Summary of Lecture 6: Software Estimation

This lecture focuses on **Software Estimation**, covering two major models:

1. **COCOMO (Constructive Cost Model)** – Estimates effort based on Lines of Code (LOC).
2. **Function Point Analysis (FPA)** – Estimates effort based on software functionality rather than LOC.

It also discusses **challenges** in software estimation, including uncertainties, resource constraints, and evolving technologies.

COCOMO (Constructive Cost Model)

🔹 **What is COCOMO?**

- A **mathematical model** to estimate **effort, cost, and schedule** for software projects.
- Developed by **Barry Boehm in 1981**.
- Uses **LOC (Lines of Code)** as the primary size metric.
- Based on empirical data, making it **widely used and customizable**.

COCOMO Estimation Process

1️⃣ Basic COCOMO

- Provides **quick & rough estimates** using a simple formula:
  [E = a \times (KLOC)^b] where:
  - **E** = Effort (in **person-months**)
  - **KLOC** = Thousands of Lines of Code
  - **a, b** = Constants based on project type

Project Type	a	b
Organic (small, well-understood)	2.4	1.05
Semi-Detached (medium, moderately complex)	3.0	1.12
Embedded (large, complex, stringent requirements)	3.6	1.20

✅ **Example Calculation:**

- Estimated LOC = **200 KLOC**
- Project Mode = **Organic**

- Formula:
  [E = 2.4 \times (200)^{1.05}] **Effort = 625.6 person-months**

2️⃣ Intermediate COCOMO

- Adds **Effort Adjustment Factor (EAF)** based on **15 cost drivers** (e.g., complexity, reliability).
- Formula:
  [E = a \times (KLOC)^b \times EAF] ✅ **Example:**
- EAF = **1.012** (adjusted based on complexity, reliability, etc.).
- Final adjusted effort = **633 person-months**

3️⃣ COCOMO II (Updated Version)

- Introduced in **2000** to address modern software needs.
- Uses **Function Points** & **Object Points** (not just LOC).
- Supports **Agile & Component-Based Development**.
- Includes **4 sub-models** and **17 cost drivers**.

COCOMO Insights

📌 **Effort grows faster than project size** – A **200KLOC project takes more than 2x effort of a 100KLOC** project.
📌 **Larger projects allow parallel work**, making them **faster per developer**.
📌 **COCOMO works best for Waterfall models** but struggles with Agile (because Agile constantly changes requirements).

✅ **Advantages:**
✔ Transparent & easy to compute.
✔ Helps estimate **feasibility, cost, and required personnel**.

❌ **Disadvantages:**
✖ **Assumes LOC is the best measure of effort** (not true for all cases).
✖ **Not suited for Agile**, where LOC constantly changes.
✖ **Requires accurate estimation of LOC beforehand**.

Function Point Analysis (FPA)

🔹 **What is Function Point Analysis?**

- Measures **software size based on functionality**, rather than LOC.
- Developed by **Allan J. Albrecht (IBM) in 1979**.
- Used in **Waterfall, Iterative, and Incremental Development**.

- Standardized by **IFPUG (International Function Point Users Group)**.

✅ **Useful for:**

- Estimating **business applications** (e.g., banking, CRM).
- Comparing different **technology stacks** (since LOC varies by language).

How Function Points Work

1️⃣ Identify Software Components (Function Types)

Function Type	Definition
External Inputs (EI)	User inputs modifying internal data (e.g., form submission).
External Outputs (EO)	Reports or processed data outputs.
External Inquiries (EQ)	Querying information **without modifications** (e.g., search results).
Internal Logical Files (ILF)	Internal databases used by the system.
External Interface Files (EIF)	Data shared between external systems.

2️⃣ Assign Complexity & Weight

Function Type	Low	Medium	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

3️⃣ Compute Total Function Points (UFP - Unadjusted Function Points)

[UFP = \sum (\text{Function Count} \times \text{Weight})] Example:

- **Total UFP = 643**

4️⃣ Apply Value Adjustment Factor (VAF)

[VAF = (TDI \times 0.01) + 0.65] where **TDI (Total Degree of Influence)** is based on **14 system characteristics** (e.g., performance, data communication).

✅ Final Adjusted Function Points

[FP = UFP \times VAF]

Example:

- VAF = **1.04**, so
- **Adjusted FP = 643 × 1.04 = 669**

Advantages of Function Points Over LOC

✅ Works across **different programming languages**.
✅ Measures **business functionality**, not just code size.
✅ Supports **Agile**, since FP can be recalculated after each backlog refinement.

Estimation Challenges

🚧 **1. Experience & Data Availability**

- Requires **past project data** or expert judgment.

🚧 **2. Emergent Requirements**

- Agile projects **evolve over time**, making traditional estimation difficult.

🚧 **3. Resource Constraints**

- Developer **skills & motivation** affect productivity.
- Customers expect **faster delivery than possible**.

🚧 **4. Technology Changes**

- **New frameworks & libraries** impact estimation accuracy.
- **Cloud computing & AI-based tools** reduce LOC but increase complexity.

🚧 **5. Human Bias (e.g., Anchoring Bias)**

- **First estimate heavily influences subsequent estimates**.
- Agile techniques (like **Planning Poker**) **reduce bias** by forcing silent estimation.

🚧 **6. Misconceptions About Adding More People**

- Some customers believe **"more developers = faster work"**.
- But **complex tasks don’t scale linearly** (e.g., 9 women can't produce a baby in 1 month).

Final Takeaways

🔹 **COCOMO estimates effort based on LOC**, making it useful for **Waterfall projects**.
🔹 **Function Point Analysis (FPA) estimates based on business functionality**, making it **better for Agile & modern software**.
🔹 **Challenges in estimation come from changing requirements, team skills, and evolving technologies**.
🔹 **No estimation method is perfect—continuous refinement is required**.

Keywords

- **COCOMO (Constructive Cost Model)**
- **Basic COCOMO**
- **Intermediate COCOMO**
- **COCOMO II**
- **Effort Adjustment Factor (EAF)**
- **Function Point Analysis (FPA)**
- **Unadjusted Function Points (UFP)**
- **Value Adjustment Factor (VAF)**
- **Estimation Challenges**
- **Emergent Requirements**
- **Technology Upgrades**
- **Anchoring Bias**