Here's a simplified breakdown of each slide to help you remember key points for your Software Project Management paper:

1. Last Lecture Summary

- IEEE 1540:2001 outlines risk management processes in software life cycle.
- Focuses on managing risks during software acquisition, development, and maintenance.

2. Words of Wisdom

- Take full responsibility for your actions and choices (Al-Qur'an 74:38).

3. Planning - Estimating

- Estimating involves gathering information like requirements, work breakdown structure (WBS), size, cost, effort, and history.
- Estimations should be revised and negotiated to refine cost and schedule.

4. Estimation

- Estimation is predicting the required effort, cost, and time for a project.

5. Duration vs. Effort vs. Productive Time

- **Duration** is the total time taken (in business days).
- **Effort** is the labor required to complete a task.
- **Productive Time** is the actual time spent working on the project, considering interruptions.

6. Elapsed time vs. work time

- Elapsed time is total time from start to finish.
- Work time is the actual time spent on tasks.

7. Why Estimation?

- Estimation helps determine if project features are feasible in terms of resources, cost, and schedule.
- Affects project monitoring and control.
- A significant portion of projects fail due to poor estimation (e.g., cost overruns, delays).

8. Why Estimate?

- Estimates help assess if project plans are realistic.
- Helps determine if the required features match available resources, schedule, and cost.

9. When to Estimate?

- Estimates should be made at different stages: initial, detailed, and re-estimates throughout the project lifecycle.

10. Cone of Uncertainty

- Estimates are less accurate at the start and become more precise as the project progresses.

11. Types of Estimates

- **Top-down (macro)** estimates: high-level, based on analogy or consensus.
- **Bottom-up (micro)** estimates: detailed, based on individual task breakdown.

12. Problems in Estimation

- Challenges include software complexity, lack of historical data, inexperience, and poor requirements management.

13. Size Estimation

- Size estimation uses metrics like Lines of Code (LOC) or function points.
- Size estimation can change during the project and can be based on algorithms, previous projects, or expert judgment.

14. Function Points

- A way to estimate software size based on functionality (inputs, outputs, queries).
- Can be used in the design stages of a project.

15. Systems View (FP perspective)

- Function points are based on different elements: inputs, outputs, internal files, etc.

16. FP - Functionality Types

- Different functionality types include external user inputs/outputs, internal logical files, and external interface files.

17. FPA Estimates

- Function points are calculated by assessing functionality (inputs, outputs), complexity, and risk factors.

18. Components of FP

- **Data functions**: Internal files (user data), external files (data maintained by other systems).
- **Transactional functions**: Inputs, outputs, and inquiries that users interact with.

19. Components of FP

- **Data functions** include internal logical files (user data) and external interface files (data from other systems).
- **Transactional functions** include screens, reports, and user inputs/outputs.

20. Complexity Level for ILF and EIF

- The complexity of data elements (e.g., number of records) affects the function points calculation.

21. Complexity Level for EO and EQ

- External outputs and inquiries have complexity levels based on the number of data elements.

22. Complexity Level for EI

- External inputs complexity depends on the type and number of data elements.

23. Weighting Factors

- Weight factors assigned to data functions help calculate unadjusted function points (UFP).

24. UFP Formula

- Unadjusted function points are computed using a formula that factors in different complexity levels.

25. Environment Complexity Factor

- Factors like data communication, distributed functions, performance, and system design impact estimation.

26. Additional Environment Complexity Factors

- More factors include reliability, system design complexity, reusability, and online data entry.

27. Q&A

- This slide is for questions and answers; no content to summarize.

For each of the 28 slides, focus on the key points that summarize the main concepts. These bullet points should help guide your understanding and recall for your exam! Let me know if you need any further clarification on any specific slide.

Here's a simplified breakdown of each slide in **Lecture 17: Software Project Management**:

---

1. Complexity Level for ILF and EIF

- **ILF (Internal Logical File)** and **EIF (External Interface File)** complexity depends on:
    - Record elements (RETs) and data elements (DETs).
    - For example, **1-19 elements**: Low complexity, **20-50**: Average, and **51+**: High complexity.

2. Complexity Level for EO and EQ

- **EO (External Output)** and **EQ (External Inquiry)** complexity depends on:
    - File types and data elements.
    - Similar complexity categorization based on the number of data elements.

3. Complexity Level for EI

- **EI (External Input)** complexity depends on:

- The number of file types and data elements involved.
- A similar categorization applies as for EO and EQ (Low, Average, High).

4. Weighting Factors

- Weighting factors for different components (e.g., ILFs, EIFs, EIs, EOs):
  - **Low, Average, Complex** categories.
  - Examples: Internal logical files (ILF) weight factors are 7, 10, and 15 for low, average, and high complexity, respectively.

5. UFP Calculation

- Unadjusted Function Points (UFP) are calculated by using the formula:
  - **UFP = Σ Xi Wj**, where i represents the components and j represents the complexity level.

6. Environment Complexity Factors (ECF)

- Factors include data communication needs, performance, system environment (existing, heavily utilized), and more.
- These factors adjust the function point count based on system complexity.

7. ECF Influence Levels

- **Influence level** ranges from **0 (not present)** to **5 (strong influence)**.
  - The total influence (N) is calculated and used to determine the **CAF (Complexity Adjustment Factor)**:
    CAF = 0.65 + 0.01 * N.

8. Determine Effort and Time

- **Effort**:
  - **Effort = AFP / Productivity**, where productivity varies based on system familiarity and specification.
- **Time**:
  - **Time = AFP / Delivery Rate**, with a standard delivery rate formula: **Delivery Rate = 0.45 * AFP**.

9. Example - Employee-Job Database

- Example system tracks employees, their job assignments, and locations.
  - Supports features like adding, changing, deleting, and querying employee details.

10. Example - ILFs and EIFs

- **ILF Example**: Employee entity with name, SSN, type, and location.
- **EIF Example**: Location (external data maintained by another system).

11. Example - ILFs and EIFs

- **Job entity** includes job number, description, and salary.
- **Job Assignment entity** links employees to jobs, with additional data like performance ratings.

12. Example - ILFs and EIFs

- **Location** entity stored in another system (external).

13. Counting ILFs and EIFs

- Identify ILFs and EIFs:
    - Three ILFs (Employee, Job, Job Assignment).
    - One EIF (Location).

14. Counting ILFs/EIFs - Complexity

- Complexity is assessed based on the number of data elements (DETs) and record element types (RETs).

