Software Architecture, Process and Management
Estimating Size and Effort

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/sapm
Semester Two 2012-13
Estimation — Non-trivial

- As I hope we showed with our Java2HTML exercise, estimation is a non-trivial task
- This lecture will look at some techniques to overcome this inherent difficulty
- First though we will look at some facts that may cause you to see the situation as even more bleak than you do now
- Most of these facts are taken from “Facts and Fallacies of Software Engineering” by Robert Glass
- These are “Facts” in the sense that they may be disputed
One of the two most common causes of runaway projects is poor estimation

- Runaway projects are those that spiral out of control, often producing no product at all
- If you’re curious the other most common cause is *unstable requirements*, more on that later
- These projects are not failures because the programmers did a poor job but because the estimates in the first place were unreal
- There is little dispute that software estimates are poor, much dispute about how to improve them
Most software estimates are performed at the beginning of the software lifecycle

- It makes sense to make some kind of estimate at the beginning
- Managers wish to know whether to embark upon the project at all
- But you at least need to know what problem you are solving
  - and you cannot know that until (at least) you have the requirements specification
- This was partly the “trick” I played with the Java2HTML example
- Again there is little dispute about this fact
Most estimates are not made by the engineers or their managers

- Most estimates are made by upper-management, customers, or users
- It is more a demand/wish than a genuine attempt at estimation
- Once again there is little or no dispute of this fact
Most estimates are made for the wrong reasons

- As we stated above, most estimates are in fact demands or wishes
- They are made by the constraints of the problem in question
- Perhaps the software project is part of a larger project and so it must fit somewhere within the timeline of the larger project
- Whatever the reason, “estimates” are often made for reasons wholly unrelated to:
  - The size and complexity of the task
  - The group assigned to it
    - the estimate may be made before the group is assigned
    - which might not be a bad thing if there is flexibility
    - often there is not
Software estimates are rarely adjusted as the project proceeds

- Software estimates are often made at the start of the project
  - This is the best time in terms of usefulness of a correct estimate
  - But the worst time in terms of accuracy
- The best time, in terms of accuracy, would be towards the completion of the project
- There is a natural tug-of-war between accuracy and usefulness
- An obvious solution to this is to regularly update estimates
- Sadly this seems to be rarely done
Software projects are often judged on the basis of these terrible estimates

- Unfortunately given how terrible software estimates tend to be, a project is often judged a success/failure based on those estimates
- This is akin to:
  - Guessing how far is it from Edinburgh to New York?
  - Guessing how fast a plane travels
  - Judging the total time taken to travel between the two based on dividing the first estimate by the second
  - Oh and make sure a six-year-old does the guessing
  - And you have to go via London
  - And you possibly have to arrive on the 4th of July
So then, estimates are:

- Done at the wrong time
So then, estimates are:

- Done at the wrong time
- By the wrong people
So then, estimates are:

- Done at the wrong time
- By the wrong people
- For the wrong reasons
- Not updated
- Used to judge the success or otherwise of a project
So then, estimates are:

- Done at the wrong time
- By the wrong people
- For the wrong reasons
- Not updated
So then, estimates are:

▶ Done at the wrong time
▶ By the wrong people
▶ For the wrong reasons
▶ Not updated
▶ Used to judge the success or otherwise of a project
Estimating Software Size and Effort

- Most methods for estimating the total effort required for a software project (to decide on schedule, staffing, and feasibility) depend on the size of the software project.

- Unfortunately, it is difficult to measure size meaningfully, it is difficult to estimate size in advance, and it is difficult to extrapolate from size to what we are really interested in.

- We will first look at methods for estimating size, then at how size can be used to estimate effort (e.g. using COCOMO).
Three-Point Estimates

- If you just ask someone for an estimate of how long a task will take, the answers you get will vary enormously as I hope we’ve demonstrated.
- A further complication is how they implicitly sample from their probability distribution.
  - In plain English do they answer with their: Worst case, Best Case, Average, Median, etc?
- A better way is to provide three-point estimates with the three-points: Optimistic, Most likely and Pessimistic.
- Generally chosen such that:
  - There is a 2.5% chance the project/task is completed before the optimistic estimate.
  - There is a 2.5% chance the project/task is completed after the pessimistic estimate.
  - Hence a 95% that it is completed somewhere in the middle.
Three-Point Estimates
Using Three-Point Estimates

- A single estimate can be computed from a three-point estimate as: \[ E = \frac{o+4m+p}{6}, \quad SD = \frac{p-o}{6} \]
- Suppose we take \( o = 5, \quad m = 15, \quad p = 40 \)
  \[ E = \frac{5+(4\times15)+40}{6} = \frac{105}{6} = 17.5, \quad SD = \frac{40-5}{6} \approx 6 \]
- These calculations are from the PERT method (discussed later), assuming that the actual values will have a Gaussian distribution.

