Managing a Large^[1] Scientific (Crystallographic^[2]) Software Project

"It's not the Algorithms"

[1] Non-trival, not small[2] I don't think it matters

Paul Emsley Aug 2011

Content

- Dependencies/Libraries
- Handling the updates
 - Source Code Repository
- Building
- Testing
- Progress
 - Mailing list
- Releases
- Deployment

Introduction to Coot.

- GNU GPL v3
- OpenGL (3D graphics) package for macromolecular model-building

i.e. making, adjusting and validating models of proteins

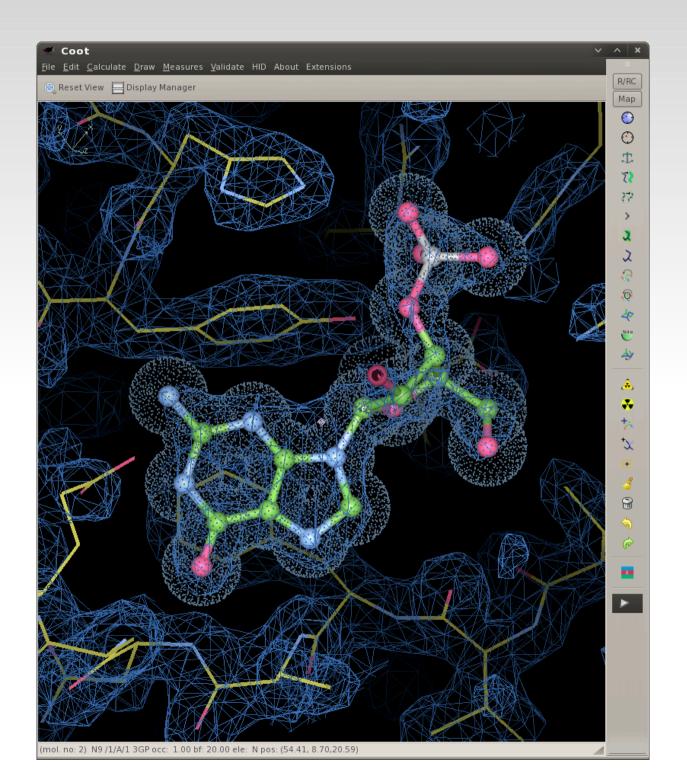
- At its peak it was 3rd most highly cited research paper in last 5 years
- Most highly cited Free Software

The GNU Heritage

- I have been a GNU fan since late '80s
- "If I have a chance to develop software, it will be based on GNU and GLPed"
 - 12 years later I got the chance
- Which meant GCC, G++, GDB, Emacs & the GNU build system

2004: The Marketplace

- Extremely entrenched user-base
 - 2 (perhaps 3) programs that were always used
 - Once chosen & learnt, users didn't swap
- Which mean that the program had to be
 - Easy to get/Freely available
 - Easy to install
 - With compelling functionality
 - Communicate with other programs
 - and "Web Services"



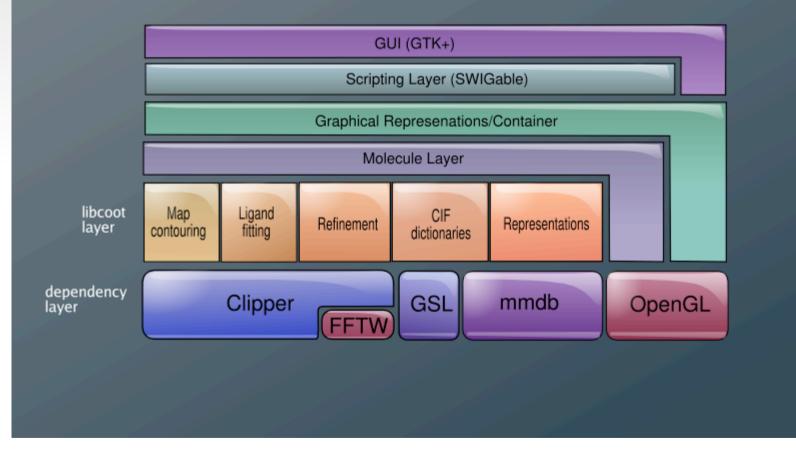
Technologies

226 package dependencies

• Core:

- C++, C, Scheme (Guile), Python
- Crystallographic Libraries:
 - CCP4 libs, mmdb, Clipper
- Graphics:
 - GTK+, glade, gnomecanvas, OpenGL
- Building:
 - Subversion, automake & autoconf, emacs,
- Documentation:
 - Texinfo, Doxygen, XHTML, Wordpress, Apache

Schematic Coot Architecture



Developing: Early on...

