Software Estimation

SWEN 256 – Software Process & Project Management
“Predictions are hard, especially about the future”
Yogi Berra

Two Types of estimates: Lucky or Lousy
Created, used or refined during

- Strategic planning
- Feasibility study and/or SOW
- Proposals
- Vendor and sub-contractor evaluation
- Project planning (iteratively)

Basic process

1) Estimate the **size** of the product
2) Estimate the **effort** (man-months)
3) Estimate the **schedule**

- NOTE: Not all of these steps are always explicitly performed
Remember, an “exact estimate” is an oxymoron

Estimate how long will it take you to get home from class today-

- On what basis did you do that?
- Experience right?
- Likely as an “average” probability
- For most software projects there is no such ‘average’
Target vs. Committed Dates
  • Target: Proposed by business or marketing
  • Do not commit to this too soon!
  • Committed dates: Team agrees to this
Cone of Uncertainty

Size Estimate Growth (in lines of source code)

Initial product definition  Approved product definition  Requirements  Architecture  Detailed design  Product complete

Estimation Methodologies

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

Rand Corp. used orig. Delphi approach in the 1940’s to predict future technologies

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
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)
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
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
Each user scenario is considered separately

The scenario is decomposed into a set of engineering tasks

Each task is estimated separately
  - May use historical data, empirical model, or experience
  - Scenario volume can be estimated (LOC, FP, use-case count, etc.)

Total scenario estimate computed
  - Sum estimates for each task
  - Translate volume estimate to effort using historical data

The effort estimates for all scenarios in the increment are summed to get an increment estimate
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 (or ‘staff-months’)

Effort Estimation
Barry Boehm – 1980’s

**Constructive Cost Model**

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.
- Parkinson’s Law: Work expands to take the time allowed
- 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
  - See “Death March” by Ed Yordan
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)
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.
For Time or Cost Estimates:

- Aggregation into larger units (Work Packages, Control Accounts, etc.)
- Perform Risk Analysis to calculate Contingency Reserves (Controlled by PM)
- Add Management Reserves: Set aside to cover unforeseen risks or changes (Total company funds available – requires Change Control activities to access)
 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
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
Remember: “manage expectations”

Parkinson’s Law
  - “Work expands to fill the time available”

The Student Syndrome
  - Procrastination until the last minute (cram)
Questions/Discussion