COSC345 Week 19

Cost Estimation

12 August 2014

Richard A. O'Keefe

Cost Estimation

To stay in business, plan your spending.

To plan your spending, estimate the cost of your work.

People are amazingly bad at estimating.

Especially software.

They are also bad at appreciating how bad they are.

Learn from your own data.

An exercise

The next two slides hold ten questions.

For each of them, write down your best guess at what the number is, and a (lower bound, upper bound) range that you are 90% confident has the true value.

With 10 questions, 90% confidence means that you think the average number of answers outside your chosen limits should be one. Getting the best estimate is not the goal; getting a good confidence interval is.

Estimate these.

A The height of Mount Cook/Aorangi

B The area of Lake Wakatipu

C The number of Prime Ministers NZ has had

D The number of Maori Kings and Queens there have been

E The number of book publishers in New Zealand in 2010

Estimate these (continued)

F The market size of the New Zealand publishing industry in 2010

G The surface temperature of the Sun

H The number of cell phones sold world wide in 2011

I The number of segments in the human spinal cord

J The number of stars visible to our eyes at night

How well did you all do?

How many of your error bounds contained the true value?

What was the distribution for the class?

You are not expected to be "general knowledge" quizz-kids.

You will not be expert in all areas relevant to your work.

It is ok to be unsure about a fact.

The problem is being more sure than you have a right to.

Results from 2013

Question	In interval	$<$ low	$>$ high	wrong
A	8	1	2	27%
B	3	4	4	73%
C	9	1	1	18%
D	2	7	1	80%
E	0	1	9	100%
F	2	2	6	80%
G	8	2	1	27%
H	4	0	7	64%
I	5	1	5	55%
J	4	7	0	64%

Causes of error

Uncertainty about the question (stars visible anywhere, or where we are? actual conditions or ideal conditions?)

Ignorance of the answer (cell phones)

Optimism: when estimating time to finish something, we tend to ignore interruptions, defects, and other problems

Desire to please

The Cone of uncertainty

As time goes on, we get more information about a project, leading to better estimates.

At the beginning, our estimates will be particularly vague. A factor of *four* either way is plausible.

Break it down

If errors are random, then breaking an estimate into N small estimates and then adding them up will reduce the error by a factor of sqrt(N).

We can get surprisingly good estimates from a sufficiently detailed breakdown, such as we get after the first phase of design.

If we systematically underestimate, this won't help, but if we just overlooked things it may help.

Never trust one number

At least present a range.

Better still, present a distribution,

5% chance of finishing by T_1

50% chance of finishing by T_2

75% chance of finishing by T_3

Range estimates are *always* more honest.

Things to estimate

Size how big a program will be

Effort how many work-days it will take to produce

Time how many calendar days it will take

Cost how much money it will take

Defects how many bugs will need fixing

Estimation should be based on data

Most relevant: data from earlier in this project

Next best: data from other projects in this outfit

Weakly relevant: industry data

Little use: gut feeling

$\rightarrow$ Build your own models from your own data

Size

What are we estimating/measuring? Do you count delivered code, prototypes, unit and system tests, mock objects, scripts, HTML?

Do you count only new code, new and reworked code, all code including code in reuse libraries that isn't actually used, code bought from other people?

SLOC is the worst measure of program size there is, except for all the others.

There are questions about SLOC as well.

It is important to be consistent!

Effort

What are we estimating/measuring? Do you count *real* work hours, or make-believe 8-work-hour-per-day hours? Or just count days?

Do you count holidays, sick leave, time spent at meetings?

Do you count training? Time spent installing support software?

What about people borrowed from another project?

Whose time do you measure? Coders? Testers? Technical writers? Secretaries?

What about time spent dreaming up the project?

It is important to be consistent!

Time

When does the project begin? (According to McConnell, Capers Jones reports that fewer than 1% of projects had a clear beginning.)

When does the project end? When the first release ships? When the last one does?

Be clear about when each phase of the project begins and ends.

It is important to be consistent!

Defects

Count reports or causes?

Count developer reports like customer reports?

Count design defects, not just code?

Count documentation defects?

Do you track how hard each defect is to fix?

Techniques

COCOMO II (famous, from a variety of old projects)

Function Points (famous, based on EDP)

Gut feel

Something tuned to your industry segment/own data

Simulation

Yourdon's book on "Death March Projects" suggests simulating the process so that you can try different policies.

Resources (Books)

Software Estimation: Demystifying the Black Art, Steve McConnell, Microsoft Press 2006. This is by far the best introduction to the subject I've seen. He thinks his readers will be terrified by Statistics; I do hope he's wrong about that.

Death March, 2nd edition, Edward Yourdon, Pearson Education 2004. "A death march project is one for which an unbiased, objective, risk assessment ... determines that the risk of failure is $\geq$ 50%."

Resources (Web sites)

Steve McConnell's web page about estimation:
http://www.construx.com/Blog_Main/?TagId=149685

The COCOMO II web site:
http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

The Wikipedia entry about COCOMO:
http://en.wikipedia.org/wiki/COCOMO

The Wikipedia entry about Software Estimation:
http://en.wikipedia.org/wiki/Estimation_in_software_engineering

Function Points web site:
http://www.ifpug.org/