Estimation

Software Architecture, Process, and Management

Last Monday's lecture on project management motivated the need for the estimation of software development tasks. This lecture presents the difficulties in doing so and looks at techniques to overcome them.

Varying Estimates

- Estimates vary wildly
- These are not necessarily caused by either estimator being poor at estimating how long a given task will take
- Though most of us are indeed poor at that
- It is not necessarily caused by a difference in the competence/suitability/required knowledge of the two estimators (though this can be a large factor)
- Often it is caused by the two estimators simply thinking of two different sets of requirements

Varying Estimates

- One may be thinking of production quality code
- One may be thinking of a rough first attempt that allows other parts of the project dependent upon it to start
- Almost any estimate could be made correct by decreasing/increasing the scope/quality/function/performance
- This is a large problem, but for the remainder of this lecture we will assume that the tasks being estimated can be and are well specified

Runaway Projects

Fact: **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
- These projects are not failures because the programmers did a poor job but because the estimates in the first place were unrealistic
- There is little dispute that software estimates are poor
  - But there is much dispute about how to improve them
- In case you are curious the other most common cause is *unstable requirements*

Software Estimates and Timing

Fact: 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
- Again there is little dispute about this fact

Who Estimates?

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
- Recall that the software portion may form only part of the project and must be scheduled as all the other tasks
- Recall that the project planning charts assume **infinite** resources
- Consider coursework deadlines

Estimate Adjustment

Fact: 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
- There seems to be little dispute about this fact

Software Judgement

Fact: 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
- Recall that **our** criteria for success is that a project is completed *on time* and *within budget*

Three-Point Estimates

- 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

Wideband-Delphi Estimating

Basically Planning Poker

1. Start with a group of experts
2. All experts meet to discuss project
3. Each expert **anonymously** and **blindly** 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

Wideband-Delphi Estimating

- It is key that the estimates are done:
  1. **blindly**
     - Such that one person's estimate is not influenced by another's
  2. **anonymously**
     - Such that a dissenter is not pressured into conforming to the group estimate
- Of course both of these are compromised to some extent by the ensueing discussion but it at least avoids the entire group conforming to the first person's estimate

Fuzzy-Logic Estimating

- Break previous projects into categories by size:

Range	Nominal KLOC	KLOC range
Very Small	2	1 - 4
Small	8	4 - 16
Medium	32	16 - 64
Large	128	64 - 256
Very Large	512	256 - 1028

- 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.: $E_i = (S_i + 4M_i + L_i)/6$
- Total size calculated from the estimated number and average size (X_i) of each type: X = sum;$_i$ E_i * X_i
- Helps break down a large project into more-easily guessable chunks

Function Point Estimating

- 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

Function	Count	Weight	Total
Inputs	8	4	32
Outputs	12	5	60
Inquiries	4	4	16
Data Files	2	10	20
Interfaces	1	7	7
Total			135

- The major question is: from where do you get the weights?
- Generally, historical data, regression analysis

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
  - This is the main message defended in Fred Brooks' popular book *The Mythical Man-Month*, Brooks 1995, Addison-Wesley

Estimating Total Effort

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

COCOMO Limitations

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

COCOMO Limitations

- 1. 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

- 2. 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

- 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