Software Effort Estimation

Lecture 13

Shahani Markus Weerawarana
Objectives

- To introduce the fundamentals of software costing and pricing
- To describe software productivity assessment
- To explain why different techniques should be used for software estimation
References

Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
- Effort Estimation in Practice
Fundamental Estimation Questions

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
What is Estimation?

- Dictionary definition of estimate:
  - A tentative evaluation or rough calculation.
  - A preliminary calculation of the cost of a project.
  - A judgment based upon one's impressions.

BUT

- When executives ask for an estimate, they are asking for a commitment or a plan to meet a target.
Target

- A **target** is a statement of a desirable business objective.
- **Examples:**
  - "We need to have Version 2.1 ready to demonstrate at a trade show in May."
  - "We need to have this release stabilized in time for the holiday sales cycle."
- Businesses have important reasons to establish targets independent of software estimates. But the fact that a target is desirable or even mandatory does not necessarily mean that it is achievable.
Commitment

- A commitment is a promise to deliver defined functionality at a specific level of quality by a certain date.

- A commitment can be the same as the estimate, or it can be more aggressive or more conservative than the estimate.
Estimates and Plans

- **Estimation** is an unbiased, analytical process.
- **Planning** is a biased, goal-seeking process.
- Estimates form the foundation for the plans – the plans don't have to be the same as estimates.
  - If the estimates are different from the targets, the project plans will need to recognize that gap and account for a certain level of risk.
Warning!

Distinguish between estimates, targets, and commitments.

When you are asked to provide an estimate, determine whether you're supposed to be estimating or figuring out how to hit a target.
Estimates as Probability Statements

- Software estimates are often presented as single-point numbers - “this project will take 14 weeks”.
- A single-point estimate is usually a target masquerading as an estimate!
- Accurate estimates acknowledge that project outcomes follow a probability distribution.
What is a “Good” Estimate?

• An explicitly stated probability is a sign of a good estimate:
  • “We are 90% confident in a 20-week schedule”.
  • “We estimate a best case of 16 weeks and a worst case of 24 weeks”.
  • “We are estimating 16 to 24 weeks”.

• A good estimation approach should provide estimates that are within 25% of the actual results, 75% of the time.

• **NOTE:** Accurate estimation results cannot be accomplished through estimation practices alone. They must be supported by effective project control.
Purpose of Estimation

- The primary purpose of software estimation is not to predict a project's outcome – it is to determine whether the project's targets are realistic enough to allow the project to be controlled to meet them.
- If the initial target and the initial estimate are within about 20% of each other, the PM will be able to control the feature set, schedule, team size, and other parameters to meet the project's business goals.
- “A good estimate is an estimate that provides a clear enough view of the project reality to allow the project leadership to make good decisions about how to control the project to hit its targets” - Steve McConnell
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
- Effort Estimation in Practice
Cone of Uncertainty

- Estimates become more accurate as a project progresses.
- Estimates created at “Initial Concept” time can be inaccurate by a factor of 4x (on the high side) or 4x on the low side.
- Estimation accuracy improves rapidly for the first 30% of the project, improving from ±4x to ±1.25x.
- Your estimate cannot have more accuracy than is possible at your project's current position within the Cone.
Cone of Uncertainty

![Cone of Uncertainty Diagram]

- Initial Concept
- Approved Product Definition
- User Interface Design Complete
- Detailed Design Complete
- Requirements Complete
- Time
- Software Complete

Estimate Variability
## Estimation Error Table

<table>
<thead>
<tr>
<th>Phase</th>
<th>Possible Error on Low Side</th>
<th>Possible Error on High Side</th>
<th>Range of High to Low Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Concept</td>
<td>0.25 x (-75%)</td>
<td>4.0 x (+300%)</td>
<td>16 x</td>
</tr>
<tr>
<td>Approved Product Definition</td>
<td>0.50 x (-50%)</td>
<td>2.0 x (+100%)</td>
<td>4 x</td>
</tr>
<tr>
<td>Requirements Complete</td>
<td>0.67 x (-33%)</td>
<td>1.5 x (+50%)</td>
<td>2.25 x</td>
</tr>
<tr>
<td>User Interface Design Complete</td>
<td>0.80 x (-20%)</td>
<td>1.25 x (+25%)</td>
<td>1.6 x</td>
</tr>
<tr>
<td>Detailed Design Complete</td>
<td>0.90 x (-10%)</td>
<td>1.10 x (+10%)</td>
<td>1.2 x</td>
</tr>
</tbody>
</table>

