Software Estimation

(SSAD - Week 2)

Some content adapted from “Rapid Development” by Steve McConnell

Project Management

- Two over-arching inter-dependent aspects of software projects
  - Process
  - Project Management

Project Management

- Main responsibilities of a project manager are
  - Project planning
  - Project monitoring and control

(Major activities: Software Estimation, Scheduling and Tracking)

Software Estimation

- “Predictions are hard, especially about the future”, Yogi Berra
- Two Types of estimates:
  - Lucky or Lousy

Estimations

- Created, used or refined during
  - Strategic planning
  - Feasibility study
  - Proposals
  - Vendor and sub-contractor evaluation
  - Project planning (iteratively)
- Basic process
  1) Estimate the **size** of the product
  2) Estimate the **effort** (man-hours/man-months)
  3) Estimate the **schedule**
  - NOTE: Not all of these steps are always explicitly performed

Estimations

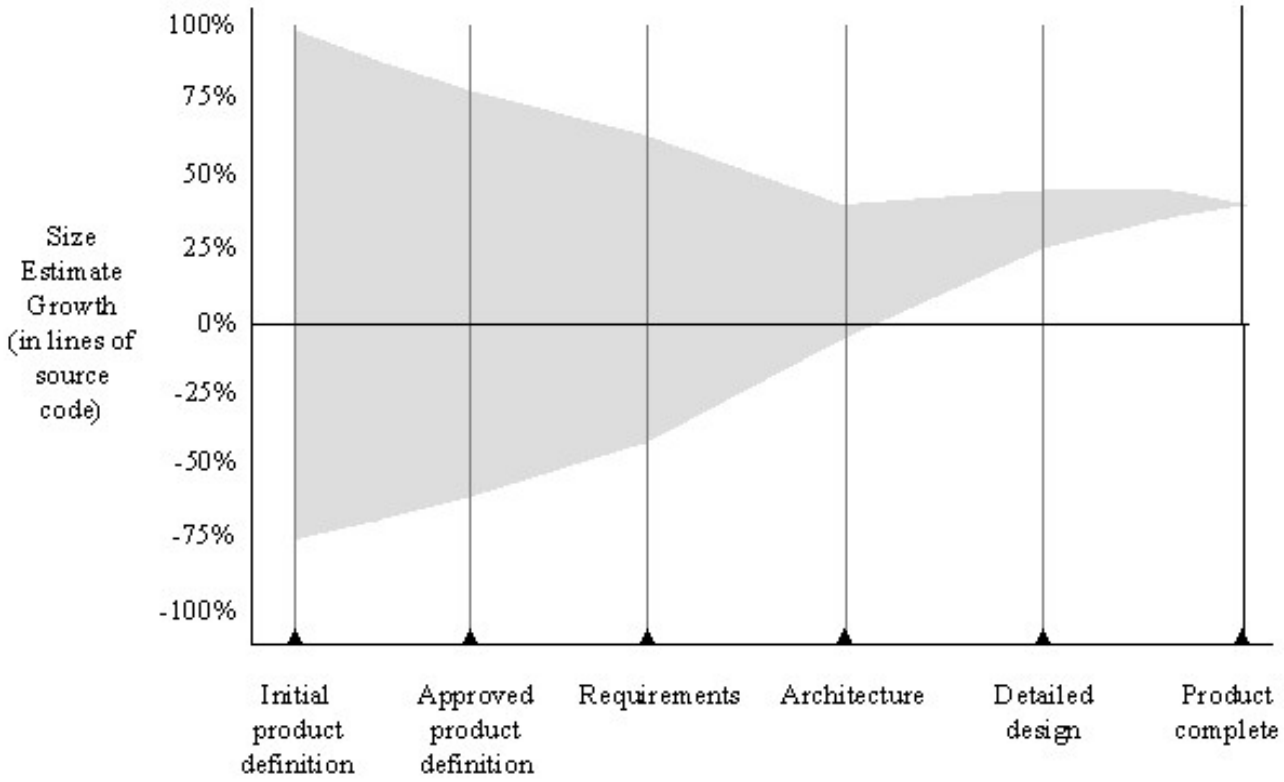
- Remember, an “exact estimate” is an oxymoron
- Estimate how long will it take you to get to dormitory or dining hall from class today-
  - On what basis did you do that?
  - Experience right? (History matters…)
  - Likely as an “average” probability
  - For most software projects there is no such ‘average’

