Estimating Software Effort using FOSS Data

ELCE Dublin 5th October 2015

Paul Sherwood

Estimating Software Effort using FOSS Data

- Background and context
- Lies, damn lies, statistics and estimates
- COCOMO
- SLOC
- 2GAD
- Results
- Call to action

In the absence of data...

- “...companies can save $250,000 more for each patch submitted upstream each year”

  (http://www.linuxfoundation.org/publications/workgroup/value-of-ltsi)

- “...it would take a team of 1,356 developers 30 years to recreate the code base present in The LF’s current collaborative projects”

  (https://www.linuxfoundation.org/sites/main/files/lfpub_cp_cost_estimate2015.pdf)

Lies, damn lies, statistics and estimates

In general

- '88.4% of statistics are made up on the spot'
- 'anyone predicting the future for a living is a charlatan' [1]
- 'the more confident the prediction, the more bull****' [2]

Specifically for us:

- Management asks for an estimate => doesn't like it => halves it.
- Engineer learns to double, treble the guess
- So #noestimates is unsurprising
- … but it doesn't help… we do actually need to understand how much effort it takes to produce software

[1] "The Black Swan", Nassim Taleb
[2] "Future Babble", Dan Gardner

Setting expectations for this talk...

- No crystal ball
- The 'state-of-the-art' is guesswork
  - most projects are late, over budget
  - most estimates (and even historic costs) are wrong
  - agile folks avoid estimates altogether

  THIS IS A HARD PROBLEM

- Can we even get to real costs for existing work?
  - people multi-task
  - companies don't share data
  - many don't even have the data
  - but there is lots of FOSS data

All I'm aiming for is...

- Something to sanity check the huge COCOMO numbers
- Something easy for engineers to compare and measure when asked for estimates

  '...we need to implement a Foo. A Foo is a bit like the FOSS projects Bar and Bibble. How much effort has been spent on those projects?'

  'we need to redo the Zub functionality. How much effort did it take last time?'
- Maybe managers can measure for themselves….

  and stop asking engineers for estimates.

What can we actually measure?

- SLOC
- Commits/patches/VCS history
- Timesheet hours (but not for FOSS)
- Contributors, start and end dates
- WTF per week

…. or what?

Maybe we don't need to measure...

- Let's use COCOMO
- Or just **mention** it…

  so folks think we used it

COCOMO

- Estimate 'Function Points' (WTF #1)
- Q: What is a Function Point, precisely?

  A: let's bung a set of parameters together in a formula and call it a standard. (at least 'planning poker' is by definition gambling)
- Calculate SLOC (#2) based on Function Points
- Calculate effort based on SLOC (2.4*(KSLOC**1.05)) (#3)

- Where is the proof COCOMO actually works?

Recommended reading: https://leanpub.com/leprechauns

SLOC

- Linux Foundation seem to use `wc -l` or some other magic..

  eg 2013 Linux kernel ~17 MLOC

  David A. Wheeler's SLOCCount at that time: ~ 12M

- So `wc -l` or magic gives a 40% over-estimate…

- OpenHub just flat out gets bigger numbers vs SLOCCount

… guess what would do to COCOMO

[NB I've not managed to get to the bottom of this so far]

SLOC #2

Some traps:

- blank lines + comments (SLOCCount handles this)
- verbose vs terse style
- auto-generated code
- imported/re-used library code
- duplicated code
- test code
- meta-data, config, specs, documentation?

SLOC #3

But maybe the deepest hole...

- if we re-work code to make it smaller…
- or remove code by adopting a library…
- it seems to have taken less effort!

Digression… grimoire

- http://bitergia.com take metrics much more seriously than I do
- Helpful community: #metrics-grimoire on freenode
- In-depth analysis of activity for OpenStack, Red Hat and others
- IIUC they have no simple solution for estimating total effort on projects, but see

http://gsyc.urjc.es/~grex/repro/2014-msr-effort/msr14-robles-estimating-effort.pdf

http://flosshub.org/system/files/What%20Does%20it%20Take%20to%20Develop.pdf

Digression… we don't think straight

“10 out of 11 projects display EFFORT that is less than the one predicted by COCOMO, overall suggesting higher productivity (as KLOC per contributor-month) in FLOSS than in closed-source.”

…. which I gently suggest is an example of a 'narrative fallacy' [1] – we instinctively find a narrative

[1] Recommended reading is “Thinking Fast, and Slow” by Daniel Kahneman

To be absolutely ****ing clear, KLOC per contributor month is NOT A MEASURE OF PRODUCTIVITY

And I'm calling COCOMO bullshit until someone proves otherwise with actual data.

New algorithm!

Team's total effort =

New algorithm!

Team's total effort = Team's git-active-days

New algorithm!

Team's total effort = Team's git-active-days

+ the entrails of a chicken

Git-active-days: GAD

- GAD: a day when an engineer makes any commits
- Sum all GAD per engineer

  (for all refs/branches, not just master)
- Add up the total for all engineers

Gitstats, git-summary etc already do this :)

Git-active-days could be too high...

- A commit doesn't take a day
- Engineer committing to multiple projects
- Engineer adds commits to boost the number (unlikely to apply to historical data)
- Merge-commits, bot-commits

Git-active-days could be too low...

