্টেমের মূল অবজেক্ট বা ধারণা বোঝায়। যেমন, ফুড ডেলিভারি অ্যাপে Order ক্লাসে থাকবে orderId, items, আর মেথড যেমন addItem()। এটা ব্যবসার মূল লজিক ধরে রাখে।

---

2. Boundary Classes

What are Boundary Classes?

Boundary Classes are responsible for the **user interface**—the part of the software that users see and interact with, like screens, forms, or reports. They manage how data from entity classes is presented to users and how user inputs are collected.

The slide explains:

> "Boundary classes are developed during design to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used."
> "Boundary classes are designed with the responsibility of managing the way entity objects are represented to users."

Example: In the food delivery app:

- **Boundary Class**: MenuScreen
  - Displays the list of restaurants and food items (data from entity classes like Restaurant or MenuItem).
  - Includes buttons for users to select items or place an order.
  - Collects user inputs (e.g., clicking "Add to Cart") and passes them to other parts of the system.

The MenuScreen boundary class doesn't store the menu data itself—it gets that data from entity classes (like MenuItem) and formats it for display (e.g., as a list with pictures and prices).

Why Boundary Classes?

- They separate the user interface from the business logic, making it easier to change the UI without affecting core functionality.
- They ensure a good user experience by presenting data clearly.

Real-World Analogy: Think of a waiter in a restaurant. The waiter (boundary class) doesn't cook the food (entity class) but presents the menu to customers, takes their orders, and delivers the food. The waiter manages how the kitchen's work is shown to customers.

In Bangla:

বাউন্ডারি ক্লাস ইউজার ইন্টারফেস তৈরি করে, যেমন স্ক্রিন বা রিপোর্ট। ফুড অ্যাপে MenuScreen মেনু দেখায়, ইউজারের ইনপুট নেয়। এটা এনটিটি ক্লাসের ডেটা ইউজারের কাছে সুন্দরভাবে উপস্থাপন করে। যেমন, রেস্তোরাঁর ওয়েটার খাবার তৈরি করে না, শুধু মেনু দেখায় আর অর্ডার নেয়।

---

3. Controller Classes

What are Controller Classes?

Controller Classes act as the "managers" of the system, coordinating interactions between entity classes, boundary classes, and other parts. They handle complex logic, ensure data flows correctly, and validate inputs.

The slide lists their responsibilities:

> "Controller classes are designed to manage:
>
> - the creation or update of entity objects;
> - the instantiation of boundary objects as they obtain information from entity objects;
> - complex communication between sets of objects;
> - validation of data communicated between objects or between the user and the application."

Example: In the food delivery app:

- **Controller Class**: OrderController
  - **Creates/Updates Entity Objects**: Creates a new Order object when a user places an order or updates it when they add items.
  - **Instantiates Boundary Objects**: Tells the MenuScreen (boundary class) to display the updated order details.
  - **Manages Communication**: Ensures the Order object's data is sent to the Payment object for processing.
  - **Validates Data**: Checks if the user's input (e.g., delivery address) is valid before creating the order.

The OrderController is like a project manager who ensures everyone (entity and boundary classes) works together smoothly.

Why Controller Classes?

- They centralize complex logic, keeping entity and boundary classes focused on their roles.
- They ensure data consistency and proper coordination.

- They make the system easier to maintain by isolating interaction logic.

Real-World Analogy: In the restaurant analogy, the controller is like the restaurant manager. The manager ensures the waiter (boundary) delivers the customer's order to the kitchen (entity), checks that the order is correct, and coordinates between staff to keep things running smoothly.

In Bangla:

কন্ট্রোলার ক্লাস সিস্টেমের ম্যানেজারের মতো। ফুড অ্যাপে OrderController:

- নতুন Order তৈরি বা আপডেট করে।
- MenuScreen-কে ডেটা দেখাতে বলে।
- অর্ডার আর পেমেন্টের মধ্যে যোগাযোগ করে।
- ইনপুট (যেমন ডেলিভারি অ্যাড্রেস) চেক করে।
  এটা রেস্তোরাঁর ম্যানেজারের মতো, যে ওয়েটার আর কিচেনের মধ্যে সমন্বয় করে।

How These Classes Work Together

Entity, Boundary, and Controller classes follow the **Model-View-Controller (MVC)** pattern, a common way to organize software:

- **Entity Classes** = **Model**: Store data and business logic (e.g., Order).
- **Boundary Classes** = **View**: Display data to users (e.g., MenuScreen).
- **Controller Classes** = **Controller**: Manage interactions (e.g., OrderController).

Example in the Food Delivery App:

1. The user opens the app and sees the MenuScreen (boundary class), which displays restaurants.
2. The MenuScreen gets restaurant data from the Restaurant entity class.
3. When the user selects items and clicks "Place Order," the OrderController (controller class):
   - Creates a new Order (entity class).
   - Validates the order (e.g., checks if items are available).
   - Updates the MenuScreen to show the order confirmation.
4. The Order entity is saved to a database (via a persistent class, not discussed here).

This separation ensures each class has a clear role, making the system easier to build, test, and maintain.

In Bangla:

এনটিটি, বাউন্ডারি, আর কন্ট্রোলার ক্লাস MVC প্যাটার্নে কাজ করে:

- এনটিটি = মডেল: ডেটা এবং লজিক (অর্ডার)।
- বাউন্ডারি = ভিউ: ইউজারকে দেখায় (মেনু স্ক্রিন)।
- কন্ট্রোলার = কন্ট্রোলার: সমন্বয় করে (অর্ডার কন্ট্রোলার)।
  যেমন, ইউজার মেনু দেখে, অর্ডার দেয়, কন্ট্রোলার অর্ডার তৈরি করে, এবং স্ক্রিনে কনফার্মেশন দেখায়।

Why These Classes Matter

- **Entity Classes**: Ensure the system accurately models the business (e.g., orders, customers).
- **Boundary Classes**: Provide a user-friendly interface, separating presentation from logic.
- **Controller Classes**: Coordinate the system, keeping other classes focused and independent.
- **Alignment with Separation of Concerns**: Each class type handles a specific concern (data, UI, coordination), reducing complexity and improving maintainability.
- **High Cohesion and Low Coupling**: Each class has a clear role (high cohesion) and minimal dependency on others (low coupling), as discussed in earlier concepts like functional independence.

Example: If you want to change the app's UI (e.g., redesign the MenuScreen), you only modify the boundary class, not the Order entity or OrderController. This makes updates easier and less error-prone.

In Bangla:

এই ক্লাসগুলো গুরুত্বপূর্ণ কারণ:

- এনটিটি: ব্যবসার ধারণা ধরে।
- বাউন্ডারি: ইউজার-বান্ধব ইন্টারফেস দেয়।
- কন্ট্রোলার: সবকিছু সমন্বয় করে।
  এগুলো কনসার্ন আলাদা রাখে, যাতে পরিবর্তন সহজ হয়। যেমন, মেনু স্ক্রিন বদলাতে শুধু

বাউন্ডারি ক্লাসে কাজ হবে।

A Real-World Analogy

Imagine a restaurant:

- **Entity Classes**: The kitchen (prepares food, like Order manages order data).
- **Boundary Classes**: The waiter (shows the menu and takes orders, like MenuScreen displays data).
- **Controller Classes**: The manager (ensures the waiter delivers orders to the kitchen and checks everything runs smoothly, like OrderController coordinates).

Each role is distinct, making the restaurant efficient and easy to manage.