Source: Adapted from *Software Estimation with Cocomo II* (Boehm et al. 2000).
Activities Commonly Missed in Estimates

- Ramp-up time for new team members
- Mentoring of new team members
- Management coordination/manager meetings and company meetings
- Requirements clarifications
- Maintaining the revision control system
- Review of technical documentation
- Coordination with team members
- Technical support of existing systems during the project
- Maintenance work on previous systems during the project
- Integration work
- Reviewing plans, estimates, architecture, detailed designs, stage plans, code, test cases, etc.
- Input to user documentation and review of user documentation
- Vacations, holidays and Sick days
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
- Effort Estimation in Practice
Determinant Factors of Estimates

- Project size
  - Don't assume that effort scales up linearly as project size does. Effort scales up exponentially!
- The kind of software
- Personnel factors
- The programming language
Project Size

- The largest driver in a software estimate is the size of the software being built.
  - A system consisting of one million LOC requires dramatically more effort than a system consisting of only 100,000 LOC.

- This fundamental factor is commonly violated in two ways:
  1) Costs, effort and schedule are estimated without knowing how big the software will be.
  2) Costs, effort and schedule are not adjusted when the size of the software is consciously increased (in response to change requests).
Project Size and Productivity

Kind of Software

- The kind of software being developed is the next biggest influence on estimates.
- A team developing an intranet system for internal use may generate code 10 to 20 times faster than a team working on an avionics project, or a real-time project, or an embedded-systems project.
### Productivity Rates of Common Project Types

<table>
<thead>
<tr>
<th>Kind of Software</th>
<th>10,000-LOC Project</th>
<th>100,000-LOC Project</th>
<th>250,000-LOC Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>100–1,000 (200)</td>
<td>20–300 (50)</td>
<td>20–200 (40)</td>
</tr>
<tr>
<td><strong>Business Systems</strong></td>
<td>800–18,000 (3,000)</td>
<td>200–7,000 (600)</td>
<td>100–5,000 (500)</td>
</tr>
<tr>
<td>Command and Control</td>
<td>200–3,000 (500)</td>
<td>50–600 (100)</td>
<td>40–500 (80)</td>
</tr>
<tr>
<td>Embedded Systems</td>
<td>100–2,000 (300)</td>
<td>30–500 (70)</td>
<td>20–400 (60)</td>
</tr>
<tr>
<td><strong>Internet Systems</strong> (public)</td>
<td>600–10,000 (1,500)</td>
<td>100–2,000 (300)</td>
<td>100–1,500 (200)</td>
</tr>
<tr>
<td>Intranet Systems (internal)</td>
<td>1,500–18,000 (4,000)</td>
<td>300–7,000 (800)</td>
<td>200–5,000 (600)</td>
</tr>
</tbody>
</table>
Programming Language

- The project team's experience with the specific language and tools has about a 40% impact on the overall productivity rate.
- Some languages generate more functionality per line of code than others.
- A strong tool support and environment associated with the language can reduce the project effort by about 50% compared to a weak toolset.
- Developers working in interpreted languages tend to be more productive than those working in compiled languages.
# Programming Language Productivity