- I used to work on just one computer
 - Making a tar file every few days.
 - (This made backtracking painful)
- I discovered SVN and that made a big difference
 - +30-40% more productive
 - More than one computer
 - Cross-platform issues

PE: testing experiences

- Kevin (2001): "You have a methodological problem"
 - "if you can't test your code you're developing wrongly"
- From 0.0 to 0.3.3 time spent was mostly bug fixing
 - 90% of time fixing 3-yr-old code
 - Frustrating
- Pre-release testing (by me)
 - conflict of interest
 - I was "hitting the buffers" on the release schedule
 - I *really* didn't want to find bugs!

PE: testing experiences

- At CCP4 meeting: Peter Briggs was talking about problems releasing 5.0
 - Me: "You must have a extensive and fullyfeatured test suite!"
 - Peter: "Hmm"
 - Harry (to PE): "Have you got one?"
 - Me: "Err, no"
 - Harry: "Shut up, then"

Testing in CCTBX

- Ralf and Peter discussed how testing in CCTBX worked.
- I was impressed at the rigour
 - Bring some of that to Coot

Previously: Test/Exercising Programs

- Generated test data for classes/objects
 - Used for development
 - Had hard-coded test values
 - the "did it work?" test was something that I determined "by eye" at/after run-time
 - Not automated
 - Not part of test suite

Other Testers

- No-one formally involved
- Student (at Workshops)
- And other ad-hoc users
- Challenge is to convert problems that they report/demonstrate into an automatic test

Practical Aspects of Test Suite

- Written in a different language
 - it is difficult to write tests with compensating errors
 - If the scripting language is embedded, it can return variables, rather than text in a log file that has to be parsed and converted to variables
- I use greg (gnu regression test suite) because guile is embedded in Coot
- We also use PyUnit

An Example

- "Can I have a "sharpening tool?"
- Yes (I thought), but I will need to store the data before converting it to a map – make it part of the molecule class:
- clipper::HKL_data<clipper::datatypes::F_phi<float> > original_fphis
- I implemented the sharpening template
- I merged in changes from Bernhard
- I ran the tests
- Crash!
- Obscure bug, tickled only after using 60+ molecules, which may not have otherwise been noticed for years

"How can I test this?"

- A new way of working for me:
 - First I export a function to the scripting layer that allows access to new function
 - then sketch out the new class and functions interface (skeleton functions)
 - Now write a test that exercises new functions
 - At first it fails Of course!
 - But eventually the test passes
 - Very gratifying
 - And test moves from devel to main testing suite

Another Example

- Many functions in Coot take a molecule-number argument
 - part of the test suite should be to exercise functions given "bad" molecule numbers
 - ~95% of time there is no problem (tested twice)
 - is it worth it for that extra ~small %age of time?
 - A crash in an interactive program on bad input is bad
 - A crash in a "batch" program on bad input is not so bad

Recent Example

 "When I mutate an adenine to a thymine, the C7 disappears"

Recent example

(greg-testcase "Mutate to a DT keeps C7" #t

```
(lambda ()
```

```
(let ((imol (ideal-nucleic-acid "DNA" "B" 1 "gatc")))
```

(if (not (valid-model-molecule? imol))

```
(throw 'fail))
```

```
(mutate-base imol "A" 2 "" "DT")
```

(list? (get-atom imol "A" 2 "" " C7 ")))))

Something I missed

- Recall that OpenGL is not immediately scriptable
 - Click on an atom and in the console it reported atom specs of atoms[0], not atoms[atom_index]
 - Very difficult to catch (or even notice)

Is it worth it?