Estimation

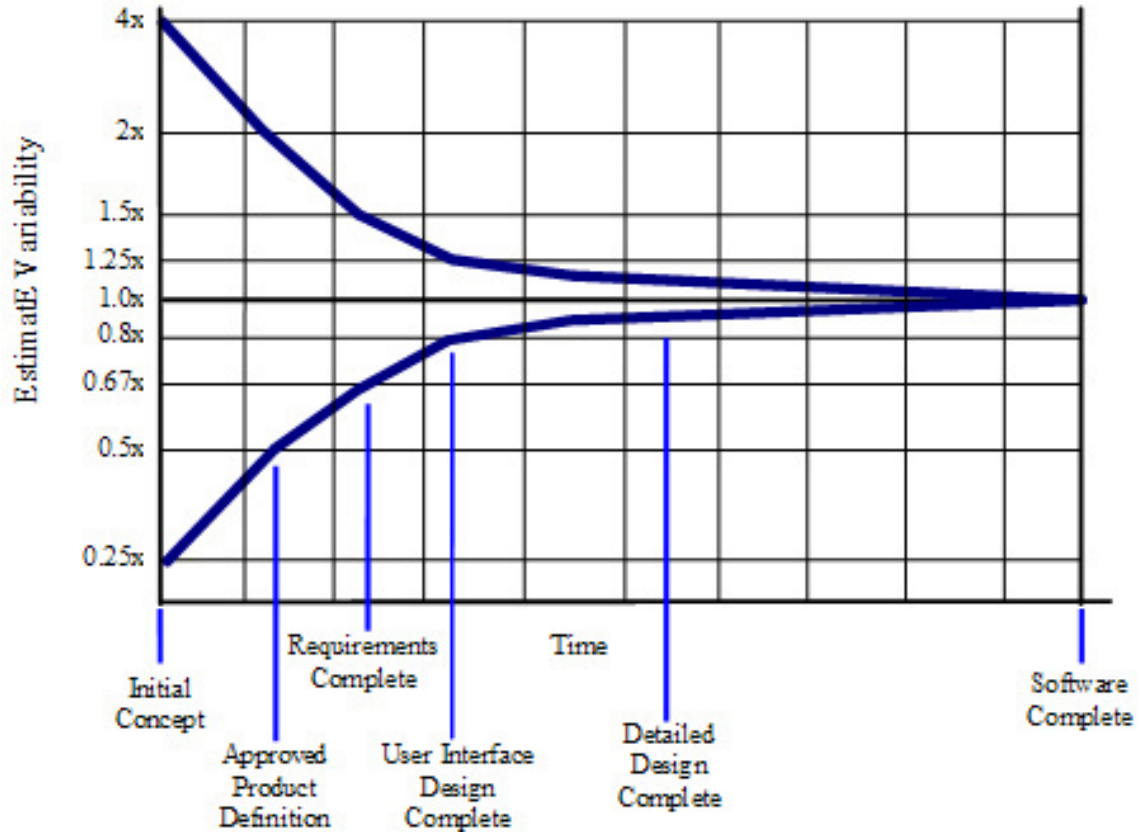
- Target vs. Committed Dates
  - Target: Proposed by business or marketing
  - Do not commit to this too soon!
  - Committed dates: Team agrees to this

  – Let's looks an assignment analogy
    - Do instructors take the various factors into consideration before assigning a deadline?