**Table 5-3: Ratio of High-Level-Language Statements to Equivalent C Code**

<table>
<thead>
<tr>
<th>Language</th>
<th>Level Relative to C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1 to 1</td>
</tr>
<tr>
<td>C#</td>
<td>1 to 2.5</td>
</tr>
<tr>
<td>C++</td>
<td>1 to 2.5</td>
</tr>
<tr>
<td>Cobol</td>
<td>1 to 1.5</td>
</tr>
<tr>
<td>Fortran 95</td>
<td>1 to 2</td>
</tr>
<tr>
<td>Java</td>
<td>1 to 2.5</td>
</tr>
<tr>
<td>Macro Assembly</td>
<td>2 to 1</td>
</tr>
<tr>
<td>Perl</td>
<td>1 to 6</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>1 to 6</td>
</tr>
<tr>
<td>SQL</td>
<td>1 to 10</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>1 to 4.5</td>
</tr>
</tbody>
</table>
Overview

• Introduction
• Estimation Accuracy
• Estimate Influences
• Software Costs
• Developer Productivity
• Estimation Techniques
• Effort Estimation in Practice
Software Cost Components

- Hardware and software costs
- Travel and training costs
- **Effort costs** (dominant factor in most projects)
  - Salaries of engineers involved in the project
  - Social and insurance costs
  - **Overheads**
    - costs of building, heating, lighting
    - costs of networking and communications
    - costs of shared facilities (e.g. library, staff restaurant, etc.)
Costing and Pricing

- Estimates are made to discover the cost (to the development organization), of producing a software system.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organizational, economic, political and business considerations influence the price charged.
## Software Pricing Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market opportunity</td>
<td>A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.</td>
</tr>
<tr>
<td>Cost estimate uncertainty</td>
<td>If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.</td>
</tr>
<tr>
<td>Contractual terms</td>
<td>A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.</td>
</tr>
<tr>
<td>Requirements volatility</td>
<td>If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.</td>
</tr>
<tr>
<td>Financial health</td>
<td>Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.</td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
- Effort Estimation in Practice
Developer Productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.
Productivity Measures

- **Size related measures** based on some output from the software process. This may be lines of delivered source code (LOC, KLOC), object code instructions, etc.

- **Function-related measures** based on an estimate of the functionality of the delivered software.
  - **Function-points** are the best known of this type of measure.
Lines of Code (LOC)

- **What is a line of code?**
  - The measure was first proposed when programs were typed on cards with one line per card;
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line.

- **What programs should be counted as part of the system?**
- **The coding rates for different kinds of software are vastly different.**
- **How is the LOC metric related to the effort in requirements elicitation, architecture and design, documentation etc.?**
LOC as a Productivity Measurement

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.
# High & Low Level Languages

<table>
<thead>
<tr>
<th></th>
<th>Analysis</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assembly code</strong></td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>8 weeks</td>
<td>10 weeks</td>
<td>2 weeks</td>
</tr>
<tr>
<td><strong>High-level language</strong></td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>4 weeks</td>
<td>6 weeks</td>
<td>2 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Effort</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assembly code</strong></td>
<td>5000 lines</td>
<td>28 weeks</td>
<td>714 lines/month</td>
</tr>
<tr>
<td><strong>High-level language</strong></td>
<td>1500 lines</td>
<td>20 weeks</td>
<td>300 lines/month</td>
</tr>
</tbody>
</table>
Function Points

- Based on a combination of program characteristics:
  - External inputs and outputs;
    - Screens, forms, dialog boxes, reports, graphs etc.
  - User interactions;
  - External interfaces;
    - Each major logical group of data or control information.
  - Files used by the system.
    - Flat file, relational database table etc.
- A weight (low, medium & high complexity) is associated with each of these and the function point count is computed by multiplying each raw count by the weight and summing all values.
Function-Points to LoC Conversion

- FPs can be used to estimate LOC depending on the 'average number of lines of code per function point' (AVC) for a given language
  - LOC = AVC * number of function points;
  - AVC is a language-dependent factor varying from 200-300 for assembly language to 2-40 for a 4GL;
Object Points