15. ILF and EIF Unadjusted FPs

- External and internal files and their corresponding function points are based on complexity levels.

16. Counting EIs - Raw Data

- Examples of external inputs (EIs): Employee maintenance, job maintenance, job assignment maintenance, and location reporting.

17. Counting EIs - Complexity

- External inputs' complexity varies based on data elements and referenced files.

18. External Input Unadjusted FPs

- Unadjusted function points for external inputs are calculated based on their complexity level (low, average, high).

19. Counting EOs - Raw Data

- Examples of external outputs (EOs): Employee report, job report, and job assignment report.

20. Counting EOs - Complexity

- External outputs' complexity is based on data elements and referenced files.

21. External Output Unadjusted FPs

- Unadjusted function points for external outputs are calculated based on their complexity.

22. Counting EQs

- Examples of external inquiries (EQs): Employee inquiry, job inquiry, job assignment inquiry, and location inquiry.

23. EQ Unadjusted FPs

- Unadjusted function points for external inquiries are calculated based on complexity.

24. Total Unadjusted Function Points

- Total unadjusted function points are calculated by considering all components (EIs, EOs, ILFs, EIFs, EQs).

25. Reading

- Suggested readings include key references like **"The Mythical Man-Month"** and **Software Engineering Project Management**.

26. Q&A

- The last slide for any questions.

This structure should help you recall the key concepts quickly and clearly. Let me know if you need further clarification on any slide!

Sure! Let's break it down with tables, similar to the format used in the slides, while also explaining each section.

---

1. Complexity Level for ILF and EIF

Record Elements (RETs)	**1-19**	**20-50**	**51+**
Data Elements (DETs)			
1-5	Low	Low	Average
6-10	Low	Average	High
11+	Average	High	High

- **ILF (Internal Logical File)** and **EIF (External Interface File)** complexity is based on:
  - **Record Elements (RETs)**: Number of elements in the record (e.g., fields).
  - **Data Elements (DETs)**: Number of data items the file holds (e.g., attributes).

---

2. Complexity Level for EO and EQ

File Type	**1-5 Data Elements (DETs)**	**6-19 DETs**	**20+ DETs**
External Output (EO)	Low	Average	High
External Inquiry (EQ)	Low	Average	High

- **EO (External Output)** and **EQ (External Inquiry)** complexity depends on:
  - **File Types**: The type of file (e.g., external inputs, outputs, internal files).
  - **Data Elements (DETs)**: The number of elements in the file.

---

3. Complexity Level for EI

File Type	1-4 Data Elements (DETs)	5-15 DETs	16+ DETs
External Input (EI)	Low	Average	High

- **EI (External Input)** complexity is determined by:
    - **File Types**: Types of files being referenced (internal, external).
    - **Data Elements (DETs)**: The number of elements in the file.
    - **File Referenced (FTRs)**: The number of files referenced in the input.

---

4. Weighting Factors for Various Components

Component Type	Low	Average	High
Internal Logical Files (ILFs)	7	10	15
External Interface Files (EIFs)	5	7	10
External Inputs (EIs)	3	4	6
External Outputs (EOs)	4	5	7
External Inquiries (EQs)	3	4	6

- These factors are used to calculate **Unadjusted Function Points (UFP)**. Each component (ILF, EIF, EI, EO, EQ) is assigned a weight based on its complexity.

---

5. UFP Calculation Formula

- **Unadjusted Function Points (UFP)** = Σ (Xi * Wj)
    - **Xi**: Number of each function type (e.g., ILF, EIF, etc.).
    - **Wj**: Weight for the function type based on complexity (low, average, high).

---

6. Environment Complexity Factor (ECF)

Factor	Not Present (0)	Insignificant (1)	Moderate (2)	Average (3)	Significant (4)	Strong Influence (5)
Data Communication Required?	0	1	2	3	4	5

Factor	Not Present (0)	Insignificant (1)	Moderate (2)	Average (3)	Significant (4)	Strong Influence (5)
Distributed Processing Functions?	0	1	2	3	4	5
Performance Critical?	0	1	2	3	4	5
Existing, Heavily Utilized Environment?	0	1	2	3	4	5

- These factors assess how the environment influences the overall complexity of the software. A higher score means a stronger influence.

---

7. Calculate CAF (Complexity Adjustment Factor)

- **CAF** = 0.65 + 0.01 * N
    - **N** = Sum of the Environment Complexity Factors (ECF).
    - After calculating **CAF**, multiply it by the **Unadjusted Function Points (UFP)** to get the **Adjusted Function Points (AFP)**.

---

8. Determine Effort and Time

- **Effort**:
    - **Effort = AFP / Productivity**.
    - **Productivity** depends on system familiarity and specification (e.g., familiar system with well-specified requirements might have higher productivity).
- **Time**:
    - **Time = AFP / Delivery Rate**.
    - **Delivery Rate** is calculated as **0.45 * AFP**.

Can You Set the Numbers?

- You **cannot** set the numbers arbitrarily. These values (like the weights and complexity factors) are based on established standards and best practices in software estimation. They provide a framework for estimating software projects and calculating function points in a consistent manner. Adjusting these numbers outside the given guidelines could lead to inaccurate estimations. However, the function points (like the number of ILFs, EIFs, etc.) depend on the specifics of your project and can be counted accordingly.

This structure should give you a clear and concise way to approach function points and their complexity in Software Project Management. Let me know if you'd like further clarifications on any section!

Let’s simplify and explain each slide in **Lecture 18: Software Project Management** for easier understanding.

---

1. Words of Wisdom

- Take full responsibility for your actions and decisions.
- Every person is accountable for what they do, according to the Qur'an (74:38).

---

2. Planning - Estimating

- **Planning and estimating** involves:
    - **Source Information**: The starting point for estimation includes documents like the **Statement of Work** and **Requirements**.
    - **Work Breakdown Structure (WBS)**: Breaking down the project into smaller tasks.
    - **Estimate Constraints**: Considering factors like size, effort, and cost.
    - **Revising and Negotiating**: Constantly revising estimates to ensure they are reasonable and achievable.

---

3. Object Points

- **Object Points**: A way to measure project size, focusing on different **objects**:
    - **Screens**: Number of user interface screens.
    - **Reports**: Number of reports the system produces.
    - **Modules**: The number of 3GL (third-generation language) modules required to supplement 4GL code.