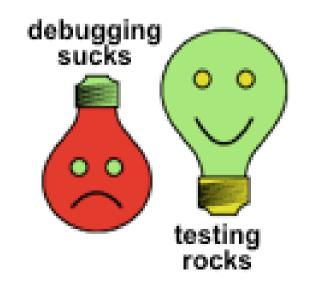
- Previously: 90% of time for the 2 or 3 years spend debugging old code
- Writing tests takes
 - +50% to +100% longer
- For me, I now spend most of my time with new code, not old code
 - That is enjoyable
 - Having the executables have to pass a test suite is an enormous feeling of security

Productivity

- Subversion and testing
 - Security
 - Liberation!
 - Freedom from paralysis

Which Framework?

- I use greg for Scheme and PyUnit for Python
- CCP4 now uses Python and PyUnit



Code Complexity

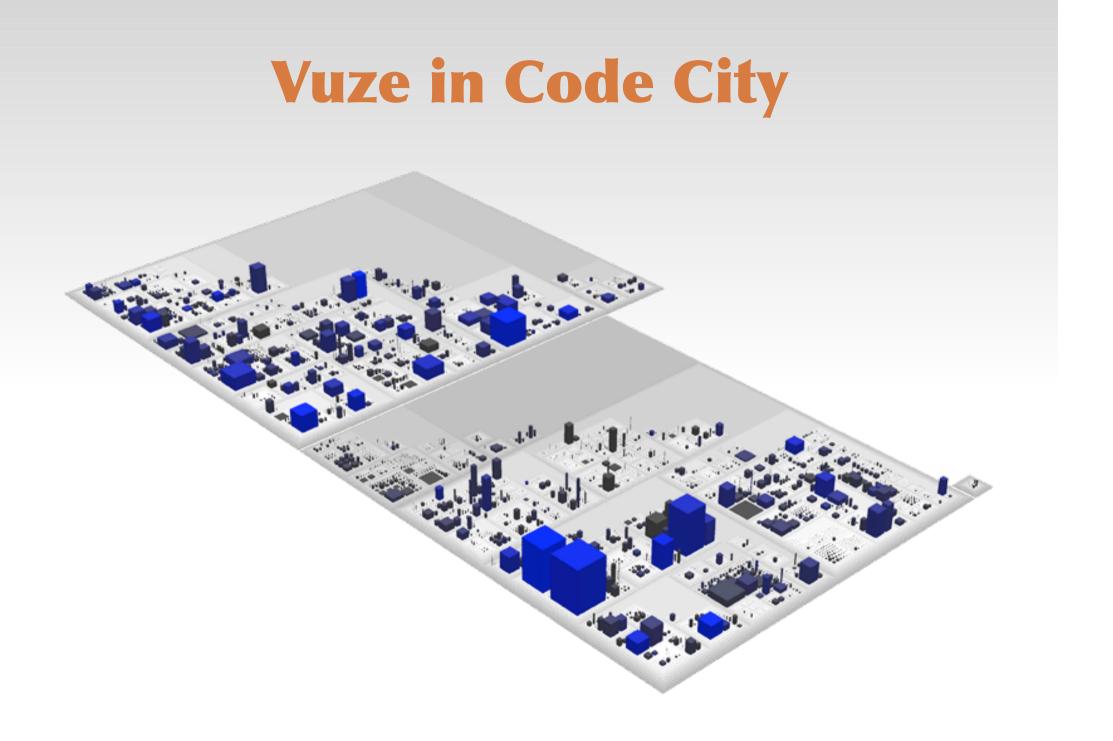
- Cyclomatic Code Complexity
 - A measure of code branches
- Correlated with the number of lines of code in a function
 - and/or the number of "if" statements
- Studies show strong correlation of code complexity with defect rate
- "metrics" plug-in for Eclipse
- Simple Rule: if it's longer than 2 pages
 - Worry...