In Bangla:

একটা রেস্তোরাঁর মতো:

- এনটিটি: কিচেন (খাবার তৈরি, অর্ডারের মতো)।
- বাউন্ডারি: ওয়েটার (মেনু দেখায়, অর্ডার নেয়)।
- কন্ট্রোলার: ম্যানেজার (সমন্বয় করে)।
  প্রত্যেকের আলাদা কাজ, যাতে রেস্তোরাঁ সহজে চলে।

Putting It All Together

In the food delivery app:

- **Entity Class (Order)**: Stores order details (items, total) and handles actions like calculating the cost.
- **Boundary Class (MenuScreen)**: Shows the menu and order summary to the user, pulling data from Order.
- **Controller Class (OrderController)**: Manages the flow—creates the Order, updates the MenuScreen, and validates user inputs.

This structure ensures:

- **Modularity**: Each class is a separate module.
- **Information Hiding**: The Order class hides its internal calculations from MenuScreen.

- **Functional Independence**: High cohesion (Order focuses on order logic) and low coupling (MenuScreen doesn’t depend on Order’s internals).
- **Refinement**: The design refines high-level requirements (e.g., “manage orders”) into specific classes and methods.

In Bangla:

ফুড অ্যাপে:

- এনটিটি (Order): অর্ডারের ডেটা আর কাজ ম্যানেজ করে।
- বাউন্ডারি (MenuScreen): মেনু আর অর্ডার দেখায়।
- কন্ট্রোলার (OrderController): সবকিছু সমন্বয় করে।
  এটা মডিউলারিটি, ইনফরমেশন হাইডিং, এবং ইন্ডিপেন্ডেন্স নিশ্চিত করে।

Characteristics of Well-Formed Design Classes

The slide mentions four qualities for good design classes:

- **Complete and Sufficient**: Include all necessary attributes/methods, but no extras.
- **Primitiveness**: Each method does one task (e.g., calculateTotal only calculates, doesn’t save data).
- **High Cohesion**: The class focuses on one responsibility (e.g., Order only handles order details).
- **Low Coupling**: Classes don’t overly depend on each other (e.g., Order doesn’t need to know how MenuScreen works).

---

4. Design Model Elements

What are Design Model Elements?

The **design model** is the blueprint for building the software, detailing how requirements will be implemented. It includes several elements:

- **Data Elements**: Data structures and databases (e.g., how orders are stored).
- **Architectural Elements**: The overall structure, including classes, relationships, and patterns.
- **Interface Elements**: User interfaces (UI), external system connections, and internal module interfaces.

- **Component Elements**: Individual modules or components.
- **Deployment Elements**: How the software runs on hardware (e.g., servers, devices).

Example: In the food delivery app:

- **Data Elements**: A database table for orders (columns: order ID, items, total).
- **Architectural Elements**: The app's structure (e.g., client-server model).
- **Interface Elements**: The app's UI (screens), API for payment gateways, and internal module connections.
- **Component Elements**: The Order module, Payment module, etc.
- **Deployment Elements**: The app runs on a cloud server and user smartphones.

In Bangla:

ডিজাইন মডেল এলিমেন্ট মানে সফটওয়্যার তৈরির নকশার অংশ। যেমন:

- ডেটা: অর্ডারের ডাটাবেস।
- আর্কিটেকচার: অ্যাপের গঠন।
- ইন্টারফেস: ইউজার স্ক্রিন, পেমেন্ট এপিআই।
- কম্পোনেন্ট: অর্ডার, পেমেন্ট মডিউল।
- ডিপ্লয়মেন্ট: ক্লাউড সার্ভারে রান করা।

Architectural Elements in Detail

The slide explains that the **architectural model** comes from:

- **Application Domain**: Knowledge of the problem (e.g., food delivery logistics).
- **Requirements Model**: Analysis classes, data flow diagrams, and relationships.
- **Architectural Patterns and Styles**: Reusable designs (e.g., client-server, MVC).

Example: For the food delivery app, the architecture might use the **Model-View-Controller (MVC)** pattern:

- **Model**: Manages data (e.g., Order class).

- **View**: Displays data (e.g., MenuScreen).
- **Controller**: Handles logic (e.g., processes user clicks).

In Bangla:

আর্কিটেকচারাল এলিমেন্ট সফটওয়্যারের গঠন নির্ধারণ করে। যেমন, ফুড অ্যাপে MVC প্যাটার্ন:

- মডেল: ডেটা ম্যানেজ করে (অর্ডার)।
- ভিউ: ডেটা দেখায় (মেনু স্ক্রিন)।
- কন্ট্রোলার: লজিক হ্যান্ডেল করে (ইউজার ক্লিক)।

Connecting to Separation of Concerns

These concepts tie back to **Separation of Concerns**:

- **Refactoring** improves individual modules without affecting others, keeping concerns separate.
- **OO Design** uses classes to encapsulate concerns (e.g., Order handles order logic only).
- **Design Classes** assign specific roles (e.g., UI vs. business logic), ensuring concerns are distinct.
- **Design Model Elements** organize concerns into data, architecture, and interfaces.

Example: In the food delivery app, the payment concern is handled by:

- A refactored Payment class (clean code).
- An OO Payment class with methods like processPayment.
- A design class (PaymentProcessor) with high cohesion and low coupling.
- A design model element (e.g., payment API interface).

In Bangla:

এই ধারণাগুলো কনসার্নের বিভাজনের সাথে যুক্ত। যেমন, পেমেন্ট কনসার্ন:

- রিফ্যাক্টরিং: কোড পরিষ্কার।
- ওও ডিজাইন: Payment ক্লাস।
- ডিজাইন ক্লাস: PaymentProcessor।
- ডিজাইন মডেল: পেমেন্ট এপিআই।

A Real-World Analogy

Think of building a restaurant:

- **Refactoring**: Rearrange the kitchen to make cooking faster, without changing the menu.
- **OO Design**: Treat the restaurant as objects (e.g., Chef, Table, Order) that interact.
- **Design Classes**:
  - **UI**: The menu board customers see.
  - **Business**: Order (tracks food items).
  - **Process**: Billing (calculates totals).
  - **Persistent**: SalesDatabase (stores transactions).
  - **System**: POSSystem (manages payments).
- **Design Model Elements**:
  - **Data**: Recipe database.
  - **Architecture**: Restaurant layout (kitchen, dining area).
  - **Interfaces**: Customer ordering app, payment terminal.

In Bangla:

একটা রেস্তোরাঁর মতো:

- রিফ্যাক্টরিং: কিচেন সাজানো, মেনু একই।
- ওও ডিজাইন: শেফ, টেবিল, অর্ডার অবজেক্ট।
- ডিজাইন ক্লাস: মেনু বোর্ড, অর্ডার, বিলিং, ডাটাবেস।
- ডিজাইন মডেল: রেসিপি ডেটা, রেস্তোরাঁ লেআউট, অর্ডার অ্যাপ।

Why These Concepts Matter

- **Refactoring**: Keeps code clean, reducing future maintenance costs.
- **OO Design**: Makes software flexible and reusable.
- **Design Classes**: Ensure clear roles for each part, improving maintainability.
- **Design Model Elements**: Provide a complete blueprint for building reliable software.

In Bangla:

রিফ্যাক্টরিং কোড পরিষ্কার রাখে, ওও ডিজাইন নমনীয়তা দেয়, ডিজাইন ক্লাস ভূমিকা স্পষ্ট করে, এবং ডিজাইন মডেল নির্ভরযোগ্য নকশা দেয়।

Project Management and Scheduling

Slide 2: Project Management Spectrum

What's This About?