---

4. Access and Classify Objects (Screens)

- Categorize **screens** based on the number of data tables involved and the number of views (client/server setup).
- For example, if you have fewer than 4 data tables and fewer than 3 views, it’s considered **simple**.

---

5. Access and Classify Objects (Reports)

- Categorize **reports** in a similar way to screens.
- Number of **sections** in a report and number of data tables used determine its complexity.

---

6. Assess the Complexity

- Complexity of screens, reports, and modules can be **simple**, **medium**, or **complex**.
    - For example, a screen with 1 element is **simple**, but one with 3 elements and 5 modules becomes **complex**.

---

7. Percentage of Reuse

- Estimate how much of the project can be reused from previous projects to **reduce object points**.
    - Formula: **NOP = (OPs) * (100 - % reuse) / 100**
    - NOP = New Object Points

---

8. Determine the Productivity Rate

- Productivity depends on the experience and capability of the developers.
    - **VL (Very Low)**, **L (Low)**, **N (Normal)**, **H (High)**, **VH (Very High)** levels of experience.
    - **Productivity** can be calculated based on the **Object Points** (NOP) and developer capabilities.

---

9. Use Case Points (UCP)

- **Use Case Points**: A method to estimate the size and complexity of software based on **use cases** (scenarios of how users interact with the system).
    - Use cases are categorized and counted to form an unadjusted point count.
    - Multiply this count by factors like **Technical Complexity (TCF)** and **Environmental Complexity (ECF)** to adjust it.

---

10. Do Use Cases Improve Estimates?

- **Use Cases** help in understanding the scope from a user's perspective.
- Size and complexity can be linked to use cases, but ultimately they rely on the details from the use case documents.

---

11. Steps to Generate Estimates

1. **Identify, classify, and weight actors** (users/systems interacting with the software).
2. **Identify, classify, and weight use cases** (actions the software must support).
3. Calculate **Unadjusted Use Case Points (UUCP)**.
4. Adjust the points by **Technical Complexity** and **Environmental Complexity**.
5. Convert points into **time and effort estimates**.

---

12. Identify, Classify and Weight Actors

- **Actors** represent users or systems interacting with your software.
    - **Simple**: A system with a defined API.
    - **Average**: A system using a protocol like TCP/IP.
    - **Complex**: A person using the software through an interface.

---

13. Identify, Classify and Weight Use Cases

- **Use Cases** represent the different actions your software needs to support:
    - **Simple**: Quick tasks with few steps or classes.
    - **Average**: Tasks with more steps or complexity.
    - **Complex**: Tasks with many steps or involving many classes.

---

14. Unadjusted Use Case Points (UUCP)

- **UUCP** = Sum of **Use Case Weights** (UUCW) and **Actor Weights** (UAW).
- This gives an initial estimate of the project size before adjusting for complexity.

---

15. Technical Complexity Factors

- Technical factors influence how difficult it is to develop the software. Examples include:
    - **Distributed systems**, **Performance needs**, **Security features**, and more.
- Each factor is assigned a weight, and their combined complexity helps calculate the **Technical Complexity Factor (TCF)**.

---

16. Technical Complexity Factors (Continued)

- Factors such as **reusability**, **portability**, **installation ease**, and **security** are rated based on their influence on the project.
    - Calculate **TCF** = 0.6 + (0.01 * sum of all technical factors).

---

17. Environment Factors

- **Environment Factors** involve elements like familiarity with tools (e.g., UML), **OO experience**, and **application stability**.
    - Each factor is assigned a weight and an influence level (0 to 5).
    - **ECF** = 1.4 + (-0.03 * sum of all environment factors).

---

18. Environment Factors (Continued)

- Additional factors include **motivation**, **leadership**, and **part-time workers**.
    - These factors affect the project environment, which influences overall complexity.

---

19. Adjusted UCP and Effort

- **UCP = UUCP * TCF * ECF**.
- Once you have the **adjusted use case points (UCP)**, you can calculate the **effort and time** needed for the project.

---

20. Correlation Between UCP and FP

- There's a mathematical relationship between **Use Case Points (UCP)** and **Function Points (FP)**.
    - **Formula**: **y = 20.85x** (where **x** is UCP and **y** is FP).
    - The graph shows a correlation where increasing UCP also increases FP.

21-27. Delphi and Wideband Delphi Methods

- **Delphi** is a group estimation method where experts independently give estimates, and their opinions are refined through multiple rounds of discussion.
- **Wideband Delphi** improves this by involving more interaction among experts.
    - Experts discuss issues, revise estimates based on group feedback, and refine until consensus is reached.

---

28. Advantages of WBD

- **Advantages**:
    - Combines knowledge from multiple experts.

    - Helps improve understanding and consensus on the software project estimate.

---

29. Drawbacks of WBD

- **Drawbacks**:
    - Group consensus can lead to incorrect estimates.
    - May lead to a false sense of confidence in estimates.

---

30. Q&A

- This slide is for questions and answers.

This breakdown simplifies the key concepts. If you'd like more detail on any specific topic or example, feel free to ask!

Here's a simplified explanation of **Lecture 19: Software Project Management**:

---

1. Words of Wisdom

- **Finish today's work today**: Don't procrastinate! Always aim to get better and faster at what you do.
- **Never save anything for tomorrow**: Tackle your tasks immediately (Tirmidhiy).

---

2. Effort Estimating Models

- **Effort Estimating Models**: These models predict how much effort will be required to complete a project, based on factors like the **size** and **complexity** of the software.
    - An **estimating model** uses inputs (like software size) and gives you an estimate of the effort needed.

---

3. A Very Simple Model

- A simple formula to estimate effort:
    - **Effort = Size × Productivity**
    - Example formulas:
        - **Staff Days = Lines of Code × Staff Days per LOC**
        - **Staff Days = Modules × Staff Days per Module**
        - **Staff Days = Function Points × Staff Days per Function Point**

---

4. A Graph of the Model

- The graph shows how effort increases as software **size** increases.
- The relationship is **linear** (straight line), but...

---

5. But We Know That...

- **Effort increases faster** as the project grows due to additional management overhead, coordination, and other complexities.

---

6. Perhaps Something Like This

- The graph now **curves upwards**, reflecting that effort grows more than linearly as the project size increases.

---

7. One Model That Produces Such a Curve

- A more realistic model is:
  Effort = a × Size^b
    - Where **a** and **b** are constants that depend on the type of software and the company.
    - This model better fits the observed data of many organizations.