- Some days engineer can't commit anything (complex change, spec, design, test, meetings)
- Maybe a culture of huge patches
- Commits squashed together
- Invalid time/tampering with history
- Multiple engineers committing as one user
- Code dumps

I think it all evens out...

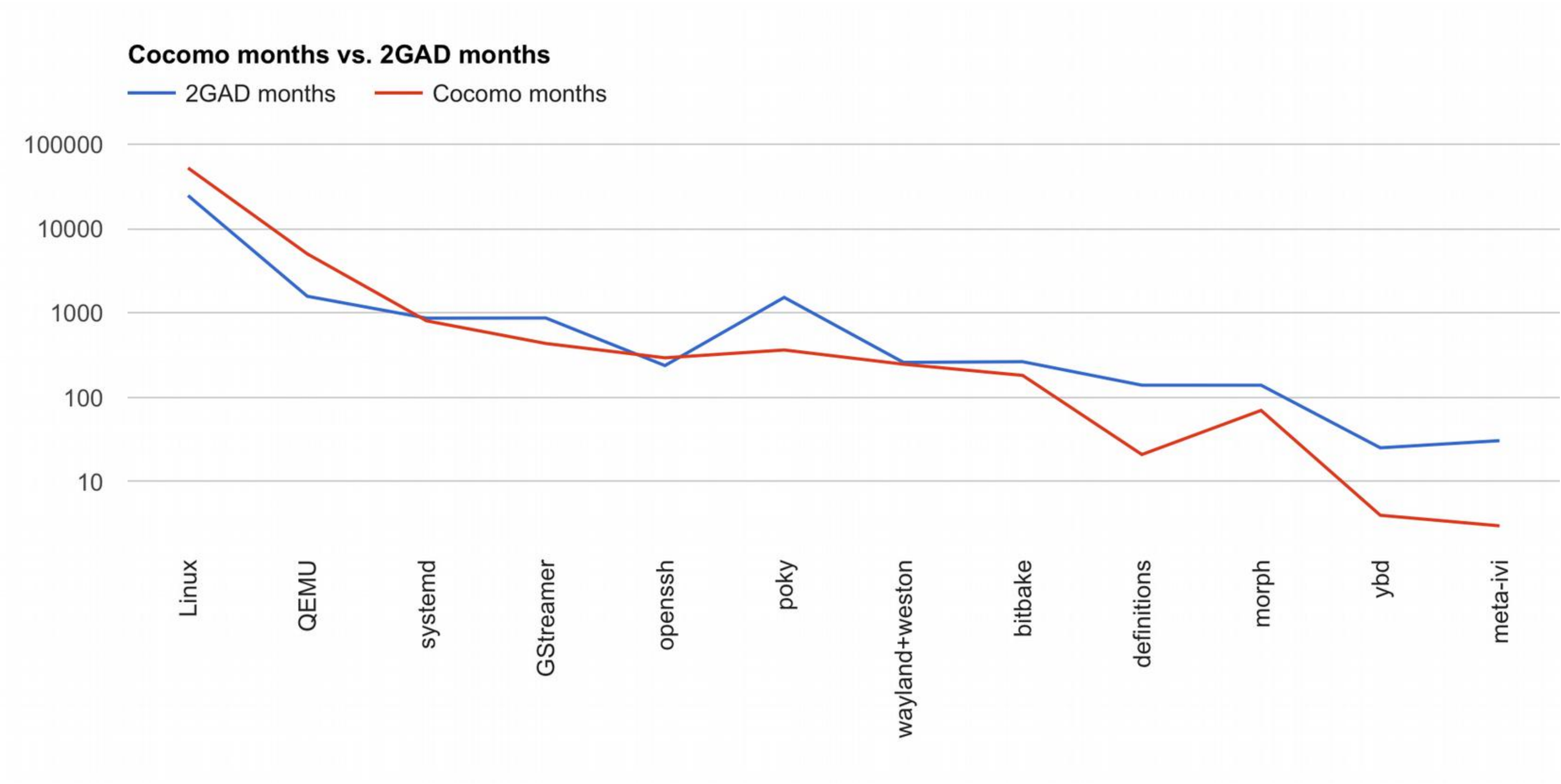
- And coding is only part of the project
- But it correlates with total effort

so my proposed algorithm is GAD * 2

2GAD

and I claim that my simple factor of 2 is more justifiable than all the parameters in COCOMO

Some results...



Some results...

For GENIVI Baseline…

link to GADmonths*2 vs COCOMO months

link to COCOMO months vs GADmonths*2

For a bigger set (including OpenStack systems)

COCOMO Basic/organic person-days vs. GAD

Differences: 2GAD vs COCOMO

- 2GAD doesn't work for code-dumps
- 2GAD gives smaller values on big projects
- 2GAD gives larger values on small projects

Advantages 2GAD vs COCOMO

- 2GAD is based on real data
- 2GAD is language + content independent
- 2GAD can be applied to
  - a set of repos
  - a date range
  - a subset of contributors eg * @ foo.bar
  - maintenance as well as development

Call to action

- Bring some rigour!
- Calculate 2GAD on your internal projects
- Compare to timesheets if possible
- Publish your results

Code I used is at

https://github.com/CodethinkLabs/research-git-active-days

Codethink