Managing a software project focuses on **four P's**:

1. **People**: The team building the app.
2. **Product**: The app itself (your freelancing platform).
3. **Process**: The steps to build it (like coding, testing).
4. **Project**: The overall plan to make it happen.

These are the pillars of success, handled by the project manager to meet goals (like launching your app on time).

Example: For your Laravel app, you need skilled coders (People), a clear idea of the app's features (Product), a development process (Process), and a timeline with deadlines (Project).

In Bangla:

প্রজেক্ট ম্যানেজমেন্টে ৪টা P:

- পিপল: টিম যারা অ্যাপ বানাবে।
- প্রোডাক্ট: তোর ফ্রিল্যান্সিং অ্যাপ।
- প্রসেস: কোডিং, টেস্টিং-এর ধাপ।
- প্রজেক্ট: পুরো প্ল্যান।

Slide 3: People

Why People Matter?

People are the heart of the project, not tools or systems. Success depends on picking the right team with the right skills.

Roles:

1. **Senior Manager**: Sets business goals (e.g., “We need an app to compete with Upwork”).
2. **Project Manager**: Plans, motivates, and keeps the team on track.
3. **Software Engineer**: Codes the app (like you with Laravel).
4. **Customer**: Gives requirements (e.g., “I want payment integration”).
5. **End Users**: Use the app (freelancers and clients).

Example: For your app, you’re the software engineer coding the backend. The project manager ensures you finish the payment system on time, while the client says, “Add Stripe.”

In Bangla:

মানুষই প্রজেক্টের মূল। টিমে থাকবে:

- সিনিয়র ম্যানেজার: বিজনেস গোল সেট করে।
- প্রজেক্ট ম্যানেজার: প্ল্যান করে, টিম চালায়।
- সফটওয়্যার ইঞ্জিনিয়ার: তুই, যে কোড করছিস।
- কাস্টমার: রিকোয়ারমেন্ট দেয়।
- এন্ড ইউজার: অ্যাপ ব্যবহার করে।

Slide 4: Project

What’s a Project?

The project is the whole journey—requirements, development, delivery, maintenance, and updates. The project manager guides the team, tracks costs, and ensures deadlines are met to avoid failure.

Example: Your freelancing app project includes gathering client needs (e.g., job posting feature), coding it in Laravel, delivering it, and fixing bugs later. The project manager keeps it all on track.

In Bangla:

প্রজেক্ট মানে পুরো কাজ—রিকোয়ারমেন্ট, কোডিং, ডেলিভারি, মেইনটেনেন্স। প্রজেক্ট ম্যানেজার টিমকে গাইড করে, বাজেট আর ডেডলাইন ঠিক রাখে।

Slide 5: Process

What’s Process?

The process is the roadmap—steps like documentation, design, coding, testing, and deployment. Without a clear process, the team’s lost, like cooking without a recipe.

Steps:

- Documentation (writing requirements).
- Design (planning the app’s structure).
- Implementation (coding in Laravel).
- Software Configuration Management (tracking code changes).
- Deployment (launching the app).
- Interaction (getting user feedback).

Example: For your app, the process starts with documenting client needs, designing the database, coding the job posting feature, testing it, and deploying it on a server.

In Bangla:

প্রসেস মানে কাজের ধাপ—ডকুমেন্টেশন, ডিজাইন, কোডিং, টেস্টিং, ডিপ্লয়মেন্ট। যেমন, তোর অ্যাপের জন্য ক্লায়েন্টের চাহিদা লেখা, ডাটাবেস ডিজাইন, কোডিং, টেস্ট, সার্ভারে লঞ্চ।

Slide 6: Product

What’s the Product?

The product is the final software (your freelancing app). Before planning, define:

- **Objectives and Scope**: What the app does (e.g., job postings, payments).
- **Alternatives**: Other ways to build it (e.g., Django vs. Laravel).
- **Constraints**: Limits like budget or time.

Without these, you can’t estimate costs, risks, or schedules accurately.

Example: Your app’s objective is to connect freelancers with clients. Scope includes job listings and payments. Constraints: $5,000 budget, 3-month deadline.

In Bangla:

প্রোডাক্ট হলো ফাইনাল সফটওয়্যার (তোর অ্যাপ)। আগে ঠিক করতে হবে: গোল (জব পোস্টিং, পেমেন্ট), অল্টারনেটিভ (Django নাকি Laravel), আর লিমিট (বাজেট, সময়)।

Slide 7: Boehm’s W5HH Principle

What’s W5HH?

Barry Boehm’s W5HH (Why, What, When, Who, Where, How, How Much) is a set of questions to plan projects simply and effectively. It’s like a checklist for your app.

Note: The slide lists too many “When” questions, likely a typo. It should be:

1. Why?
2. What?
3. When?
4. Who?
5. Where?
6. How?
7. How Much?

Slides 8-10: W5HH Questions

The Questions:

1. **Why is the system being developed?**
    - Focuses on business reasons. Does the app justify the cost and time?
    - **Example**: Your app solves the problem of connecting freelancers with clients, worth the investment.
    - **Bangla**: কেন অ্যাপ বানাচ্ছিস? ফ্রিল্যান্সারদের ক্লায়েন্টের সাথে কানেক্ট করতে।
2. **What activities are needed?**
    - Lists tasks like coding, testing, designing. Helps set a schedule.
    - **Example**: Tasks for your app: design UI, code backend, integrate Stripe.
    - **Bangla**: কী কী কাজ করতে হবে? UI ডিজাইন, ব্যাকএন্ড কোডিং, Stripe ইন্টিগ্রেশন।
3. **When will it be completed?**
    - Sets start and end dates for tasks.

- **Example**: UI design starts week 1, ends week 3; app launches in 3 months.
- **Bangla**: কখন শেষ হবে? UI ৩ সপ্তাহে, অ্যাপ ৩ মাসে।

4. **Who is responsible for each activity?**
    - Assigns roles (e.g., you code backend, designer does UI).
    - **Example**: You handle Laravel coding, a teammate tests the app.
    - **Bangla**: কে কোন কাজ করবে? তুই ব্যাকএন্ড, টিমমেট টেস্টিং।
5. **Where are they organizationally located?**
    - Notes where team members are (e.g., in-house or client's team).
    - **Example**: Your coders are in Dhaka, but the client is in the USA.
    - **Bangla**: কারা কোথায় আছে? তোর টিম ঢাকায়, ক্লায়েন্ট আমেরিকায়।
6. **How will the job be done technically and managerially?**
    - Defines technical (e.g., Laravel) and management (e.g., Agile) strategies.
    - **Example**: Use Laravel for backend, Agile with weekly sprints.
    - **Bangla**: কীভাবে করবি? Laravel দিয়ে কোডিং, Agile দিয়ে ম্যানেজ।
7. **How much of each resource is needed?**
    - Estimates resources like developers, servers, budget.
    - **Example**: 3 developers, 1 server, $5,000 for your app.
    - **Bangla**: কত রিসোর্স লাগবে? ৩ জন ডেভেলপার, ১টা সার্ভার, ৫০০০ ডলার।

Why It's Useful: W5HH ensures you've thought through every angle of your app project, from purpose to resources.

Slide 11: Software Measurements

What's This?

Measurements quantify aspects of a project (e.g., size, effort) to make decisions. They're like checking your app's progress with a ruler.