- People still underestimate the pessimistic case (Vose 1996)

- but the results are more repeatable than simply asking for a single number at the start
Approaches to Estimating Size

- Through expert consensus (Wideband-Delphi)
- From historical population data (Fuzzy logic)
- From standard components (Component estimating)
- From a model of function (Function points)
- See “A Discipline for Software Engineering” Humphrey (2002), Addison-Wesley for more information on these methods and other more complicated ones
Why we need consensus

- Expert analysis usually means “Been there, done that”
- Software is different from many other kinds of projects, because software is inherently sharable
- Once you make one of a software project, you have made many of the same project
- That is not true of buildings/bridges/transport infrastructure/stadia
- Granted often such projects want to be new/different
- But software projects are by their nature new
  - Otherwise, you wouldn’t have anything to build
Wideband-Delphi Estimating — Basically Planning Poker

1. Start with a group of experts
2. All experts meet to discuss project
3. Each expert *anonymously* estimates size
4. Each expert gets to see all estimates (*anonymously*)
5. Stop if the estimates are sufficiently close together
6. Otherwise, back to step 2

Helps get a group of engineers committed to a particular schedule
Fuzzy-Logic Estimating

- Break previous projects into categories by size:

<table>
<thead>
<tr>
<th>Range</th>
<th>Nominal KLOC</th>
<th>KLOC range included</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Small</td>
<td>2</td>
<td>1 - 4</td>
</tr>
<tr>
<td>Small</td>
<td>8</td>
<td>4 - 16</td>
</tr>
<tr>
<td>Medium</td>
<td>32</td>
<td>16 - 64</td>
</tr>
<tr>
<td>Large</td>
<td>128</td>
<td>64 - 256</td>
</tr>
<tr>
<td>Very Large</td>
<td>512</td>
<td>256 - 1028</td>
</tr>
</tbody>
</table>

- Then look at the previous projects in each category and decide which category contains projects similar to this one.

- Problem: Only a very rough estimate, yet requires several relevant historical datapoints in each range (rare)
Standard Component Estimating

- Gather historical data on types and sizes of key components
- For each type \((i)\), guess how many you will need \((M_i)\)
- Also guess largest \((L_i)\) and smallest \((S_i)\) extremes
- Estimated number \((E_i)\) is a function of \(M_i\), \(L_i\) and \(S_i\), e.g.:
  \[
  E_i = \frac{S_i + 4M_i + L_i}{6}
  \]
- Total size calculated from estimated number and average size \((X_i)\) of each type: \(X = \sum_i E_i \times X_i\)
- Helps break down a large project into more-easily guessable chunks
Function Point Estimating (1)

- Popular method based on a weighted count of common functions of software
- The five basic functions are:
  - **Inputs**: Sets of data supplied by users or other programs
  - **Outputs**: Sets of data produced for users or other programs
  - **Inquiries**: Means for users to interrogate the system
  - **Data files**: Collections of records which the system modifies
  - **Interfaces**: Files/databases shared with other systems
### Function Point Estimating (2)

<table>
<thead>
<tr>
<th>Function</th>
<th>Count</th>
<th>Weight</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>Outputs</td>
<td>12</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>Inquiries</td>
<td>4</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Data Files</td>
<td>2</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Interfaces</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>135</strong></td>
</tr>
</tbody>
</table>

- The major question is: from where do you get the weights?
- Generally, historical data
Estimating Total Effort

- Once we have the size estimate, we can try to estimate the total effort involved, e.g. in person-months, e.g. to decide on staffing levels.
- Unfortunately, the total amount of effort required depends on the staffing levels.
  - see The Mythical Man-Month, Brooks 1995, Addison-Wesley
- Note: **total** person-months depends upon the number of persons
  - **not just** the wall-clock time
  - That is why the title includes the word “Mythical”
- So it is easy to get stuck in circular reasoning
- Still, with some big assumptions, it is possible to try to use historical experience with similarly sized projects
COCOMO Model

- The Constructive Cost Model (COCOMO; Boehm 1981) is popular for effort estimation
- COCOMO is a mathematical equation that can be fit to measurements of effort for different-sized completed projects
  - providing estimates for future projects
- COCOMO II (Boehm et al. 1995) is the current version
  - see http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html
- but we will focus on the original simpler equation
- All we are hoping to get is a rough (order of magnitude) estimate
Basic COCOMO Model

- In its simplest form COCOMO is: $E = C \times P^s \times M$ where:
  - $E$ is the estimated effort (e.g. in person-months)
  - $C$ is a complexity factor
  - $P$ is a measure of product size (e.g. KLOC)
  - $s$ is an exponent (usually close to 1)
  - $M$ is a multiplier to account for project stages
Basic COCOMO Model Examples

- We ignore the multiplier, M, so \( E = C \times P^s \) Then we fit C and s to historical data from different types of projects:
  - **Simple** \( E = 2.4 \times P^{1.05} \) : A well understood application developed by a small team
  - **Intermediate** \( E = 3.0 \times P^{1.12} \) : A more complex project for which team members have limited experience of related systems
  - **Embedded** \( E = 3.6 \times P^{1.20} \) : A complex project in which the software is part of a complex of hardware, software, regulations and operational constraints
Behaviour of the Basic Examples