---

8. How to Determine the Curve

- By collecting data from previous projects, you can calculate typical values for **a** and **b**, which reflect the **effort curve** for your company.

---

9. COCOMO

- **COCOMO (Constructive Cost Model)**: A popular model for estimating software project effort, developed by Barry Boehm in 1981.
    - It predicts **effort** and **duration** based on software **size** and other factors.

---

10. Original COCOMO

- COCOMO has **three models**:
    1. **Basic**: Used early in the project.
    2. **Intermediate**: After requirements are defined.
    3. **Advanced/Detailed**: After design is complete.

- All models use the equation:
  Effort = a × Size^b
    - **Size** is in **Thousands of Lines of Code (KLOC)**.
    - **Effort** is measured in **Staff Months**.

---

11. Three Levels of Difficulty

- **Organic**: Simple projects with familiar tools and a small team (typically 2-50 KLOC).
- **Semi-Detached**: Average projects with some new challenges (50-300 KLOC).
- **Embedded**: Complex projects with strict constraints (size can vary widely).

---

12. COCOMO Modes

- **COCOMO Modes** have **different constants for each difficulty level** (a and b values).
    - **Organic**: 2.4 for a, 1.05 for b
    - **Semi-Detached**: 3.0 for a, 1.12 for b
    - **Embedded**: 3.6 for a, 1.20 for b

---

13. Basic COCOMO Equation in Graph Form

- The graph shows how effort increases for **organic**, **semi-detached**, and **embedded** projects.

---

14. Intermediate COCOMO

- COCOMO includes **cost drivers** (factors that affect the project's cost), such as:
    - Reliability, database size, product complexity, etc.
- These factors can adjust the effort prediction.

15-18. Cost Drivers

- **Cost drivers** include things like:
    - **The nature of the job**: For example, the required reliability for real-time systems.
    - **Tools and practices**: Use of modern programming tools or methods like **OO (Object-Oriented)**.
    - **People**: Experience and capability of analysts, programmers, and other staff.
    - **External factors**: For example, unstable systems or security requirements.

---

19. Intermediate COCOMO Adjustment Factors

- **Adjustment factors** are applied to the basic equation to modify the estimated effort based on the cost drivers.

---

20. Intermediate COCOMO Equation

- The revised equation is:
  Effort = EAF × a × Size^b
    - **EAF** is the **Effort Adjustment Factor**, calculated by multiplying the adjustment factors for all cost drivers.

---

21. Intermediate COCOMO - Revised Values of "a"

- The **"a" value** changes based on the **effort adjustment factor (EAF)**, but **"b" remains constant** across the different levels of difficulty.

---

22. Cost Adjustment Factors

- These factors adjust the cost based on the values assigned to each cost driver (e.g., reliability, experience, tools used).

---

23. Detailed COCOMO

- **Detailed COCOMO** is the most complex version:
    - Decomposes the project into components and calculates costs for each one individually.
    - Provides more specific formulas to calculate cost adjustments for each component.

---

24. Schedule Estimation

- **Schedule estimation**: COCOMO also predicts the **development time** (TDEV) using the formula:
  TDEV = c × (Size^d)
    - Where **c** and **d** depend on the project's mode (organic, semi-detached, or embedded).

---

25. Issues with COCOMO (and Other Estimating Models)

- **Challenges**:

- Lots of judgment is required to choose the right values for cost drivers and determine which mode fits the project.
- The data might not perfectly match the model's assumptions.
- Some models, like COCOMO, assume the **waterfall model** (which doesn't always fit modern agile projects).

---

26. Each Model Includes Only Selected Tasks of Software Development

- For example, COCOMO focuses on coding and testing but **does not include**:
  - **System-level analysis and design**, **requirements analysis**, or **system-level integration and testing**.

---

27. Q&A

- This slide is for any questions you have.

This lecture covers **effort estimation models** like **COCOMO**, which are used to predict the **effort** and **time** needed for software projects based on **size** and other influencing factors (called **cost drivers**). It also explains the different **COCOMO models** for varying levels of complexity and how they help estimate the resources required for a project.

Let me know if you need more clarification on any part of the lecture!

Here's a simplified explanation of **Lecture 20: Software Project Management**:

---

1. Words of Wisdom

- **Focus on what you can control**: Instead of stressing over things you can't control, focus on tasks you can handle.
- **Repent and make amends**: Those who wrong others, but later seek forgiveness and make things right, are forgiven by Allah (Qur'an 16:119).

---

2. Issues with COCOMO

- **Judgment in Models**: A lot of judgment is needed to decide how to apply cost adjustment factors and which model to use.
- **Data Mismatch**: Your company's data may not perfectly match the data the model was built with.
- **Assumption of Waterfall Model**: Many models, like COCOMO, assume a **waterfall process** (e.g., 30% design, 30% coding, 40% integration and testing), which may not match your project's process.

---

3. Each Model Includes Only Selected Tasks of Software Development

- **COCOMO doesn't include**:
    - System-level analysis and design.
    - Software requirements analysis.
    - System-level integration and testing.

---

4. Software Practice

- **End-User Programming**: Users developing applications themselves using tools like spreadsheets or query systems.
- **Application Generators & Composition Aids**: Tools like **Microsoft Access** help users generate applications easily.
- **System Integration**: Involves integrating different software systems to work together, such as in large-scale systems like **manufacturing support systems**.

---

5. COCOMO & Cone of Uncertainty

- The **Cone of Uncertainty** shows how project cost and schedule become clearer as the project progresses. Initially, estimates are **more uncertain** (broad), and as more details are known, estimates get more **accurate**.

---

6. COCOMO II Models

- **Application Composition**: Involves prototyping and risk management to explore high-risk areas.
- **Early Design**: Used early in the project when not much is known.
- **Post-Architecture**: Used after the architecture is defined, focusing on actual development and maintenance.

---

7. Application Composition & Reuse

- **eKSLOC** (equivalent KLOC): The number of lines of new code to be written, considering reusable code that needs modification.
- The formula for this adjusts for the amount of **reuse** and modification.

---

8. Application Composition & Reuse (Continued)

- **AAM (Adaption Adjustment Modifier)**: Calculates the difficulty of reusing code based on factors like **software understanding** and **programmer unfamiliarity**.