• Object points (alternatively named application points) are an alternative function-related measure when 4GLs or similar languages are used for development.
• Object points are NOT the same as object classes.
• The number of object points in a program is a weighted estimate of,
  • The number of separate screens that are displayed;
  • The number of reports that are produced by the system;
  • The number of program modules that must be developed to supplement the database code;
Object Point Estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and programming language modules.
- They can therefore be estimated at a fairly early point in the development process.
- At this stage, it is very difficult to estimate the number of lines of code in a system.
## Factors Affecting Productivity

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain experience</td>
<td>Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.</td>
</tr>
<tr>
<td>Process quality</td>
<td>The development process used can have a significant effect on productivity. This is covered in Chapter 31.</td>
</tr>
<tr>
<td>Project size</td>
<td>The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.</td>
</tr>
<tr>
<td>Technology support</td>
<td>Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.</td>
</tr>
<tr>
<td>Working environment</td>
<td>As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.</td>
</tr>
</tbody>
</table>
Quality & Productivity

- All metrics based on volume/unit time are flawed because they do not take quality into account.
- Productivity may generally be increased at the cost of quality.
- It is not clear how productivity/quality metrics are related.
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
  - Estimation as an Art Techniques
  - Estimation as a Science Technique
- Effort Estimation in Practice
Estimation Techniques

- Current software estimation research is focused on improving techniques so organizations can achieve project results within ±5% of estimated results (instead of ±10%).
  - Mathematically intensive, and typically available within software estimation tools.
  - Also known as “science of estimation” practices.
  - Example: COCOMO 2

- Many organizations are looking to reduce estimation error by 25% or less.
  - Simple formulas, “rules of thumb” procedures etc.
  - Also known as “art of estimation” techniques.
Estimation Fundamentals

- **Count, Compute, Judge**
  - *Count* if at all possible.
    - Find something that can be counted, that would meaningfully measure the scope of work.
    - **Examples**: Features, use cases, marketing requirements, user stories, tasks, classes etc.
  - *Compute* when counting is not possible.
    - For improved accuracy, collect historical data that allows computation of an estimate from a count.
    - If historical data is not available, industry average data can be used (with weaker accuracy).
  - *Use judgment* alone, only as a last resort.
# Quantities that can be Counted

<table>
<thead>
<tr>
<th>Quantity to Count</th>
<th>Historical Data Needed to Convert the Count to an Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marketing requirements</td>
<td>• Average effort hours per requirement for development</td>
</tr>
<tr>
<td></td>
<td>• Average effort hours per requirement for independent testing</td>
</tr>
<tr>
<td></td>
<td>• Average effort hours per requirement for documentation</td>
</tr>
<tr>
<td></td>
<td>• Average effort hours per requirement to create engineering requirements from marketing requirements</td>
</tr>
<tr>
<td>Features</td>
<td>• Average effort hours per feature for development and/or testing</td>
</tr>
<tr>
<td>Use cases</td>
<td>• Average total effort hours per use case</td>
</tr>
<tr>
<td></td>
<td>• Average number of use cases that can be delivered in a particular amount of calendar time</td>
</tr>
<tr>
<td>Stories</td>
<td>• Average total effort hours per story</td>
</tr>
<tr>
<td></td>
<td>• Average number of stories that can be delivered in a particular amount of calendar time</td>
</tr>
</tbody>
</table>
Quantities that can be Counted

<table>
<thead>
<tr>
<th>Engineering requirements</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average number of engineering requirements that can be <em>formally</em> inspected per <em>hour</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average effort hours per requirement for development/test/documentation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Points</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average development/test/documentation effort per Function Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average lines of code in the target language per Function Point</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change requests</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average development/test/documentation effort per change request (depending on variability of the change requests, the data might be decomposed into average effort per small, medium, and large change request)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web pages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average effort per Web page for <em>user</em> interface work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average whole-project effort per Web page (less reliable, but can be an interesting data point)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reports</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average effort per report for report work</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dialog boxes</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average effort per dialog for user interface work</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Quantities that can be Counted

