Chemuturi, M. (2009) – Chapter 5: Approaches to Software Estimation

🧩 Overview

This chapter serves as a **taxonomy of estimation methodologies**, breaking down both traditional and modern approaches. Chemuturi stresses that no method fits all, and good estimators know how to **mix and adapt techniques based on project context**.

🔧 Estimation Techniques

- **Top-Down Estimation**
    - Starts from overall effort/time and distributes to components.
    - Suitable for early-phase or high-level planning.
    - Risk: Ignores complexity at the micro level.
- **Bottom-Up Estimation**
    - Builds estimates for small units (e.g., modules, functions), then aggregates.
    - More detailed, time-consuming.
    - Works best with defined requirements and known tasks.
- **Expert Judgment**
    - Leverages experience, often used in agile contexts.
    - Highly subjective; accuracy depends on expert's historical knowledge.
- **Heuristic Estimation**
    - Uses rule-of-thumb or fixed ratios (e.g., "10% of code time = testing time").
    - Useful for recurring patterns but **can oversimplify unique cases**.
- **Empirical Models**
    - Based on data and algorithms (e.g., **COCOMO**, **Function Point Analysis**).
    - Require historic data, calibration.
    - COCOMO includes size (in KLOC), project type (organic/semi-detached/embedded), and adjustment factors (cost drivers).
- **Analogy-Based Estimation**
    - Uses previous similar projects as a baseline.
    - Most effective when past data is accessible and project types are comparable.

🧠 Practical Insights

- Use **triangulation**—apply multiple estimation methods and cross-check.
- Understand the limitations of each method (e.g., empirical models can't help when requirements are unclear).
- Avoid over-engineering simple estimates and oversimplifying complex ones.

🎯 Takeaway

Estimation is a **toolkit, not a template**. Skilled estimators match the technique to the scenario—balancing accuracy, effort, and available information.