Cone of Uncertainty



Cone of Uncertainty



From Rapid Development by Steve McConnell

Estimation Methodologies

- Top-down
- Bottom-up
- Analogy
- Expert Judgment
- Priced to Win (request for quote – RFQ)
- Parametric or Algorithmic Method
  - Using formulas and equations

Top-down Estimation

- Based on overall characteristics of project
  - Some of the others can be "types" of top-down (Analogy, Expert Judgment, and Algorithmic methods)
- Advantages
  - Easy to calculate
  - Effective early on (like initial cost estimates)
- Disadvantages
  - Some models are questionable or may not fit
  - Less accurate because it doesn't look at details

Bottom-up Estimation

- Create WBS – Work Breakdown Structure, identify individual tasks to be done.
- Add from the bottom-up
- Advantages
  - Works well if activities well understood
- Disadvantages
  - Specific activities not always known
  - More time consuming

Expert Judgment

- Use somebody who has recent experience on a similar project
- You get a "guesstimate"
- Accuracy depends on their 'real' expertise
- Comparable application(s) must be accurately chosen

Estimation by Analogy

- Use past project
  - Must be sufficiently similar (technology, type, organization)
  - Find comparable attributes (ex: # of inputs/outputs)
- Advantages
  - Based on actual historical data
- Disadvantages
  - Difficulty ‘matching’ project types
  - Prior data may have been mis-measured
  - How to measure differences – no two exactly same

Algorithmic Measures

- Lines of Code (LOC)
- Function points
- Feature points or object points
- LOC and function points most common
  - (of the algorithmic approaches)
- Majority of projects use none of the above

Lines of Code (LOC) based Estimates

- LOC Advantages
  - Commonly understood metric
  - Permits specific comparison
  - Actuals easily measured
- LOC Disadvantages
  - Difficult to estimate early in cycle
  - Counts vary by language
  - Many costs not considered (ex: requirements)
  - Programmers may be rewarded based on this
    - Can use: # defects/# LOC
  - Code generators produce excess code

LOC Estimate Issues

- How do you know how many in advance?
- What about different languages?
- What about programmer style?
- Stat: avg. programmer productivity: 3,000 LOC/yr
- Most algorithmic approaches are more effective after requirements (or have to be after)

Wideband Delphi

- Group consensus approach
- Present experts with a problem and response form
- Conduct group discussion, collect anonymous opinions, then feedback
- Conduct another discussion & iterate until consensus
- Advantages
  - Easy, inexpensive, utilizes expertise of several people
  - Does not require historical data
- Disadvantages
  - Difficult to repeat
  - May fail to reach consensus, reach wrong one, or all may have same bias

Function Points

- Software size measured by number & complexity of functions it performs
- More methodical than LOC counts
- House analogy
  - House's Square Feet ~= Software LOC
  - # Bedrooms & Baths ~= Function points
  - Former is size only, latter is size & function
- Six basic steps

Function Point Process

- 1. Count # of business functions per category
  - Categories: outputs, inputs, DB inquiries, files or data structures, and interfaces
- 2. Establish Complexity Factor for each and apply
  - Low, Medium, High
  - Set a weighting multiplier for each (0 →15)
  - This results in the "unadjusted function-point total"
- 3. Compute an "influence multiplier" and apply
  - It ranges from 0.65 to 1.35; is based on 14 factors
- 4. Results in "function point total"
  - This can be used in comparative estimates

Function point multipliers

	Function Points		
Program Characteristic	Low Complexity	Medium Complexity	High Complexity
Number of Inputs	x 3	x 4	x 6
Number of Outputs	x 4	x 5	x 7
Inquiries	x 3	x 4	x 6
Logical internal files	x 7	x 10	x 15
External interface files	x 5	x 7	x 10