<table>
<thead>
<tr>
<th>Category</th>
<th>Quantitative Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database tables</td>
<td>- Average effort per table for database work</td>
</tr>
<tr>
<td></td>
<td>- Average whole-project effort per table (less reliable, but can be an interesting data point)</td>
</tr>
<tr>
<td>Classes</td>
<td>- Average effort hours per class for development</td>
</tr>
<tr>
<td></td>
<td>- Average effort hours to formally inspect a class</td>
</tr>
<tr>
<td></td>
<td>- Average effort hours per class for testing</td>
</tr>
<tr>
<td>Defects found</td>
<td>- Average effort hours per defect to fix</td>
</tr>
<tr>
<td></td>
<td>- Average effort hours per defect to regression test</td>
</tr>
<tr>
<td></td>
<td>- Average number of defects that can be corrected in a particular amount of calendar time</td>
</tr>
<tr>
<td>Configuration settings</td>
<td>- Average effort per configuration setting</td>
</tr>
<tr>
<td>Lines of code already written</td>
<td>- Average number of defects per line of code</td>
</tr>
<tr>
<td></td>
<td>- Average lines of code that can be formally inspected per hour</td>
</tr>
<tr>
<td></td>
<td>- Average new lines of code from one release to the next</td>
</tr>
<tr>
<td>Test cases already written</td>
<td>- Average amount of release-stage effort per test case</td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
  - Estimation as an Art Techniques
  - Estimation as a Science Technique
- Effort Estimation in Practice
'Art of Estimation' Techniques

- Individual Expert Judgment
- Estimation by Analogy
- Proxy-Based Estimates
Individual Expert Judgment

- Most common approach used in practice.
- Bottom-up estimation based on expert judgment estimates of individual tasks.
- Ideally, task-level estimates should be done by the developers who will actually do the task.
- **Task Granularity:** Accuracy can be improved by separating large tasks into smaller tasks
  - Decompose into tasks that will require no more than 2 days of effort
Individual Expert Judgment

- **Use of Ranges**: For each feature, obtain the 'Best Case', 'Worst Case', and 'Most Likely Case' estimates.
- **Checklists**: Use an estimation checklist to improve individual estimates (to prevent consideration omissions).
- **Formulas**: Use PERT (Program Evaluation and Review Technique) to compute 'Expected Case'.

\[
\text{ExpectedCase} = \frac{\text{BestCase} + (4 \times \text{MostLikelyCase}) + \text{WorstCase}}{6}
\]
Individual Estimation Example

<table>
<thead>
<tr>
<th>Feature</th>
<th>Best Case</th>
<th>Most Likely Case</th>
<th>Worst Case</th>
<th>Expected Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature 1</td>
<td>1.25</td>
<td>1.5</td>
<td>2.0</td>
<td>1.54</td>
</tr>
<tr>
<td>Feature 2</td>
<td>1.5</td>
<td>1.75</td>
<td>2.5</td>
<td>1.83</td>
</tr>
<tr>
<td>Feature 3</td>
<td>2.0</td>
<td>2.25</td>
<td>3.0</td>
<td>2.33</td>
</tr>
<tr>
<td>Feature 4</td>
<td>0.75</td>
<td>1</td>
<td>2.0</td>
<td>1.13</td>
</tr>
<tr>
<td>Feature 5</td>
<td>0.5</td>
<td>0.75</td>
<td>1.25</td>
<td>0.79</td>
</tr>
<tr>
<td>Feature 6</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
<td>0.46</td>
</tr>
<tr>
<td>Feature 7</td>
<td>1.5</td>
<td>2</td>
<td>2.5</td>
<td>2.00</td>
</tr>
<tr>
<td>Feature 8</td>
<td>1.0</td>
<td>1.25</td>
<td>1.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Feature 9</td>
<td>0.5</td>
<td>0.75</td>
<td>1.0</td>
<td>0.75</td>
</tr>
<tr>
<td>Feature 10</td>
<td>1.25</td>
<td>1.5</td>
<td>2.0</td>
<td>1.54</td>
</tr>
<tr>
<td>TOTAL</td>
<td>10.5</td>
<td>13.25</td>
<td>18.25</td>
<td>13.62</td>
</tr>
</tbody>
</table>
Checklist