Two Types:

1. **Direct Measures**: Concrete stuff like:

- Cost (money spent).
- Effort (hours worked).
- Lines of Code (LOC).
- Errors found.

2. **Indirect Measures**: Abstract stuff like:
   - Functionality (features working).
   - Quality (bug-free).
   - Complexity (how tricky the code is).

Example: For your app, direct measures: 10,000 LOC, $5,000 spent, 5 bugs. Indirect: good UX (quality), complex payment system.

In Bangla:

মেজারমেন্ট মানে প্রজেক্টের পরিমাণ। ডাইরেক্ট: LOC, খরচ, বাগ। ইনডাইরেক্ট: কোয়ালিটি, কমপ্লেক্সিটি। যেমন, তোর অ্যাপে ১০,০০০ লাইন কোড, ৫টা বাগ, ভালো UX।

Slide 12: Software Metrics

What Are Metrics?

Metrics are formulas to measure project aspects, helping plan and track progress.

Three Types:

1. **Product Metrics**: Measure the app (size, quality, reliability).
   - Example: Your app’s size (10,000 LOC), few bugs (high quality).
2. **Process Metrics**: Measure development (e.g., time to test, defect rate).
   - Example: Testing took 2 weeks, found 5 bugs.
3. **Project Metrics**: Measure management (e.g., number of developers, cost).
   - Example: 3 developers, $5,000 budget.

In Bangla:

মেট্রিক্স মানে পরিমাপের ফর্মুলা। প্রোডাক্ট: অ্যাপের সাইজ, কোয়ালিটি। প্রসেস: টেস্টিং সময়, বাগ রেট। প্রজেক্ট: ডেভেলপার সংখ্যা, খরচ।

Slide 13: Principles of Software Measurement

How to Measure?

1. **Formulation**: Choose what to measure (e.g., LOC for size).
2. **Collection**: Gather data (e.g., count lines of code).
3. **Analysis**: Calculate metrics (e.g., LOC per developer).
4. **Interpretation**: Understand results (e.g., high LOC means complex app).
5. **Feedback**: Suggest improvements (e.g., simplify code).

Example: For your app, measure LOC (10,000), analyze it (2 developers wrote it), interpret (each wrote 5,000 lines), and suggest pair programming to improve.

In Bangla:

মেজারমেন্টের ধাপ:

- ফর্মুলেশন: কী মাপবি (LOC)।
- কালেকশন: ডেটা জমা (কোড লাইন গণনা)।
- এনালাইসিস: হিসাব (ডেভেলপার প্রতি LOC)।
- ইন্টারপ্রিটেশন: বোঝা (বেশি LOC মানে জটিল)।
- ফিডব্যাক: উন্নতি (পেয়ার প্রোগ্রামিং)।

Slides 14-15: Size Metrics: LOC

What’s LOC?

Lines of Code (LOC) measures program size by counting code lines (excluding comments/blank lines). It’s simple but has flaws.

Example: A C++ program with 9 LOC (shown in slide). For your app, if you write 10,000 lines in Laravel, that’s your LOC.

Advantages:

- Easy to estimate costs.
- Simple to measure.

Disadvantages:

- Doesn’t measure requirements size.
- Bad design can inflate LOC.

- Language-dependent (PHP vs. C++ differs).
- Hard for non-coders to understand.

Table Example:

- Project A: 10,000 LOC, 110 cost units, 18 errors.
- Project B: 12,000 LOC, 115 cost units, 20 errors.

In Bangla:

LOC মানে কোডের লাইন গণনা। যেমন, তোর অ্যাপে ১০,০০০ লাইন। সুবিধা: খরচ হিসাব সহজ। অসুবিধা: খারাপ ডিজাইনে LOC বাড়ে, ল্যাঙ্গুয়েজের উপর নির্ভর করে।

Slides 16-17: Size Metrics: FP

What's FP?

Function Points (FP) measure functionality (what the app does) by counting inputs, outputs, inquiries, files, and interfaces. It's better for non-coders.

FP Attributes:

1. External Inputs (e.g., user forms).
2. External Outputs (e.g., reports).
3. External Inquiries (e.g., search queries).
4. Internal Files (e.g., databases).
5. External Interfaces (e.g., APIs like Stripe).

Example: For your app:

- Inputs: Job posting form.
- Outputs: Job listing page.
- Inquiries: Search jobs.
- Files: User database.
- Interfaces: Stripe API.
- Total FP = 400 (from slide table).

Advantages:

- Works with requirements.
- Language-independent.

- Detailed specs needed.

Disadvantages:

- Ignores quality.
- Subjective counting.

In Bangla:

FP মানে ফাংশনালিটি মাপা (ইনপুট, আউটপুট, ফাইল)। যেমন, তোর অ্যাপে জব ফর্ম, সার্চ, ডাটাবেস। সুবিধা: ল্যাঙ্গুয়েজ ফ্রি। অসুবিধা: কোয়ালিটি মাপে না।

Slide 18: Software Project Estimation

Why Estimate?

Estimation predicts time and cost to complete a project. It’s tough but critical to avoid failure.

Who’s Responsible?

- Software Manager.
- Engineers (like you).
- Estimators.

Example: Estimating your app’s cost ($5,000?) and time (3 months?) using tools like LOC or FP.

In Bangla:

এস্টিমেশন মানে সময় আর খরচ হিসাব। তুই, ম্যানেজার, এস্টিমেটর মিলে করবি। যেমন, তোর অ্যাপে ৫০০০ ডলার, ৩ মাস।

Slide 19: Factors Affecting Project Estimation

Key Factors:

1. **Cost**: Ensure enough funds ($5,000 for your app).
2. **Time**: Overall duration and task timing (3 months total, 2 weeks for UI).
3. **Size & Scope**: All tasks needed (e.g., coding, testing).
4. **Risk**: Predict problems (e.g., API failure).
5. **Resources**: People, tools, servers.

Example: If your app's budget is short, you might skip advanced features. Risks like client changes need a plan.

In Bangla:

এস্টিমেশনের ফ্যাক্টর:

- খরচ: ৫০০০ ডলার আছে কি না।
- সময়: ৩ মাস, UI-এ ২ সপ্তাহ।
- সাইজ: সব কাজ (কোডিং, টেস্টিং)।
- রিস্ক: এপিআই ফেল।
- রিসোর্স: ডেভেলপার, সার্ভার।

Slides 20-24: Steps of Software Project Estimation

Four Steps:

1. **Estimate Project Size (LOC & FP)**:
    - Use requirements, SRS, or past projects to estimate size.
    - Example: Your app might be 10,000 LOC or 400 FP, based on similar apps.
2. **Estimate Efforts (Person-Months/Hours)**:
    - Calculate work needed (e.g., coding, testing).
    - Use historical data or models like COCOMO.
    - Example: Your app needs 18 person-months (3 developers x 6 months).
3. **Estimate Project Schedule (Months)**:
    - Assign tasks to people, create a Work Breakdown Structure (WBS).
    - Formula: Schedule = 3.0 * (man-months)^(1/3).
    - Example: 18 man-months = ~6 months.
4. **Estimate Project Cost**:
    - Include labor (hours x rate), hardware, software, travel.
    - Example: 3 developers at $20/hour, 1,000 hours = $20,000 + $1,000 for servers.