---

9. Early Design

- The formula for estimating effort in **Early Design** is:
  Effort = A × Size^B
    - **Size** is measured in **KDSI (Kilo Delivered Source Instructions)** or **KSLOC (Kilo Source Lines of Code)**.
    - **Scaling Factors** (like **precedentness**, **development flexibility**, etc.) adjust the model to reflect project-specific factors.

---

10. Scaling Factors

- **Precedentness**: How familiar your team is with the project and related systems.
- **Development Flexibility**: How rigid the project's requirements are.
- **Architecture/Risk Resolution**: The importance of resolving technical risks and the level of architecture support.
- **Team Cohesion**: How well the project team works together.

---

11. Scaling Factors (Continued)

- **Process Maturity**: How developed your project's processes are (e.g., using **CMM** levels).

12-13. Scaling Factor Table

- The table shows the values for different **Scaling Factors (SF)** based on how each factor is rated (Very Low, Low, Normal, High, etc.).

---

14. Early Design - Adjusted Effort

- **Adjusted Effort** is calculated by multiplying the **nominal effort** by a set of **effort multipliers (EM)** that are based on project-specific drivers (like **product reliability**, **personnel capability**, etc.).

---

15. Early Design Effort Multipliers

- **RCPX** (Product Reliability), **RUSE** (Reusability), and others are factors that influence the effort required for the project.
    - For example, **RCPX** can range from **0.49 (very low)** to **2.72 (extra high)**.

---

16. COCOMO II Post-Architecture Level

- **Post-Architecture** efforts are similar to **Early Design**, but with additional factors like **Documentation (DOCU)** and **Platform difficulty (PDIF)** added.

---

17. Post-Architecture Effort Drivers

- New drivers in **Post-Architecture** include:
    - **DOCU**: Documentation needs.
    - **RUSE**: Reusability.
    - **PCON**: Personnel continuity.
    - **SITE**: Multi-site development.

---

18. Post-Architecture Effort Drivers (Continued)

- **PVOL (Platform Volatility)**: The stability of the platform used for the project.
- **PEXP (Programmer Experience)**: The level of experience the programmers have in the relevant domain.

---

19. Cost Drivers

- **Cost Drivers** include factors like **required software reliability**, **product complexity**, and **execution time constraints** that affect the effort and time required.

---

20. Effort Multipliers COCOMO II

- **Effort multipliers** adjust the basic effort equation based on specific cost drivers:
    - For example, **RELY (software reliability)** and **DATA (data size)** can increase the effort required for a project.

---

21. Schedule Estimates

- **Schedule Estimation**: Involves adjusting the effort and calculating the development time (TDEV) for the project, considering the **effort** and **schedule multipliers**.

---

22. Project Phases

- Phases included in **COCOMO II**:
    - Product Design, Implementation, Testing, Project Management, Configuration Management, and Quality Assurance.
- Phases not included:
    - Requirements definition and project-level activities.

---

23. Effort and Schedule Distribution

- The distribution of effort across project phases:
    - **Planning & Requirements**: 2-15% of effort.
    - **Product Design**: 17% effort.
    - **Programming**: 52-64% of effort.
    - **Testing**: 19-31% effort.

---

24. Spiral-Based/RUP Phases

- The **Spiral Model** and **RUP (Rational Unified Process)** have different phase distributions:
    - **Construction** phase takes up 65% of the effort.
    - **Transition** phase takes 10%.

---

25. Q&A

- This slide is for any questions and answers.

This lecture covers how **COCOMO** and other models estimate the **effort** and **schedule** of software projects. It discusses the importance of **scaling factors** and **cost drivers** in adjusting these estimates based on project complexity. The COCOMO model is updated to **COCOMO II**, which includes more modern factors like **platform volatility** and **personnel continuity**.

Let me know if you need any further clarification!

Here's a simplified explanation of **Lecture 21: Software Project Management**:

---

1. Words of Wisdom

- **Know your value**: You were created by Allah as the best creation. Remember your importance and worth, just like Adam, the first human, was honored by Allah (Qur'an 16:119).

---

2. Scheduling Duration Estimates

- **Scheduling** is the process of turning a project action plan into a detailed timeline.
    - **Work Breakdown Structure (WBS)** helps break down the project into smaller tasks.
    - **Network schedule** helps organize tasks in a sequence (which task comes before or after another).

---

3. Scheduling

- **WBS**: Organizes work tasks hierarchically, showing which tasks are parts of others.
- **Schedule Network**: A sequence of tasks showing which ones need to be completed first.
- A **schedule** helps ensure timely completion by specifying:
    - Time allowed for each task.
    - Resources (people, skills) needed for each task.
    - Risks and important issues.

---

4. Importance of Project Schedules

- **Why is scheduling important?**
    - **Project delays** are a common challenge.
    - A 1995 report found that **average time overrun** was 222%, improving to 63% by 2001.
    - Time has **no flexibility**—it passes no matter what!
    - **Schedule issues** often lead to conflicts in the later stages of a project.

---

5. Conflict Intensity Over the Life of a Project

- **Conflict** tends to increase as the project progresses, especially concerning **schedules**, priorities, and technical opinions.

---

6. Scheduling Steps

- Steps to create a schedule:
    1. **Identify tasks** and estimate resources needed.
    2. **Find dependencies** between tasks (which tasks must come before or after others).
    3. **Create a network** of tasks and determine the timeline.
    4. **Develop resource loading profiles**.
    5. **Iterate** to balance tasks, timeline, and resources.

---

7. PMI's Project Time Management Processes

- **Time management processes** include:

- **Activity definition**: Break down the work.
- **Activity sequencing**: Determine task order.
- **Duration estimating**: Estimate how long tasks will take.
- **Schedule development**: Build the overall timeline.
- **Schedule control**: Monitor and adjust the schedule as needed.

---

8. Where Do Schedules Come From?

- **Schedules are based on**:
  - **Project charter**: Contains start and end dates, budget.
  - **Scope statement and WBS**: Define what needs to be done.
  - **Activity definition**: Break down the WBS into specific tasks and deliverables.

---

9. Activity Sequencing

- **Activity sequencing** involves determining the **dependencies** between tasks:
  - **Mandatory**: Inherent dependencies (e.g., Task B must follow Task A).
  - **Discretionary**: Defined by the project team.
  - **External**: Dependence on outside factors.

---

10. Project Network Diagrams

- A **network diagram** visually shows the sequence of tasks and their relationships.