Visualisation of Code

Code City

10





Hosting Issues

- The mailing list and SVN were hosted at the University
 - Burden on our sys. adm., who had other things to do
 - Then the server was hacked, new computer brought in but I was not allowed to run SVN there
 - So

A Wiki too

2.3Gb/month

- SVN hosting moved to Google
- Mailing list moved to JISCmail
 - 600+ subscribers

Build Status

- Many machines
- Dispersed
 - Some I can use the file system directly and some files are accessed over the web
- Status is polled every 10 minutes
 - guile script
- Can access the age of the latest successful build and the build logs (in case of failure)

Build Status

Coot SVN and Build Summary

Generated Sat 04 Apr 10:40:03 2009 BST

SVN Repository Revision: 1946 svn log

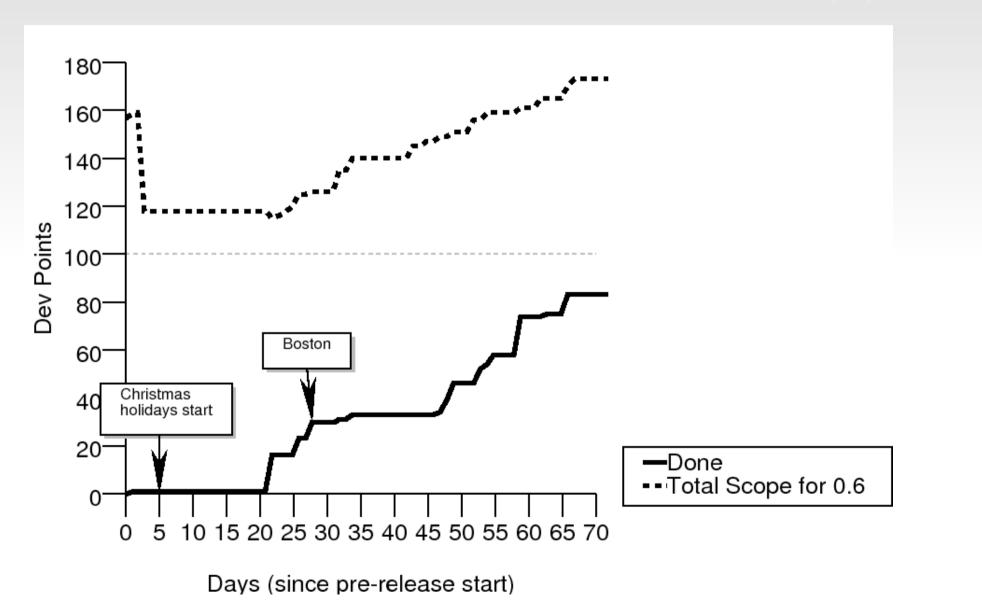
Source code <u>coot-0.6-pre-1-revision-1946</u> 1946 19 hours 25 minutes source-build pass Sat Apr 4 10:34:41 BST 2009 <u>source build log</u>