In Bangla:

৪টা ধাপ:

1. সাইজ: ১০,০০০ LOC বা ৪০০ FP।
2. এফোর্ট: ১৮ ম্যান-মান্থ (৩ জন x ৬ মাস)।
3. শিডিউল: ১৮ ম্যান-মান্থ = ৬ মাস।
4. খরচ: ৩ জন x ১০০০ ঘণ্টা x ২০ ডলার + ১০০০ ডলার সার্ভার = ২১,০০০ ডলার।

Slides 25-28: Decomposition Techniques

What’s Decomposition?

Break the project into smaller parts to estimate cost, time, and effort.

Techniques:

1. **Software Sizing**: Estimate product size using:
   - **Fuzzy Logic**: Guess based on app type (e.g., web app).
   - **Function Point**: Count FP (inputs, outputs).
   - **Standard Component**: Estimate based on modules (e.g., UI, database).
   - **Change Sizing**: For modifying existing code.
2. **Problem-Based Estimation**: Use LOC/FP to size each part, then estimate effort.
3. **Process-Based Estimation**: Estimate effort for each process step (e.g., coding, testing).

Example: For your app, size the job posting module (2,000 LOC), estimate effort (4 person-months), and assign tasks (2 weeks for coding, 1 for testing).

In Bangla:

ডিকম্পোজিশন মানে প্রজেক্ট ভাগ করা।

- সাইজিং: LOC, FP, মডিউল দিয়ে সাইজ হিসাব।
- প্রবলেম-বেসড: মডিউলের LOC/FP দিয়ে এফোর্ট।
- প্রসেস-বেসড: কোডিং, টেস্টিং-এর এফোর্ট।

Slides 29-30: Problem-Based Estimation

How It Works:

- Use LOC/FP to size each module.
- Compute expected value: S = (s_opt + 4s_m + s_pess) / 6.
- Use past project data to estimate effort.

Example: For your app's job posting module:

- Optimistic: 1,800 LOC.
- Likely: 2,000 LOC.
- Pessimistic: 2,500 LOC.
- Expected: (1,800 + 4*2,000 + 2,500) / 6 = 2,050 LOC.
- Effort: Based on past projects, 2,050 LOC = 4 person-months.

In Bangla:

প্রবলেম-বেসড এস্টিমেশন: LOC/FP দিয়ে মডিউল সাইজ। ফর্মুলা: (অপটিমিস্টিক + ৪*লাইকলি + পেসিমিস্টিক) / ৬। যেমন, জব পোস্টিং: ২,০৫০ LOC = ৪ ম্যান-মান্থ।

Slide 31: Process-Based Estimation

How It Works:

- Break the process into tasks (e.g., design, code, test).
- Estimate effort for each task.
- Apply labor rates to calculate cost.

Example: For your app:

- Design: 2 person-months, $30/hour.
- Coding: 10 person-months, $20/hour.
- Testing: 3 person-months, $25/hour.
- Total cost = (2*160*$30) + (10*160*$20) + (3*160*$25) = $41,600.

In Bangla:

প্রসেস-বেসড: কাজ ভাগ কর (ডিজাইন, কোড, টেস্ট)। প্রতিটার এফোর্ট হিসাব। যেমন, ডিজাইন ২ ম্যান-মান্থ, কোডিং ১০, টেস্টিং ৩। খরচ = ৪১,৬০০ ডলার।

Slide 32: Software Cost Estimation

What's This?

Predicting the approximate cost before development. Considers:

- Specification & Scope.
- Location (e.g., Dhaka vs. USA).
- Duration.
- Team Efforts.
- Resources.

Example: Your app's cost depends on features (scope), developer rates in Dhaka, and 3-month duration.

In Bangla:

কস্ট এস্টিমেশন মানে ডেভেলপমেন্টের আগে খরচ হিসাব। ফ্যাক্টর: ফিচার, লোকেশন (ঢাকা), সময় (৩ মাস), টিম, রিসোর্স।

Slides 33-36: Tools and Techniques of Software Cost Estimation

Techniques:

1. **Expert Judgment**: Ask experienced people (e.g., a senior Laravel dev).
2. **Analogous Estimation**: Use data from similar projects (e.g., another Laravel app).
3. **Parametric Estimation**: Use past project data to estimate man-hours.
4. **Bottom-Up Estimation**: Estimate each task (WBS) and sum up.
5. **Three-Point Estimation (PERT)**: Use optimistic, likely, pessimistic estimates.
6. **Reserve Analysis**: Keep a budget for risks.
7. **Cost of Quality**: Include costs to avoid/fix failures.
8. **Vendor Bid Analysis**: Compare vendor quotes.

Example: For your app, use analogous estimation (past Laravel projects) and PERT (optimistic: 2 months, likely: 3, pessimistic: 4).

In Bangla:

টুলস:

- এক্সপার্ট জাজমেন্ট: সিনিয়র ডেভের পরামর্শ।

- অ্যানালগাস: আগের প্রজেক্টের ডেটা।
- প্যারামেট্রিক: ম্যান-আওয়ার হিসাব।
- বটম-আপ: প্রতি কাজের খরচ।
- থ্রি-পয়েন্ট: অপটিমিস্টিক, লাইকলি, পেসিমিস্টিক।

Slide 37: Typical Problems with Cost Estimation

Issues:

1. Estimating large projects is complex.
2. Inexperienced estimators.
3. Underestimation bias (seniors forget juniors' skills).
4. Forgetting integration/testing costs.
5. Management wanting firm numbers early.

Example: You might underestimate your app's testing time, thinking it's simple, but integration with Stripe takes extra effort.

In Bangla:

সমস্যা:

- বড় প্রজেক্ট এস্টিমেট জটিল।
- নতুন লোকের ভুল।
- কম হিসাব করা।
- ইন্টিগ্রেশন/টেস্টিং ভুলে যাওয়া।
- ম্যানেজমেন্টের তাড়া।

Slides 38-50: COCOMO Model

What's COCOMO?

Constructive Cost Model (by Barry Boehm, 1981) estimates effort, time, and cost based on LOC.

Project Types (Slides 39-40):

1. **Organic**: Small, simple (2-50 KLOC, e.g., inventory system).
2. **Semidetached**: Medium, mixed complexity (50-300 KLOC, e.g., DBMS).

3. **Embedded**: Large, complex (300+ KLOC, e.g., banking software).

Your App: Likely semidetached (medium size, Laravel-based).

COCOMO Types (Slides 42-47):

1. **Basic COCOMO**:
    - Simple, uses LOC.
    - Effort = a * (KLOC)^b.
    - Time = c * (Effort)^d.
    - Example (Slide 43): Semidetached, 300 KLOC:
        - Effort = 3.0 * (300)^1.12 = 1784.42 man-months.
        - Time = 2.5 * (1784.42)^0.35 = 34.35 months.
        - Persons = 1784.42 / 34.35 ≈ 52.
2. **Intermediate COCOMO**:
    - Adds Effort Adjustment Factor (EAF) for accuracy.
    - Example (Slide 46): 300 KLOC, EAF = 0.9348 (high app experience, low programming experience).
        - Effort = 3.0 * (300)^1.12 * 0.9348 = 1668.07 man-months.
        - Time = 2.5 * (1668.07)^0.35 = 33.55 months.
3. **Detailed COCOMO**:
    - Breaks project into modules and phases (planning, design, coding, testing).
    - Example (Slide 48): MIS system with database (semidetached), GUI (organic), communication (embedded).

Advantages (Slide 49):

- Systematic estimation.
- Works at different stages.
- Uses historical data.

Disadvantages (Slide 50):

- Ignores customer skills, hardware.

- Relies on assumptions.
- Focuses on size.

In Bangla:

COCOMO মানে খরচ, সময় হিসাব। ৩ ধরন:

- অর্গানিক: ছোট (ইনভেন্টরি)।
- সেমিডিটাচড: মাঝারি (তোর অ্যাপ)।
- এমবেডেড: বড় (ব্যাঙ্কিং)।
  বেসিক: LOC দিয়ে হিসাব। ইন্টারমিডিয়েট: EAF যোগ। ডিটেইলড: মডিউল ভাগ।

How to Use This for Your Exam

1. **Explain the 4 P's**: People (team roles), Product (app features), Process (development steps), Project (management).
2. **W5HH**: List the 7 questions and give an example for your app (e.g., Why: connect freelancers).
3. **LOC vs. FP**: LOC is simple but language-dependent; FP measures functionality, better for clients.
4. **COCOMO**: Explain Basic (simple LOC-based), Intermediate (EAF for accuracy), Detailed (module-based). Use the semidetached example.
5. **Estimation Steps**: Size (LOC/FP), Effort (man-months), Schedule (calendar months), Cost (labor + resources).

Example Answer:

"Project management focuses on the 4 P's: People (e.g., coders for my Laravel app), Product (freelancing platform), Process (coding, testing), and Project (timeline). W5HH helps plan by asking Why (connect freelancers), What (tasks like UI design), etc. For estimation, I'd use FP (400 for my app) to size it, estimate 18 man-months effort, 6 months schedule, and $21,000 cost. COCOMO's Intermediate model adjusts effort with EAF for accuracy."

In Bangla:

এক্সামে বলবি: ৪ P (পিপল, প্রোডাক্ট, প্রসেস, প্রজেক্ট)। W5HH-এর ৭ প্রশ্ন (কেন: ফ্রিল্যান্সার কানেক্ট)। LOC সহজ কিন্তু ল্যাঙ্গুয়েজ নির্ভর, FP ফাংশনালিটি মাপে। COCOMO: বেসিক (LOC), ইন্টারমিডিয়েট (EAF), ডিটেইলড (মডিউল)। এস্টিমেশন: সাইজ, এফোর্ট, শিডিউল, কস্ট।

Software Testing Lecture Notes

Slide 1: Introduction to Software Testing

Overview:

This slide introduces software testing, a critical process to ensure software functionality and quality. Testing verifies that a software product, such as a freelancing application, meets specified requirements.

Key Point:

Testing is analogous to quality assurance, ensuring the software performs as intended before deployment.

In Bangla:

সফটওয়্যার টেস্টিং মানে অ্যাপ ঠিকমতো কাজ করে কি না চেক করা।

Slide 2: Principles of Software Testing

Definition:

Software testing is a method to verify if a software product aligns with expected requirements and is free of defects. It identifies errors, gaps, or missing features by comparing actual outcomes to requirements.

Key Points:

- **Purpose**: Detect errors, ensure reliability, scalability, portability, re-usability, and usability.
- **Importance**: Mandatory to prevent software failures that could lead to critical issues (e.g., payment system crashes).
- **Scope**: Conducted at every Software Development Life Cycle (SDLC) phase (requirements, design, coding, testing, deployment).
- **Stakeholders**: Software testers, developers, project managers, and end-users.

Example:

For a freelancing app, testing ensures the job posting feature is reliable (no crashes) and user-friendly for freelancers.

In Bangla:

টেস্টিং চেক করে অ্যাপ ক্লায়েন্টের চাহিদা মতো কি না, বাগ-ফ্রি কি না। রিলায়েবিলিটি, স্কেলেবিলিটি, ইউজেবিলিটি টেস্ট হয়। SDLC-র সব ফেজে টেস্ট হয়। টেস্টার, ডেভেলপার, ম্যানেজার, ইউজার সবাই টেস্ট করে।

Slide 3: Types of Software Testing

Overview:

This slide serves as a placeholder for various testing types, detailed in subsequent slides.

Key Point:

Testing types vary based on scope, methodology, and objectives (e.g., unit, integration, black box, white box).

In Bangla:

টেস্টিং-এর বিভিন্ন ধরন, পরে বিস্তারিত আসবে।

Slide 4: Test Case

Definition:

A test case is a documented set of actions or conditions used to verify if a software component meets requirements by comparing expected and actual results.

Components (from slide example):

- **Test Scenario ID**: Unique identifier (e.g., Login-1).
- **Test Case ID**: Specific test (e.g., Login-1A).
- **Description**: Purpose (e.g., positive login test).
- **Priority**: Importance (e.g., High).
- **Pre/Post-Requisites**: Conditions (e.g., valid user account).
- **Execution Steps**: Actions, inputs, expected/actual outputs, result (pass/fail).

Example Table:

- **Scenario**: Login to Facebook.
- **Steps**:
    1. Launch facebook.com → Expected: Homepage → Actual: Homepage → Pass.
    2. Enter valid email (test@xyz.com) and password, click login → Expected: Login success → Actual: Login success → Pass.

Application Example:

For the freelancing app, a test case for login:

- **Step 1**: Open login page → Expected: Login form loads.
- **Step 2**: Enter valid email/password → Expected: Dashboard loads.
- **Result**: Pass if actual matches expected; fail indicates a bug.

In Bangla:

টেস্ট কেস মানে স্টেপ লিখে চেক করা অ্যাপ ঠিক কাজ করে কি না। যেমন, লগইন টেস্ট: URL ওপেন, ইমেইল-পাসওয়ার্ড দাও, ড্যাশবোর্ড আসার কথা। পাস হলে ঠিক, না হলে বাগ।

Slide 5: Principles of Software Testing (Contd.)

Note:

This slide repeats the title from Slide 2, likely a typo. Principles are detailed in Slides 6-9.

Slide 6: Principles 1 & 2

1. Testing Shows Presence of Defects:

- **Concept**: Testing identifies defects but cannot guarantee a 100% defect-free product. It reduces undiscovered bugs through effective test cases.
- **Example**: Testing the freelancing app's payment system identifies Stripe integration errors, minimizing risks.

2. Exhaustive Testing is Impossible:

- **Concept**: Testing every possible scenario is impractical due to time and resource constraints. Focus on high-risk areas and prioritize critical features.

- **Example**: Prioritize testing job posting and payment features over minor UI elements.

In Bangla:

- **ডিফেক্ট দেখায়**: টেস্টিং বাগ খুঁজে, কিন্তু ১০০% বাগ-ফ্রি নয়। ভালো টেস্ট কেস দিয়ে বাগ কমাও।
- **এক্সহস্টিভ টেস্টিং অসম্ভব**: সব টেস্ট করা যায় না। জব পোস্ট, পেমেন্ট বেশি টেস্ট কর।

Slide 7: Principles 3 & 4

3. Early Testing:

- **Concept**: Begin testing during requirements or design phases. Early defect detection reduces fixing costs and time.
- **Example**: Identify a missing field in the job posting feature during requirements analysis, avoiding costly code changes later.

4. Defect Clustering:

- **Concept**: Most defects (80%) occur in a small portion (20%) of code (Pareto's 80-20 rule). Focus testing on critical modules.
- **Example**: The payment module may contain most bugs, requiring intensive testing.

In Bangla:

- **আর্লি টেস্টিং**: রিকোয়ারমেন্ট ফেজে টেস্ট শুরু। আগে বাগ ধরলে খরচ কম।
- **ডিফেক্ট ক্লাস্টারিং**: ৮০% বাগ ২০% কোডে। পেমেন্ট মডিউল বেশি টেস্ট কর।

Slide 8: Principles 5 & 6

5. Pesticide Paradox:

- **Concept**: Repeated use of the same test cases fails to identify new defects. Regularly update test cases to uncover fresh issues.
- **Example**: If login tests only use valid credentials, add tests for invalid inputs to find new bugs.

6. Testing is Context-Dependent:

- **Concept**: Testing varies by application type (e.g., e-commerce, banking, freelancing). Each requires tailored test cases.
- **Example**: The freelancing app needs tests for job postings, while a banking app focuses on security.

In Bangla:

- **পেস্টিসাইড প্যারাডক্স**: একই টেস্ট কেস বারবার চালালে নতুন বাগ পাওয়া যায় না। নতুন টেস্ট লেখ।
- **কনটেক্সট-ডিপেন্ডেন্ট**: ফ্রিল্যান্সিং অ্যাপে জব পোস্ট টেস্ট, ব্যাঙ্কিং অ্যাপে সিকিউরিটি।

Slide 9: Principle 7

7. Absence of Errors Fallacy:

- **Concept**: A defect-free application is insufficient if it fails to meet user needs or is impractical. Testing must validate functionality and usability.
- **Example**: A bug-free freelancing app fails if the job posting form is confusing for users.

In Bangla:

বাগ না থাকলেও অ্যাপ ইউজারের চাহিদা না মিটালে লাভ নেই। যেমন, জব ফর্ম কঠিন হলে ফেইল।

Slide 10: White Box Testing

Definition:

White box testing examines the internal code structure and logic, requiring programming knowledge. It is typically performed by developers during unit and integration testing.

Aliases: Clear box, Open box, Transparent box, Code-based, Glass box testing.

Tools: EclEmma, NUnit, PyUnit, HTMLUnit, CppUnit.

Example:

Test the Laravel code for the job posting function to ensure it correctly saves data to the database.

In Bangla:

হোয়াইট বক্স মানে কোডের ভেতর চেক করা। ডেভেলপার করে। যেমন, জব পোস্ট ফাংশনের কোড টেস্ট।

Slide 11: Verification in White Box Testing

Aspects Verified:

- Code security (e.g., no vulnerabilities).
- Code paths (all logic flows).
- Input processing through code.
- Loops, conditions, and statements.
- Individual functions/modules.
- Expected outputs.

Example:

Test the payment function’s Stripe API call to verify input handling and retry loops.

In Bangla:

কোডের সিকিউরিটি, পাথ, লুপ, কন্ডিশন, ফাংশন চেক। যেমন, পেমেন্টে Stripe API ঠিক কাজ করে কি না।

Slide 12-13: White Box Testing Techniques

Techniques:

1. **Path Coverage**: Test all possible code paths from entry to exit.
    - **Example**: Test job posting function for valid/invalid inputs.
2. **Loop Testing**: Verify loops (e.g., while, for) for correct execution and termination.
    - **Example**: Test a payment retry loop (e.g., 3 attempts).
3. **Branch/Condition Testing**: Test true/false conditions in if/else statements.
    - **Example**: Test if logged-in users see the dashboard, others see the login page.
4. **Statement Coverage**: Ensure every code line executes at least once.

- **Example**: Test all lines in the payment function.

Code Example (Slide 13):

```
if (result > 0)
  Print("Positive", result);
else
  Print("Negative", result);
```

- Test with result = 5 (positive) and result = -5 (negative) to cover both branches.

In Bangla:

- **পাথ কভারেজ**: সব কোড পাথ টেস্ট।
- **লুপ টেস্টিং**: লুপ ঠিক কাজ করে কি না।
- **ব্রাঞ্চ টেস্টিং**: if/else চেক।
- **স্টেটমেন্ট কভারেজ**: প্রতি লাইন কোড চালাও।

Slide 14: White Box Testing - Advantages & Disadvantages

Advantages:

- Identifies hidden code errors.
- Supports test automation.
- Detects defects early, improving quality.
- Covers most code paths.

Disadvantages:

- Complex, costly, and time-consuming.
- Requires skilled programmers.
- Cannot detect missing functionalities.
- Code changes necessitate new test cases.

Example:

White box testing finds a payment bug in the freelancing app early but is time-intensive and misses absent features.

In Bangla:

সুবিধা: লুকানো বাগ ধরে, অটোমেট করা যায়, কোয়ালিটি ভালো।

অসুবিধা: জটিল, খরচ বেশি, নতুন ফিচার মিস করে।

Slide 15: Black Box Testing

Definition:

Black box testing evaluates software functionality without knowledge of internal code structure, focusing on inputs and outputs. It is performed by software testers.

Aliases: Behavioral, Functional, Closed box testing.

Tools: QTP, Selenium, Loadrunner, Jmeter.

Example:

Test the freelancing app's login feature by entering an email/password and verifying successful login, ignoring Laravel code.

In Bangla:

ব্ল্যাক বক্স মানে কোড না জেনে ফাংশনালিটি টেস্ট। ইনপুট দাও, আউটপুট চেক। যেমন, লগইনে ইমেইল-পাসওয়ার্ড চেক।

Slide 16: Types of Black Box Testing

Overview:

Placeholder for black box testing techniques, detailed in Slides 17-19.

In Bangla:

ব্ল্যাক বক্সের ধরন, পরে বিস্তারিত।

Slide 17-19: Black Box Testing Techniques

Techniques:

1. **Equivalence Partitioning**: Divide inputs into groups with similar outcomes, testing one from each.

- **Example**: For age input (18-60), test 17 (invalid), 25 (valid), 61 (invalid).

2. **Boundary Value Analysis**: Test boundary values (min/max).
   - **Example**: For age 18-56, test 17, 18, 19, 55, 56, 57.
3. **Decision Table Testing**: Test input combinations in a table.
   - **Example**: For Gmail login, test valid/invalid email-password pairs.
4. **Error Guessing**: Predict problem areas based on experience.
   - **Example**: Test file uploads with no file or oversized files.
5. **State Transition Testing**: Test behavior changes with inputs.
   - **Example**: Test login with 3 incorrect passwords to verify account lockout.
6. **All Pairs Testing**: Test combinations of inputs (e.g., form fields).
   - **Example**: Test job posting form with various field combinations.

Application Example:

Use boundary value analysis for job bids ($0, $1, $1000) or error guessing for payment failures.

In Bangla:

- **ইকুইভ্যালেন্স পার্টিশনিং**: ইনপুট গ্রুপ করে টেস্ট।
- **বাউন্ডারি ভ্যালু**: মিন-ম্যাক্স টেস্ট।
- **ডিসিশন টেবিল**: ইনপুট কম্বিনেশন।
- **এরর গেসিং**: সমস্যার জায়গা আন্দাজ।
- **স্টেট ট্রানজিশন**: ইনপুটে বদল।
- **অল পেয়ার্স**: ফর্ম কম্বিনেশন।

Slide 20: Black Box Testing - Advantages & Disadvantages

Advantages:

- Requires no programming knowledge.
- Effective for large systems.

- Tests from the user’s perspective.
- Identifies unclear requirements.

Disadvantages:

- Designing test cases without code knowledge is challenging.
- Misses code structure errors.
- Time-consuming for large input sets.

Example:

Black box testing verifies the freelancing app’s UI but misses code logic errors.