From Rapid Development by Steve McConnell

Counting the Number of Function Points

	Function Points		
Program Characteristic	Low Complexity	Medium Complexity	High Complexity
Number of Inputs	5 x 3 = 15	2 x 4 = 8	3 x 6 =18
Number of Outputs	6 x 4 = 24	6 x 5 = 30	0 x 7 = 0
Inquiries	0 x 3 = 0	2 x 4 = 8	4 x 6 = 24
Logical internal files	5 x 7 = 35	2 x 10 =20	3 x 15 = 45
External interface files	8 x 5 = 40	0 x 7 = 0	2 x 10 = 20
Unadjusted function-point total			287
Influence multiplier			1.20
Adjusted function-point total			344

From Rapid Development by Steve McConnell

Code Reuse & Estimation

- Does not come for free
- Code types: New, Modified, Reused
- If code is more than 50% modified, it's "new"
- Reuse factors have wide range
  - Reused code takes 30% effort of new
  - Modified is 60% of new
- Integration effort with reused code almost as expensive as with new code

Effort Estimation

- Now that you know the “size”, determine the “effort” needed to build it
- Various models: empirical, mathematical, subjective
- Expressed in units of duration
  - Man-months (‘staff-months’) or Man-hours

COCOMO

- Barry Boehm – 1980's
- **CO**nstructive **CO**st **MO**del
- Input – LOC, Output - Person Months
- Allows for the type of application, size, and "Cost Drivers"
- Cost drivers using High/Med/Low & include
  - Motivation, Ability of team, Application experience, etc.
- Biggest weakness?
  - Requires input of a product size estimate in LOC

Estimation Issues

- Quality estimations needed early but information is limited
- Precise estimation data available at end but not needed
  - Or is it? What about the next project?
- Best estimates are based on past experience
- Politics of estimation:
  - You may anticipate a "cut" by upper management
- For many software projects there is little or none
  - Technologies change
  - Historical data unavailable
  - Wide variance in project experiences/types
  - Subjective nature of software estimation

Over and Under Estimation

- Over estimation issues
  - The project will not be funded
    - Conservative estimates guaranteeing 100% success may mean funding probability of zero.
  - Danger of feature and scope creep
  - Be aware of "double-padding": team member + manager
- Under estimation issues
  - Quality issues (short changing key phases like testing)
  - Inability to meet deadlines
  - Morale and other team motivation issues

Estimation Guidelines

- Estimate iteratively!
  - Process of gradual refinement
  - Make your best estimates at each planning stage
  - Refine estimates and adjust plans iteratively
  - Plans and decisions can be refined in response
  - Balance: too many revisions vs. too few

Know Your Deadlines

- Are they ‘Real Deadlines’?
  - Tied to an external event
  - Have to be met for project to be a success
  - Ex: end of financial year, contractual deadline, Y2K
- Or ‘Artificial Deadlines’?
  - Set by arbitrary authority
  - May have some flexibility (if pushed)

Estimation "Presentation"

- How you present the estimation can have **huge** impact
- Techniques
  - Plus-or-minus qualifiers
    - 6 months +/-1 month
  - Ranges
    - 6-8 months
  - Risk Quantification
    - +/- with added information
    - +1 month of new tools not working as expected
    - -2 weeks for less delay in hiring new developers
  - Cases
    - Best / Planned / Current / Worst cases
  - Coarse Dates
    - Q3 02
  - Confidence Factors
    - April 1 – 10% probability, July 1 – 50%, etc.

Other Estimation Factors

- Account for resource experience or skill
  - Up to a point
  - Often needed more on the "low" end, such as for a new or junior person
- Allow for "non-project" time & common tasks
  - Meetings, phone calls, web surfing, sick days
- There are commercial 'estimation tools' available
  - They typically require configuration based on past data

Summary

- Software estimation involves estimation of
  - Size
  - Effort
  - Resources
- There are various estimation techniques. For example
  - Wideband Delphi
  - CoCoMo