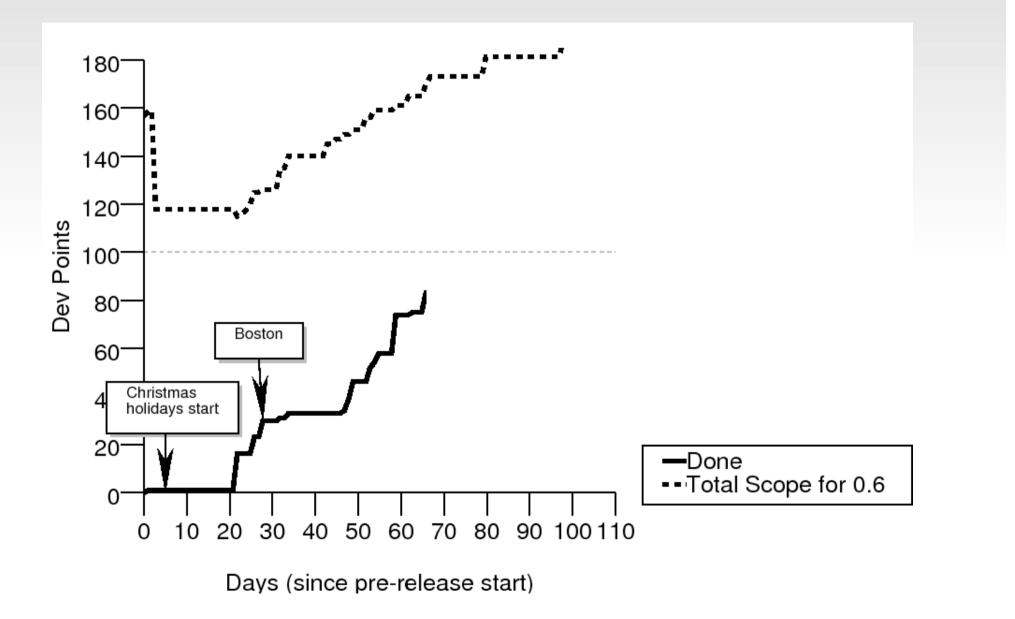
binary-Linux -i386-fedora-3 1945	binary-Linux -i386-fedora-3-python 1945
pass-build pass-second-test <u>build-log build-dir test-log</u>	pass-build pass-second-test build-log build-dir test-log python-test-log
1 day 8 hours 33 minutes	1 day 12 hours 9 minutes
binary-Linux -i396-fedora-4-python-qtk2 1946	binary-Linux-i396-fedora-8-python-gtk2 1945
pase build pase second test build-log build-dir test-log python-test-log	pass-build pass-test build-log build-dir test-log python-test-log
4 hours 4 minutes	1 day 8 hours 52 minutes
binary-Linux-i386-fedora-10-python-gtk2 1946	binary-Linux-i386-redhat-8.0 1946
pass-build pass-test <u>build-log build-dir test-log python-test-log</u>	pass-build pass-second-test <u>build-log build-dir test-log</u>
8 hours 37 minutes	5 hours 11 minutes
binary-Linux -i386-redhat-8.0-python 1946	binary-Linux -x 86 64-rhel-4-gtk2 1944
pass-build pass-second-test build-log build-dir test-log python-test-log	missing-file fail-build <u>build-log build-dir</u> test-log python-test-log
3 hours 43 minutes	2 days 2 hours 23 minutes
binary-Linux -x86 64-rhel-4-python-gtk2 1946	binary-Linux-i386-centos-4-gtk2 1929
missing-file fail-build build-log build-dir test-log python-test-log	pass-build pass-test <u>build-log build-dir test-log python-test-log</u>
binary-Linux-i386-centos-4-python-qtk2 1930 pass-build pass-test <u>build-log build-dir test-log python-test-log</u> 16 days 22 hours 26 minutes	binary-Linux + 2 minutes binary-Linux + 2 86 64-centos-5-gtk2 1946 pass-build pass-second-test build-log build-dir test-log python-test-log 2 hours 59 minutes
binary-Linux -x 86 64-centos-5-python-gtk2 1946	binary-Linux-i686-ubuntu-6.06.1-python-qtk2 1946
pass-build pass-second-test <u>build-log build-dir test-log python-test-log</u>	pass-build pass-test <u>build-log build-dir test-log python-test-log</u>
18 hours 18 minutes	9 hours 15 minutes
binary-Linux -i686-ubuntu-8.04.2 1946	binary-Linux -i686-ubuntu-8.04.2-python 1946
pass build pass second test <u>build-log build-dir test-log</u>	pass build pass second test <u>build-log build-dir test-log python-test-log</u>
1 hour 24 minutes	6 hours 22 minutes