1. Is what's being estimated clearly defined?
2. Does the estimate include all the kinds of work needed to complete the task?
3. Does the estimate include all the functionality areas needed to complete the task?
4. Is the estimate broken down into enough detail to expose hidden work?
5. Did you look at documented facts (written notes) from past work rather than estimating purely from memory?
6. Is the estimate approved by the person who will actually do the work?
7. Is the productivity assumed in the estimate similar to what has been achieved on similar assignments?
8. Does the estimate include a Best Case, Worst Case, and Most Likely Case?
9. Is the Worst Case really the worst case? Does it need to be made even worse?
10. Is the Expected Case computed appropriately from the other cases?
11. Have the assumptions in the estimate been documented?
12. Has the situation changed since the estimate was prepared?
'Art of Estimation' Techniques

- Individual Expert Judgment
- Estimation by Analogy
- Proxy-Based Estimates
Estimation by Analogy

- Get detailed size, effort, and cost results for a similar previous project. If possible, get the information decomposed by feature area, by work breakdown structure (WBS) category, or by some other decomposition scheme.
- Compare the size of the new project piece-by-piece to the old project.
- Build up the estimate for the new project's size as a percentage of the old project's size.
- Create an effort estimate based on the size of the new project compared to the size of the previous project.
- Check for consistent assumptions across the old and new projects.
'Art of Estimation' Techniques

• Individual Expert Judgment
• Estimation by Analogy
• Proxy-Based Estimates
  • Fuzzy Logic Approach
  • Standard Components Approach
  • T-Shirt Sizing
Proxy-Based Estimates

- Identify a *proxy* that is correlated with what you really want to estimate, and which is easier to estimate or count.
- Useful technique to create whole-project estimates and for providing whole-project insights.
- Not suitable for creating detailed task-by-task or feature-by-feature estimates.
'Art of Estimation' Techniques

- Individual Expert Judgment
- Estimation by Analogy
- Proxy-Based Estimates
  - Fuzzy Logic Approach
  - Standard Components Approach
  - T-Shirt Sizing
Fuzzy Logic Approach

- Features are classified as,
  - Very Small
  - Small
  - Medium
  - Large
  - Very Large.
- Use historical data of average LOC for each category to compute the total LOC.
- Fuzzy logic works best when the sizes are calibrated from the organization's historical data.
- The differences in size between adjacent categories should be at least a factor of 2.
Example: Size Estimation with Fuzzy Logic

<table>
<thead>
<tr>
<th>Feature Size</th>
<th>Average Lines of Code per Feature</th>
<th>Number of Features</th>
<th>Estimated Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Small</td>
<td>127</td>
<td>22</td>
<td>2,794</td>
</tr>
<tr>
<td>Small</td>
<td>253</td>
<td>15</td>
<td>3,795</td>
</tr>
<tr>
<td>Medium</td>
<td>500</td>
<td>10</td>
<td>5,000</td>
</tr>
<tr>
<td>Large</td>
<td>1,014</td>
<td>30</td>
<td>30,420</td>
</tr>
<tr>
<td>Very Large</td>
<td>1,998</td>
<td>27</td>
<td>53,946</td>
</tr>
<tr>
<td>TOTAL</td>
<td>-</td>
<td>104</td>
<td>95,955</td>
</tr>
</tbody>
</table>
Effort Estimation with Fuzzy Logic

- Fuzzy logic can be used to estimate effort if underlying data exists.