---

11. Arrow Diagramming Method (ADM)

- A method where:
  - Activities are shown as arrows.
  - Nodes or circles represent the start and end points.
  - Shows **finish-to-start** dependencies (Task B can't start until Task A finishes).

---

12. Process for Creating AOA Diagrams

- Steps to create **AOA (Activity-on-Arrow)** diagrams:
  1. **Start at node 1** and draw arrows for activities that start there.
  2. **Continue drawing** the diagram, working from left to right.
  3. **Include all activities** with dependencies.

---

13. Precedence Diagramming Method (PDM)

- A method where:
    - Activities are shown as **boxes**.
    - Arrows show dependencies between activities.
    - More commonly used than ADM and supports various types of dependencies.

---

14. Task Dependency Types

- Types of dependencies:
    1. **Finish-to-Start**: Task B can't start until Task A finishes.
    2. **Start-to-Start**: Task B can't start until Task A starts.
    3. **Finish-to-Finish**: Task B can't finish until Task A finishes.
    4. **Start-to-Finish**: Task B can't finish until Task A starts.

---

15. Sample Precedence Diagramming Method (PDM) Network Diagram

- A diagram showing tasks and dependencies using the **Precedence Diagramming Method**.

---

16. Activity Duration Estimating

- **Estimating duration** involves calculating how much time each task will take.
    - **Duration** includes both the actual working time and the **elapsed time**.

---

17. Schedule Development

- **Schedule development** involves creating a realistic project schedule using:
    - **Gantt charts**, **PERT analysis**, **critical path analysis**, and **critical chain scheduling**.

---

18. Gantt Charts

- **Gantt charts** show tasks along a timeline.
    - Activities are listed with start and finish dates.
    - A **milestone** (black diamond) represents key events with zero duration.

---

19. Critical Path Method (CPM)

- **CPM** is used to determine the **longest path** through the project's activities, which dictates the **earliest possible completion time**.
    - The **critical path** has the least amount of slack, meaning any delay on the critical path will delay the whole project.

---

20. Finding the Critical Path

- To find the **critical path**:
    1. Develop a **network diagram**.
    2. Add the durations for all activities along each path.
    3. The **longest path** is the critical path.

---

21. Simple Example of Determining the Critical Path

- A simple project example where you calculate the paths, durations, and identify the critical path.

---

22. More on the Critical Path

- If activities on the **critical path** are delayed, the whole project will be delayed unless corrective action is taken.
    - There can be **multiple critical paths** if they have the same total duration.

---

23. Using Critical Path Analysis to Make Schedule Trade-offs

- **Slack/Float**: The time an activity can be delayed without affecting the project schedule.
    - **Free slack**: Time delay without affecting the next task.
    - **Total slack**: Time delay without affecting the project finish date.

---

24. Determining the Critical Path

- This slide shows the **critical path** determination process in more detail.

---

25. Q&A

- This slide is for any questions and answers.

Summary:

This lecture covers **scheduling** techniques used in project management, focusing on creating and managing **project schedules**. The **Critical Path Method (CPM)** helps identify the longest sequence of tasks and determine the project's **earliest completion time**. Tools like **Gantt charts** and **Precedence Diagramming** are used to visualize and track the project timeline.

Let me know if you'd like any further clarification or help with the next lecture!

Here's a simplified explanation of **Lecture 22: Software Project Management**:

---

1. Critical Path Method (CPM)

- **CPM** is a method used to predict the total **duration** of a project.
- The **critical path** is the sequence of tasks that determines the **earliest possible completion time** for the project.
- The **critical path** is the **longest path** in the project network, and it has the least amount of **slack** or **float**.

---

2. CPM Example

- A **network diagram** is used to visualize activities and their durations.
    - **Earliest Start (ES)** and **Earliest Finish (EF)** are calculated for each task.
    - The **critical path** is identified by finding the longest path with the least slack.

---

3. Finding the Critical Path

- To find the **critical path**:
    1. Develop a **network diagram** with all tasks.
    2. Add the durations for each path.
    3. The **longest path** with no slack is the critical path.

---

4. Simple Example of Determining the Critical Path

- In this example, each task has a time duration (in days). You calculate all paths and find the critical path by identifying the longest one.

---

5. More on the Critical Path

- If any task on the **critical path** is delayed, the entire project will be delayed unless corrective action is taken.
- There can be **multiple critical paths** if they have the same total duration.
- The critical path can **change** as the project progresses.

---

6. Using Critical Path Analysis to Make Schedule Trade-offs

- **Slack/Float**: The amount of time a task can be delayed without affecting the project's overall schedule.
    - **Free slack**: Time a task can be delayed without delaying the start of subsequent tasks.
    - **Total slack**: Time a task can be delayed without delaying the overall project completion.

7-12. Free and Total Float or Slack

- These slides explain how to calculate and use **slack/float** for managing delays and adjusting schedules.

---

13. Techniques for Shortening a Project Schedule

- Two main techniques:
    1. **Crashing**: Adding resources to critical tasks to speed up completion.
    2. **Fast tracking**: Doing tasks in parallel instead of sequentially.

---

14. Importance of Updating Critical Path Data

- It's important to **update the critical path** as actual start and finish dates are recorded.
- If the project will be delayed, it's crucial to **negotiate with the project sponsor**.

---

15. CPM and Float

- **Float** is the **time difference** between the available time for a task and the time needed to complete it.
    - **Forward pass**: Calculates the **earliest start and finish** times for each task.
    - **Backward pass**: Calculates the **latest start and finish** times for each task.

16-18. Float – Forward Pass & Backward Pass

- The **forward pass** calculates the **earliest times** tasks can start and finish, while the **backward pass** calculates the **latest times** they can start and finish without delaying the project.

---

19. PERT (Program Evaluation and Review Technique)

- **PERT** is a technique used to estimate project duration when there's uncertainty in task durations.
    - It uses a **beta distribution** (optimistic, most likely, and pessimistic time estimates).
    - The **expected duration** is calculated using a weighted average formula.

---

20. PERT Formula and Example

- The **PERT formula**:
  Expected duration = (Optimistic + 4 × Most Likely + Pessimistic) / 6
    - Example:
      (8 + 4 × 10 + 24) / 6 = 12 days

---

21. Critical Chain Scheduling

- **Critical Chain Scheduling** takes **resource constraints** into account when scheduling tasks.
    - It includes **buffers** to protect the project's completion date from delays.
    - This method assumes that **resources do not multitask**, as multitasking delays task completion.