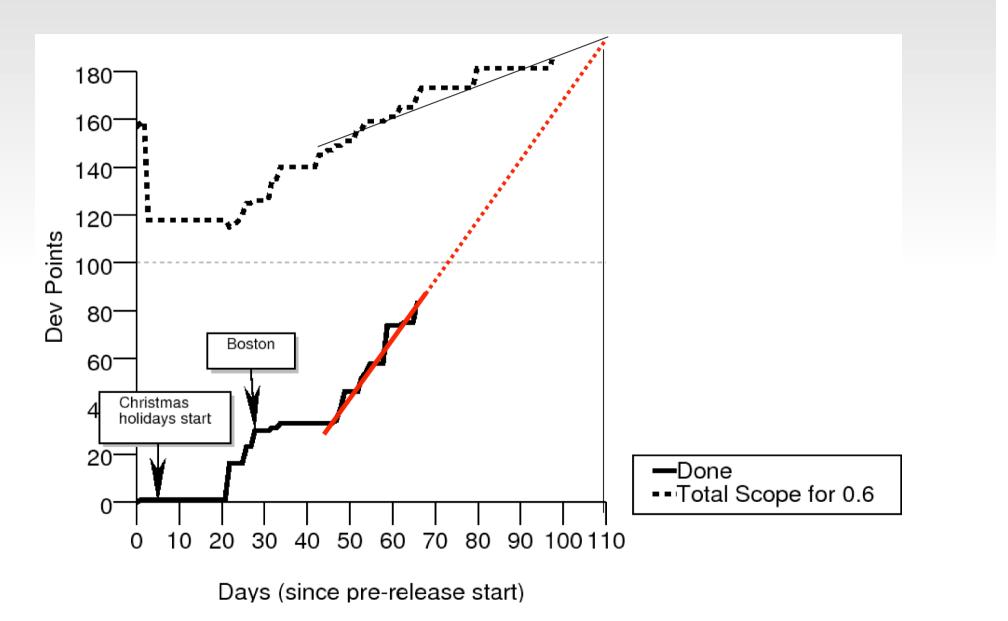
Gathering Specs

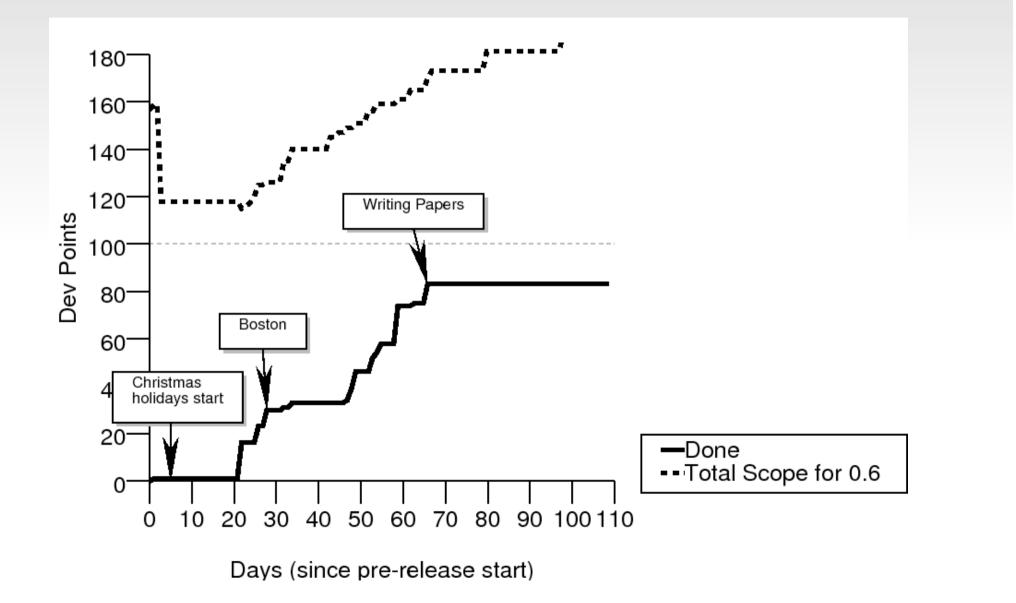
- Lots of email,
 - personal and via mailing list
 - I travel a lot and do presentations
- Thus specs are gathered
 - I select those that the users most require and estimate how long to implement
- I choose enough for ~2 months (full-on) work
 - Each feature taking n "half-days"

Made by PyChart









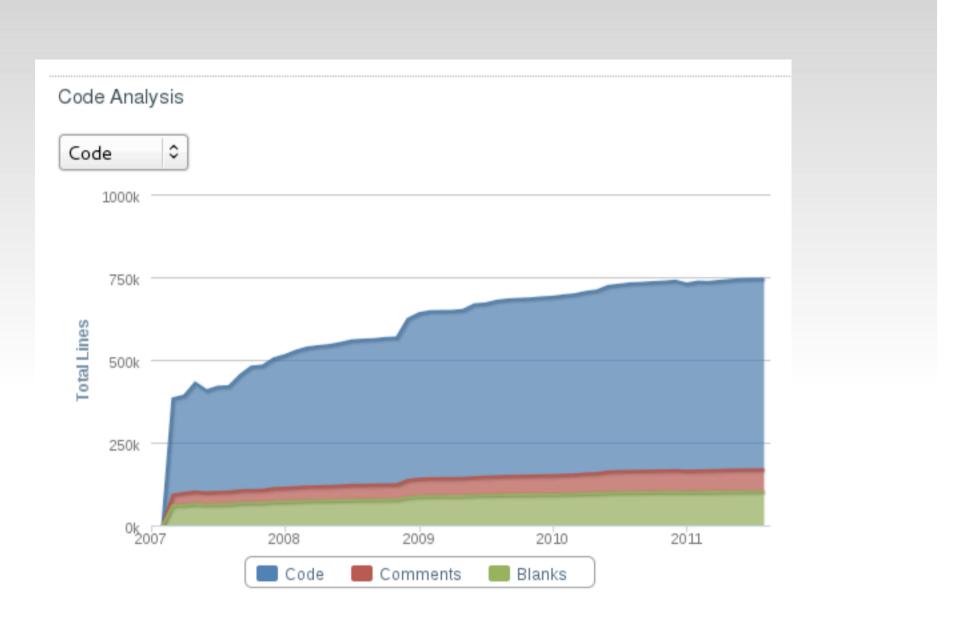
- Project managing:
 - Either define the release date...
 - or the spec
 - NOT both!
 - (That's a recipe for a "death march")