<table>
<thead>
<tr>
<th>Size</th>
<th>Average Staff Days per Feature</th>
<th>Number of Features</th>
<th>Estimated Effort (Staff Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Small</td>
<td>4.2</td>
<td>22</td>
<td>92.4</td>
</tr>
<tr>
<td>Small</td>
<td>8.4</td>
<td>15</td>
<td>126</td>
</tr>
<tr>
<td>Medium</td>
<td>17</td>
<td>10</td>
<td>170</td>
</tr>
<tr>
<td>Large</td>
<td>34</td>
<td>30</td>
<td>1,020</td>
</tr>
<tr>
<td>Very Large</td>
<td>67</td>
<td>27</td>
<td>1,809</td>
</tr>
<tr>
<td>TOTAL</td>
<td>-</td>
<td>104</td>
<td>3,217</td>
</tr>
</tbody>
</table>
'Art of Estimation' Techniques

- Individual Expert Judgment
- Estimation by Analogy
- Proxy-Based Estimates
  - Fuzzy Logic Approach
  - Standard Components Approach
  - T-Shirt Sizing
Standard Components Approach

• Can be used when many projects are architecturally similar to each other.
• First, the relevant elements/components from previous systems need to be identified (dynamic web pages, static web pages, database tables, business rules, screens etc.).
• The average LOC per component should then be computed from past systems.
• Once the historical data is prepared, the number of standard components in the new system should be estimated.
## Standard Components Example

<table>
<thead>
<tr>
<th>Standard Component</th>
<th>LOC per Component</th>
<th>Minimum Possible Number</th>
<th>Most Likely Number</th>
<th>Maximum Possible Number</th>
<th>Estimated Number</th>
<th>Estimated LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Web pages</td>
<td>487</td>
<td>11</td>
<td>25</td>
<td>50</td>
<td>26.8</td>
<td>13,052</td>
</tr>
<tr>
<td>Static Web pages</td>
<td>58</td>
<td>20</td>
<td>35</td>
<td>40</td>
<td>33.3</td>
<td>1,931</td>
</tr>
<tr>
<td>Database tables</td>
<td>2,437</td>
<td>12</td>
<td>15</td>
<td>20</td>
<td>15.3</td>
<td>37,286</td>
</tr>
<tr>
<td>Reports</td>
<td>288</td>
<td>8</td>
<td>12</td>
<td>20</td>
<td>12.7</td>
<td>3,658</td>
</tr>
<tr>
<td>Business rules</td>
<td>8,327</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>8,327</td>
</tr>
<tr>
<td>TOTAL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>64,254</td>
</tr>
</tbody>
</table>
'Art of Estimation' Techniques

- Individual Expert Judgment
- Estimation by Analogy
- Proxy-Based Estimates
  - Fuzzy Logic Approach
  - Standard Components Approach
  - T-Shirt Sizing
T-Shirt Sizing

- Useful in situations where non-technical stakeholders want (need) to make decisions about project scope during the wide part of the Cone of Uncertainty.

- Developers classify each feature's size relative to other features as 'Small', 'Medium', 'Large' or 'Extra Large'.

- In parallel, non-technical stakeholders classify each feature's business value on the same scale.

- The two sets of entries are then combined, allowing decisions to be made based on the relationship between business value and development cost of each feature.

- Allows for early-in-project decisions to rule out certain features.
### T-Shirt Sizing Example

<table>
<thead>
<tr>
<th>Feature</th>
<th>Business Value</th>
<th>Development Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature A</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>Feature B</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td>Feature C</td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td>Feature D</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Feature E</td>
<td>Medium</td>
<td>Large</td>
</tr>
<tr>
<td>Feature F</td>
<td>Large</td>
<td>Medium</td>
</tr>
<tr>
<td>Feature G</td>
<td>Small</td>
<td>Small</td>
</tr>
<tr>
<td>Feature H</td>
<td>Small</td>
<td>Medium</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature ZZ</td>
<td>Small</td>
<td>Small</td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
  - Estimation as an Art Techniques
  - Estimation as a Science Technique
- Effort Estimation in Practice
Algorithmic Effort Estimation Modelling

- A formulaic approach which is generally based on the size of the software
- Early models were typically regression-based (based on historical information)
- Now, there are sophisticated models based on,
  - Case-based reasoning,
  - Classification and regression trees,
  - Simulation,
  - Neural networks,
  - Bayesian statistics,
  - Lexical analyses of requirement specifications,
  - Genetic programming / Linear programming,
  - Economic production models, and
  - Combinations of one or more of these models.
Algorithmic Effort Estimation Modelling