- Simple
- Intermediate
- Embedded

- Person-months
- K Lines of Source Code

- 1000
- 0
- 0
- 120
Extending the COCOMO Model

- The basic examples didn't use the multiplier, $M$
- $M$ can be used to adjust the basic estimate by including expert knowledge of the specific attributes of this project
- Potential attributes/constraints to consider:
  - Product attributes (e.g. reliability)
  - Computer attributes (e.g. memory constraints)
  - Personnel attributes (e.g. programming language experience)
  - Project attributes (e.g. project development schedule)
If the basic estimate is 1216 person-months then we can add estimates of the effect of various constraints or attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Magnitude</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Very High</td>
<td>1.4</td>
</tr>
<tr>
<td>Complexity</td>
<td>Very High</td>
<td>1.3</td>
</tr>
<tr>
<td>Memory Constraint</td>
<td>High</td>
<td>1.2</td>
</tr>
<tr>
<td>Tool use</td>
<td>Low</td>
<td>1.1</td>
</tr>
<tr>
<td>Schedule</td>
<td>Accelerated</td>
<td>1.23</td>
</tr>
</tbody>
</table>

\[
E = 1216 \times 1.4 \times 1.3 \times 1.2 \times 1.23 = 3593
\]
COCOMO Multiplier Example 2

- Using the basic estimate is 1216 person-months then we can add estimates of the effect of various constraints or attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Magnitude</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Very Low</td>
<td>0.75</td>
</tr>
<tr>
<td>Complexity</td>
<td>Very Low</td>
<td>0.7</td>
</tr>
<tr>
<td>Memory Constraint</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>Tool use</td>
<td>High</td>
<td>0.9</td>
</tr>
<tr>
<td>Schedule</td>
<td>Normal</td>
<td>1</td>
</tr>
</tbody>
</table>

- New estimate: \( E = 1216 \times 0.75 \times 0.7 \times 1 \times 1 = 575 \)
COCOMO Limitations

- Like any mathematical model, COCOMO has two main potential types of error:
  1. **model** error
  2. **parameter** error

- Model error: Do projects really scale with KLOC as modeled?

- From the COCOMO II web site:
  
  "The 1998 version of the model has been calibrated to 161 data points [projects]... Over those 161 data points, the 98 release demonstrates an accuracy of within 30% of actuals 75% of the time"

- Thus even looking retroactively, with accurate KLOC estimates, 25% of projects are more than 30% mis-estimated
COCOMO Limitations

- Parameter estimation error:
- Can the various parameters be set meaningfully?
- E.g. result depends crucially on KLOC, which is difficult to estimate accurately
- The other parameters can also be difficult to estimate for a new project, particularly at the beginning when scheduling and feasibility need to be decided
Estimation Limitations

- Model predictions can be sensitive to small changes in parameters, so be sure to perform a sensitivity analysis for different parameter estimates.
- In any case, early estimates are likely to be wrong, and should be revised once more data is available.
- Also, predictions can strongly affect the outcome:
  - If estimate is too high, programmers may relax and work on side issues or exploring many alternatives.
  - If estimate is too low, quality may be sacrificed to meet the deadline.
  - It may be worse than that: the quality that is sacrificed may pertain to the code itself.
  - Which may hinder future development.
Models and Small Parameter Changes

- Edward Lorenz was working on weather prediction using a computer to model the weather system.
- One day he noticed an odd curiosity:
  - He had run a particular model for a (simulated) time, say $t=0 - 100$
  - He decided to run the model for longer, but since the simulation was computationally expensive (this was the early 1960s),
  - he started it half-way through, at $t=50$
  - Using the output of the original simulation
  - He found that at time $t=100$, this new simulation differed greatly from the original simulation
It turned out the output was given in less precision (3 digits) than the internal representation (6 digits) so he started the new simulation at very slightly different values than what they were in the original simulation.

Despite the differences being much smaller they gave hugely differing results.

His resulting paper is from where the phrase “The Butterfly Effect” comes from.

“Predictability: Does the Flap of a Butterfly’s Wings in Brazil set off a Tornado in Texas?”
Summary

- No size estimation method is foolproof or particularly accurate
- Even once size is available, hard to extrapolate to effort, cost, estimated schedule, etc.
- Estimates can be self-fulfilling or self-defeating
- Thus it is difficult to evaluate how well estimation is working, even retroactively
- Use an appropriate method for how much data you have — if no data, then gut instinct estimation is reasonable
- Try to avoid depending on your estimates being accurate
Let me say that again, it’s important

- Try to **avoid** depending on your estimates being accurate
SAPM — Related Reading

- **Required Reading**
  - No absolutely required reading today

- **Suggested Reading**
Blog Posts

- We have, so far, one blog post
- “The black art of software engineering”
- I haven’t set the blog up quite yet
- I intend to do this tomorrow morning
Any Questions?