---

22. Multitasking Example

- An example showing how **multitasking** can delay task completion and increase total project duration.

---

23. Buffers and Critical Chain

- **Buffers** are extra time added to tasks to account for uncertainty:
    - **Project buffer**: Time added before the project's final due date.
    - **Feeding buffers**: Time added before tasks on the critical path to ensure they don't delay the project.

---

24. Steps in Critical Chain

- Steps include:
    1. Develop a **late start schedule**.
    2. **Eliminate resource conflicts**.
    3. Identify the **critical chain** (longest sequence of dependent tasks).
    4. Add **feeding buffers** and **project buffer**.

---

25. Example

- An example showing how to schedule tasks and resolve resource conflicts.

---

26. Resource Conflicts

- **Resource conflicts** occur when multiple tasks require the same resource at the same time. These conflicts need to be resolved to keep the project on track.

---

27. Resolve Contentions

- **Contentions** are resolved by adding artificial dependencies between tasks, making sure resources are used efficiently.

---

28. Determine Critical Chain

- The **critical chain** is identified after resolving resource conflicts and adjusting the schedule.

---

29. Insert Feeding Buffers

- **Feeding buffers** are added before tasks on the critical chain to protect the project timeline.

---

30. Insert Project Buffer

- The **project buffer** is added at the end of the critical path to account for any unforeseen delays.

---

31. Q&A

- This slide is for any questions and answers.

Summary:

This lecture focuses on **Critical Path Method (CPM)** and **Critical Chain Scheduling**:

- **CPM** helps identify the **longest sequence of tasks** (critical path) that determines the project's **earliest completion time**.
- **Critical Chain Scheduling** factors in **resource constraints** and uses **buffers** to protect the schedule.
- Techniques like **crashing** and **fast tracking** can shorten project timelines.

Let me know if you need any further clarifications!

Here's a simplified explanation of **Lecture 23: Software Project Management**:

---

1. Normalizing Schedule

- **Normalize schedule**: Check if resources are available as planned.
    - **Reschedule non-critical tasks** (tasks that don't affect the overall project deadline):
        - **Late scheduling**: Schedule them as late as possible.
        - **Early scheduling**: Schedule them as early as possible.
        - **Subset scheduling**: Complete some tasks first, and finish the rest as milestones are met.
- **Assign resources**: Make sure you know the availability of the best people and allocate resources efficiently to avoid gaps.

---

2. Loading Up and Leveling Out

- **Resource loading**: The amount of work each team member or resource is assigned.
- **Resource leveling**: If a team member is overloaded, redistribute work from over-booked resources to others with lighter schedules.
    - Key questions:
        1. How many hours is each resource available each day?
        2. Is a resource assigned to multiple projects?
        3. Have you accounted for interruptions and overhead?

---

3. Excessive Resource Loading

- If resources are overloaded:
    - **Change the scope** or **add more resources**.
    - **Extend task time** or split tasks.
    - **Move tasks to a later time** when resources are free.
    - Consider **outsourcing** or **reprioritizing** tasks.
    - You may need to **negotiate for more time** and budget.

---

4. Controlling Changes to the Project Schedule

- **Input to schedule control**:

    - Project schedule, performance reports, change requests, and schedule management plans.
- Ensure that:
    - Schedules are reviewed for **realistic accuracy**.
    - Allow for **contingencies** (unexpected events).
    - Be **honest** in meetings with stakeholders about schedule issues.

---

5. Working with People Issues

- Strong leadership is key to project success more than tools like PERT charts.
    - Use **empowerment**, **incentives**, **discipline**, and **negotiation** to manage the team.

---

6. Using Software to Assist in Time Management

- **Project management software** helps with scheduling and communication.
- Software tools can **analyze trade-offs** and provide insights for **better time management**.

---

7. Words of Caution on Using Project Management Software

- Many people misuse project management software because they don't understand key concepts or haven't received proper training.
    - Make sure to input **dependencies** and **actual schedule information** to track progress.

---

8. Scheduling - Summary

- Key steps:
    1. **Estimate task durations** and review with your team.
    2. **Determine the critical path** and task float.
    3. **Adjust resource assignments** to optimize the schedule.
    4. **Review** with the team and get final approval.

---

9. The 4 P's

- **People**: The most important element in any project.
- **Product**: The software being developed.
- **Process**: The steps, tasks, and activities used to develop the product.

- **Project**: All the work needed to make the product a reality.

---

10. Human Resources

- The most important rule in management is to ensure you have **good people** who can perform well in a positive environment.
    - **Great design** comes from **great designers**, and good methodology **empowers creativity**.

---

11. Human Resources

- **People are the most important asset** in an organization.
    - Project resources include:
        - **People** (staff, engineers).
        - **Reusable software components**.
        - **Hardware/software tools**.

---

12. Resourcing

- Key steps for resourcing a project:
    - **Acquire the right staff** through recruitment or training.
    - **Identify key roles and skills** needed.
    - **Assign roles** and **develop a staffing plan**.
    - Set up **reporting relationships** and **lines of communication** within the team.

---

13. Project Roles

- Key roles on a project include:
    - **Programmers**, **QA engineers**, **DBAs**, **Network engineers**, **Business analysts**, and **Project manager**.
    - Special roles like **security specialists** and **trainers** may also be needed.

---

14. Management & Technical Roles

- Small projects (5-6 people): One person may handle both **management** and **technical leadership**.
- Medium projects (10-20 people): Separate management and technical leadership roles.

- Large projects (>25 people): More specialized roles and supporting staff for both management and technical leadership.

---

15. Staffing Management Plan

- A **staffing management plan** is part of the **Software Development Plan** and includes:
    - **Roles needed**, timing (start/end), and resource assignments.
    - **Project directory** with contact information.

---

16. Staffing Profile

- Project teams **change size** throughout the project as needed, based on tasks.

---

17. Staffing the Project

- **Hire effectively**: Recruit the best people, even if they cost more.
    - **Role matching**: Match people's skills to the tasks they will perform.
    - **Career development**: Help people progress in their careers.
    - **Teamwork**: Choose team members who will work well together.
    - **Misfit elimination**: Quickly replace problem team members.

---

18. Q&A

- Time for questions and clarifications.

Summary:

This lecture covers **resource management** and the **importance of staffing** in a project. It emphasizes the **critical path** in scheduling, **resource loading**, and **leveling**, and the **role of good leadership**. Effective use of **project management software** and careful attention to **people issues** are essential for successful project completion.

Let me know if you'd like more details or help with another lecture!

Here’s a simplified explanation of **Lecture 29: Software Project Management**:

---

1. Risk Mitigation Plans

- **Risk Mitigation Plans** address risks that can be **avoided soon**. Examples include:
    - **Training** to improve developer skills.
    - **Using a faster processor** to improve performance.

    - **Automating configuration management** to reduce risks.
    - **Changing testing procedures** to avoid issues.
- The goal is to address **potential problems early** to avoid future issues.

---

2. Risk Mitigation Plan Format

A **Risk Mitigation Plan** includes:

1. **Name and ID** of the risk.
2. **Nature** of the risk.
3. **Actions to take** to mitigate the risk.
4. **Responsible party** (who will take action).
5. **Resources** to be applied.
6. **Progress milestones** (checkpoints to track progress).
7. **Completion criteria** (how to know if the risk is mitigated).
8. **Planned completion date**.

- The plan should be **reviewed periodically** for progress and revised as needed.

---

3. Contingency Plans

- **Contingency plans** address risks that may become problems in the future but don't need immediate action.
    - Example: Concern about a **key staff member** leaving the project, even if the project is on schedule right now.
- A contingency plan includes:
    - **Risk description**, **tracking method**, **problem trigger**, and a **contingent-action plan**.

---

4. Format of a Contingency Plan

A contingency plan includes:

- **Description** of the risk.
- **Tracking method** for the risk.
- **Trigger** event when the plan will be activated.
- **Contingent actions** to address the risk if it occurs.

---

5. Crisis Management

- Crises can happen due to:
    1. **Lack of attention** to potential problems.
    2. **Unmitigated risks** that were foreseen but not addressed.
    3. **Unanticipated situations**.
    4. **Failed contingency plans**.
- Projects would avoid crises if **risk management** were completely effective.

---

6. How to Respond to Crises

- **Acknowledge the crisis** and inform all relevant parties.
- **Assign responsibilities** and provide all necessary resources.
- **Review status daily** (possibly twice daily).
- **Work in "burn-out mode"**: Around-the-clock effort until the crisis is resolved.
- Set a **"drop-dead date"**: The date when the crisis is declared unsolvable.

---

7. Continuous Risk Management

- Risk management isn't just for the beginning of the project but should be **ongoing** throughout the project.
    - New risks may arise, some risks might never materialize, and some may evolve into new issues.

---

8. Tracking of Top-n Risk Factors

- In each status review, generate a **"top-n" list** of risk factors, including:
    - Risk factors, action items, and contingency plans.
    - This list helps prioritize and track risk management efforts.

---

9. Status Reviews

- **Status reviews** track:
    - **New risk factors**.
    - **Retired or resurfaced risk factors**.
    - The **status** of ongoing action items and contingency plans.

---

10. Levels of Risk Management

- **Crisis management**: Everything is broken.
- **Fix on failure**: Something broke, fix it!
- **Risk mitigation**: What will we do when something breaks?
- **Prevention**: How to keep things from breaking?
- **Eliminate root causes**: Understand why something could break and address it.

---

11. Reactive Approach

- In a **reactive approach**, the team only addresses risks **after they happen**:
    - **Mitigation**: Plan for more resources if a risk arises.
    - **Fix on failure**: Apply resources once the risk happens.
    - **Crisis management**: Failure happens and the project is in jeopardy.

---

12. Proactive Approach

- In a **proactive approach**, risks are identified **before** they occur:
    - **Formal risk analysis** is done, and the root causes of risks are addressed.
    - Use **quality management concepts** and **software quality assurance** to prevent risks from materializing.

---

13. IEEE 1540:2001 Risk Management Process Model

- This model provides a formal process for managing risk in software projects.

---

14. Technical and Management Processes

- **Risk Management (RM)** requires information on:
    - Risk areas of concern.
    - Risk acceptability thresholds.
    - Decision-making processes regarding risks.

---

15. Plan and Implement Risk Management

- **Establish RM policies** to guide decision-making.
- Define the **RM process**, **responsibilities**, and **resources**.
- Establish **risk communication** channels and **evaluation methods**.

---

16. Manage the Project Risk Profile

- Maintain a **risk profile** that gives a historical and current view of risks.
    - Define **risk thresholds** and communicate the **risk status** to stakeholders.

---

17. Perform Risk Analysis

- Identify risks, estimate their likelihood, and prioritize them.
- Document **risk treatment** and **contingency plans**.

---

18. Perform Risk Treatment

- **Risk treatment** involves selecting actions to reduce the risk.
- If a **12207 Life Cycle Process** is used, risk treatment is handled by the **management process**.

---

19. Perform Risk Monitoring

- **Monitor risks** by reviewing and updating risk states.
- Assess the effectiveness of **risk treatments** and search for new risks.

---

20. Evaluate the Risk Management Process

- **Evaluate the RM process** to capture lessons learned and improve it for future projects.

---

21. Q&A

- Time for **questions and answers**.

Summary:

This lecture focuses on **risk management** in software projects:

- **Risk mitigation** plans address foreseeable risks that can be avoided, while **contingency plans** prepare for risks that might occur later.
- **Crisis management** is needed when things go wrong, but it's better to take a **proactive approach** by identifying risks early.
- Regular **status reviews**, **tracking of risks**, and **adjusting the risk management process** are critical to handling risks effectively.

Let me know if you'd like further explanations or need help with other lectures!

Here are the lectures I've simplified for you:

1. **Lecture 16** - Planning and Estimating
2. **Lecture 17** - Scheduling, Resource Loading, and Leveling
3. **Lecture 18** - Scheduling, Risk Mitigation, and Control
4. **Lecture 19** - Effort Estimating Models, COCOMO, and Scheduling Techniques
5. **Lecture 20** - Software Project Scheduling and the Critical Path Method (CPM)
6. **Lecture 21** - Software Project Scheduling, Resource Management, and Team Roles
7. **Lecture 22** - Critical Path Analysis, Risk Management, and Crises Management
8. **Lecture 23** - Resource Management, Team Building, and Project Roles
9. **Lecture 29** - Risk Mitigation, Contingency Planning, and Crisis Management

Let me know if you'd like me to simplify more lectures or need more details!