- Most regression-based effort estimation models feature mathematical functions of product, project and process attributes
- \[ \text{Effort} = A \times \text{Size}^B \times M \]
  - \( A \) is an organisation independent constant
  - \( \text{Size} \) is functionality-based or estimated LOC-based
  - \( B \) reflects the disproportionate effort for large projects
  - \( M \) is the “adjustment factor”, a multiplier reflecting product, process and people attributes → typically a weighted sum of parameters related to project complexity, team skills, support tools, process etc.
- When using such models, it is important to calibrate the model to the organization, based on historical data.
The COCOMO Model

- COnstructive COst MOdel is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity.
- An empirical model based on project experience by Dr. Barry Boehm.
- Well-documented, ‘independent’ model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.
COCOMO 2 Models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - **Application composition model.** Used when software is composed from existing parts.
  - **Early design model.** Used when requirements are available but design has not yet started.
  - **Reuse model.** Used to compute the effort of integrating reusable components.
  - **Post-architecture model.** Used once the system architecture has been designed and more information about the system is available.
Basic COCOMO applies the parameterized equation without much detailed consideration of project characteristics. To use it, you decide which of the three project types best characterizes the project, then use one of these sets of parameter values:

**ORGANIC**

\[ \text{Person Months} = 2.4 \times \text{KDSI} \]

\[ a = 2.4 \quad b = 1.05 \]

**SEMI-DETACHED**

\[ \text{Person Months} = 3.0 \times \text{KDSI} \]

\[ a = 3.0 \quad b = 1.12 \]

**EMBEDDED**

\[ \text{Person Months} = 3.6 \times \text{KDSI} \]

\[ a = 3.6 \quad b = 1.20 \]

This will normally be represented as a table, which presumes we know the basic form of the model:

<table>
<thead>
<tr>
<th>Basic COCOMO</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORGANIC</td>
<td>2.4</td>
<td>1.05</td>
</tr>
<tr>
<td>SEMI-DETACHED</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>EMBEDDED</td>
<td>3.6</td>
<td>1.20</td>
</tr>
</tbody>
</table>

KDSI = Delivered Source Instructions (in thousands)

Well understood, Developed by small teams.

More complex projects. Team members may have limited experience with related projects.

Complex projects. Software is part of a strongly coupled system of hardware, procedures etc.
Intermediate COCOMO

Here we use the same basic equation for the model, but 20 or so project characteristics are rated on a scale of 1 to 5, and these ratings are combined to produce an Effort Adjustment Factor EAF:

\[ \text{EFFORT} = \text{EAF} \times \alpha \times \text{SIZE}^b \]

**Project Characteristics**

An estimator looks closely at many factors of a project, such as amount of external storage required, execution speed constraints, experience of the programmers on the team, experience with the implementation language, use of software tools, etc. For each characteristic, the estimator decided where on the scale of "very low" to "very high" the project falls. Each characteristic gives an adjustment factor (from the table) and all factors are multiplied together to get the total EAF. If a project is judged normal for some factor, the adjustment factor is 1, which will not alter the basic COCOMO model.

**Model Parameters**

In addition to the EAF, the model parameter "\(a\)" is slightly different in Intermediate COCOMO from the basic model. The "\(b\)" parameter remains the same in both models.

<table>
<thead>
<tr>
<th>Intermed. COCOMO</th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORGANIC</td>
<td>3.2</td>
<td>1.05</td>
</tr>
<tr>
<td>SEMI-DETACHED</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>EMBEDDED</td>
<td>2.8</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Estimation Accuracy
- Estimate Influences
- Software Costs
- Developer Productivity
- Estimation Techniques
  - Estimation as an Art Techniques
  - Estimation as a Science Technique
- Effort Estimation in Practice
Estimation Methods

- Each method has strengths and weaknesses
- **Estimation should be based on several methods**
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates
Top-down and Bottom-up Estimation

- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate