Estimating Software Size

Size isn't everything in a software project but it does influence most things (*e.g.* resources, cost).

If we don't have an accurate prediction of size it is difficult to plan.

We look at:

- Different approaches to size estimation.
- Trying to tame the size problem via re-use.

Different Approaches

- Through expert consensus (Wideband-Delphi).
- From historical "population" data (Fuzzy).
- From standard components (Component estimating).
- From a model of function (Function point).

Wideband-Delphi Estimating

1. Group of experts: $[E_1, \ldots, E_i, \ldots, E_n]$.
2. Meet to discuss project.
3. Each anonymously estimates size: $[X_1, \ldots, X_i, \ldots, X_n]$.
4. Each E_i gets to see all the Xs (anonymously).
5. Stop if the estimates are sufficiently close together.
6. Otherwise, back to step 2.

Fuzzy Estimating

Size classes of code in hypothetical domain (in K-LOC).

Range	Size	Low	High
V Small	2	1	4
Small	8	4	16
Medium	32	16	64
Large	128	64	256
V Large	512	256	1028

Need lots of recent historical data for this and it only gives a gross estimate.

Standard Component Estimating

- Gather historical data on key components.
- Guess how many of each type you will need (M_i).
- Also guess largest (L_i) and smallest (S_i) extremes.
- Final estimate (E_i) is a function of M_i, L_i and S_i.
- For example, $E_i = (S_i + (4 * M_i) + L_i)/6$

Function Point Estimating (1)

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)

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

May adjust function point total using "influence factors".

Re-Use

With ground-up programming the cost of development rises sharply as software size increases.

Software size tends to be, on the whole, larger in systems year on year.

Maybe we can take advantage of earlier effort by re-using its products?

What might be Re-used?

- Code
- Designs and architectures.
- Documentation.
- Tests.
- anything else which is experience.

Motivation for Re-use

- Saves money.
- Cumulative debugging.
- Shorter development time.
- Encourages modularity.

Pitfalls of Re-use

- Big components are the most tempting and most difficult to re-use.
- Re-used components are older so may reach obsolescence sooner.
- May not be able to re-use component and documentation.
- May be hard to find the original designer if it goes wrong.
- Hard to find the right thing.
- Tempting to twist project to fit re-usable components.
- It costs to design components specifically for re-use.
- Need to consider re-use in the *previous* project.