In Bangla:

সুবিধা: কোড জানা লাগে না, ইউজার ভিউ থেকে টেস্ট।

অসুবিধা: কোডের ভুল ধরে না, সময় লাগে।

Slide 21-24: Black Box vs. White Box Testing

Comparison:

Aspect	Black Box Testing	White Box Testing
Knowledge	No code knowledge	Requires code knowledge
Aliases	Functional, Behavioral	Structural, Glass box
Performed By	Testers	Developers
Testing Level	System, Acceptance	Unit, Integration
Time	Less time-consuming	More time-consuming
Focus	Input-output functionality	Code paths and logic
Example	Test Google search	Test code loops

Application Example:

Use black box to test job posting functionality (user view) and white box to verify Laravel code logic.

In Bangla:

ব্ল্যাক বক্স: কোড জানা লাগে না, টেস্টার করে, ফাংশনালিটি। হোয়াইট বক্স: কোড জানতে হয়, ডেভেলপার করে, কোড লজিক।

Slide 25-29: Unit Testing

Definition:

Unit testing verifies individual components (e.g., functions, modules) of software during the coding phase, using white box techniques. It is performed by developers.

Purpose:

- Validates code correctness.
- Tests every function/procedure.
- Detects bugs early, reducing costs.
- Aids documentation and code reuse.
- Enhances development efficiency.

In Object-Oriented Context:

Tests classes, methods, and attributes (e.g., public/private).

Example:

Test the freelancing app's postJob() function to ensure it saves job details correctly.

Advantages:

- Tests components independently.
- Focuses on functionality.
- Early defect detection.
- Improves software quality.
- Speeds up development.

Disadvantages:

- Time-intensive to create/maintain test cases.
- Limited to individual units, missing interactions.
- Requires ongoing maintenance.

Tools: JUnit, NUnit, PHPUnit (suitable for Laravel).

In Bangla:

ইউনিট টেস্টিং মানে কোডের ছোট অংশ (ফাংশন) টেস্ট। ডেভেলপার করে, হোয়াইট বক্স। যেমন, postJob() টেস্ট। **সুবিধা**: বাগ আগে ধরা, কোয়ালিটি ভালো। **অসুবিধা**: সময় লাগে, মডিউল ইন্টারঅ্যাকশন মিস।

Slide 30-32: Integration Testing

Definition:

Integration testing, conducted after unit testing, verifies interactions between integrated modules. It is also known as thread or string testing.

Purpose:

- Ensures module compatibility despite different developers' logic.
- Validates database interactions.
- Addresses changed requirements.
- Tests hardware-software compatibility.

Example:

Test if the freelancing app's login module correctly passes data to the job posting module.

Tools: Selenium, PyTest, JUnit, Jasmine, Mockito.

In Bangla:

ইন্টিগ্রেশন টেস্টিং মানে মডিউল একসাথে কাজ করে কি না চেক। যেমন, লগইন থেকে জব পোস্ট ঠিক যায় কি না।

Slide 33-37: Types of Integration Testing

1. Incremental Integration Testing:

- **Concept**: Integrates and tests modules one by one, ensuring logical relationships.
- **Subtypes**:
    - **Top-Down**: Begins with high-level modules, using stubs (dummy modules) for undeveloped lower modules.
        - **Example**: Test login, then job posting, with a stub for payments.

    - **Bottom-Up**: Starts with low-level modules, using drivers (dummy callers) for undeveloped higher modules.
        - **Example**: Test payment, then job posting, with a driver for login.
- **Application Example**: For a Flipkart-like flow: Login → Home → Search → Payment.

2. Non-Incremental (Big Bang) Testing:

- **Concept**: Integrates and tests all modules simultaneously. Suitable for small systems but challenging for debugging.
- **Example**: Test the entire freelancing app (login, job posting, payment) at once.

In Bangla:

- **ইনক্রিমেন্টাল**: একটা একটা মডিউল টেস্ট। টপ-ডাউন (লগইন দিয়ে শুরু, স্টাব), বটম-আপ (পেমেন্ট দিয়ে, ড্রাইভার)।
- **বিগ ব্যাং**: সব মডিউল একসাথে। ছোট সিস্টেমে ভালো, বাগ ধরা কঠিন।

Slide 38: Incremental vs. Non-Incremental Testing

Comparison:

Aspect	Incremental Testing	Non-Incremental Testing
Approach	Gradual, module-by-module	All modules at once
Planning	Requires detailed scheduling	Simpler, single test
Resources	Higher (separate tests)	Lower (one test)
Issue Detection	Early detection	Late detection
Complexity	Lower (smaller units)	Higher (entire system)

Application Example:

Incremental testing is preferred for the freelancing app to identify bugs early in critical modules like login or payment.

In Bangla:

ইনক্রিমেন্টাল: ধাপে ধাপে, বাগ আগে ধরা, বেশি রিসোর্স। নন-ইনক্রিমেন্টাল: একবারে, সহজ কিন্তু বাগ দেরিতে ধরা।

Slide 39: Conclusion

Overview:

A closing slide, marking the end of the presentation.

Exam Preparation Guide

To excel in your exam, focus on the following key areas with examples from the freelancing app:

1. **Seven Principles of Testing**:
    - **Explanation**: Early testing reduces costs; defect clustering targets high-risk modules (e.g., payment).
    - **Example Answer**: “Early testing during requirements catches job posting design flaws, saving costs. Defect clustering suggests 80% of bugs are in 20% of code, like the payment module.”
2. **Test Case**:
    - **Explanation**: Define a test case with steps, expected/actual results.
    - **Example Answer**: “A login test case: Open login page, enter valid email/password, expect dashboard. If actual matches, it passes.”
3. **White Box vs. Black Box Testing**:
    - **Explanation**: White box tests code (developers), black box tests functionality (testers).
    - **Example Answer**: “White box tests the Laravel postJob() function’s logic, while black box tests the job posting form’s functionality using boundary value analysis.”
4. **Unit Testing**:
    - **Explanation**: Tests individual components, performed by developers, catches bugs early.
    - **Example Answer**: “Unit testing verifies the postJob() function saves data correctly, using PHPUnit for Laravel.”
5. **Integration Testing**:
    - **Explanation**: Tests module interactions, with incremental (top-down/bottom-up) or big bang approaches.

- **Example Answer**: “Incremental top-down testing checks login to job posting flow, using stubs for payments, ensuring early bug detection.”

6. **Testing Techniques**:
    - **Explanation**: White box: path, loop, branch, statement coverage. Black box: equivalence partitioning, boundary value analysis, decision tables.
    - **Example Answer**: “Path coverage tests all job posting function paths. Boundary value analysis tests job bids at $0, $1, $1000.”

Sample Exam Answer:

“Software testing ensures the freelancing app meets client requirements and is defect-free. The seven principles guide effective testing: early testing reduces costs, defect clustering focuses on critical modules like payments (80% bugs in 20% code). White box testing, used in unit testing, verifies Laravel code logic (e.g., postJob() function), while black box testing checks functionality (e.g., job posting form) using techniques like boundary value analysis. Integration testing, preferably incremental top-down, ensures login and payment modules interact correctly, catching bugs early.”

In Bangla:

এক্সামে বলো: টেস্টিং অ্যাপ ঠিক কি না চেক করে। ৭ প্রিন্সিপল: আর্লি টেস্টিং খরচ বাঁচায়, ৮০% বাগ ২০% কোডে। হোয়াইট বক্সে কোড টেস্ট (postJob()), ব্ল্যাক বক্সে ফাংশনালিটি (জব ফর্ম)। ইন্টিগ্রেশন টেস্টে লগইন-পেমেন্ট চেক, ইনক্রিমেন্টালে বাগ আগে ধরা যায়।