Deployment

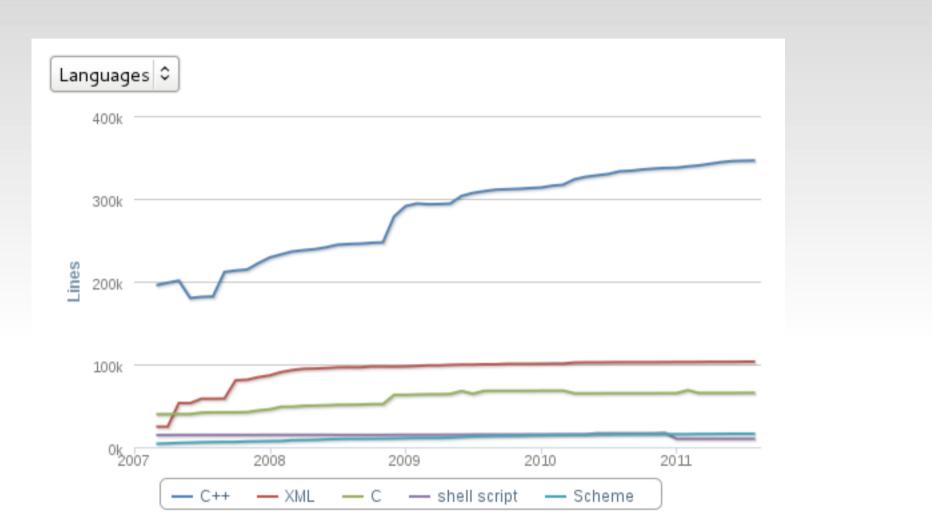
- Integration on a hourly/daily basic (SVN)
- Daily deployment
 - multiple platforms
- Bug fixes, feature requests available quickly
 - real communication with "expert users"
 - take customer feedback seriously
- Problem of "monolithic release" goes away
 - because the program is "always deployable"

coot

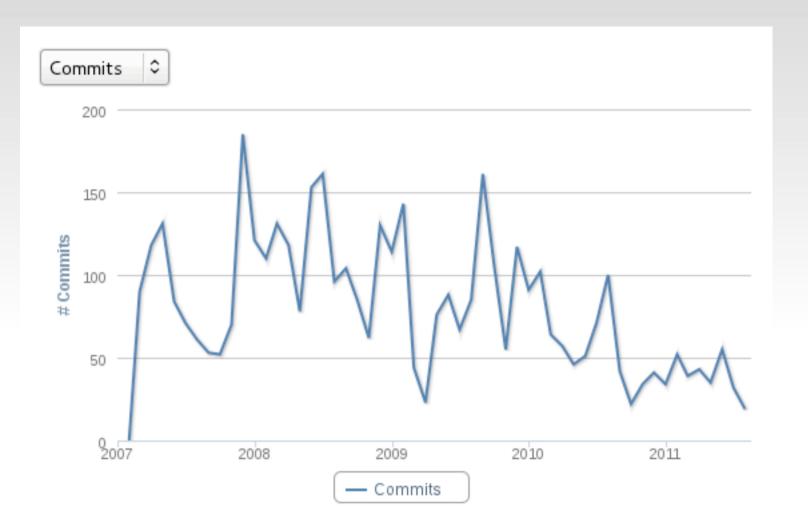




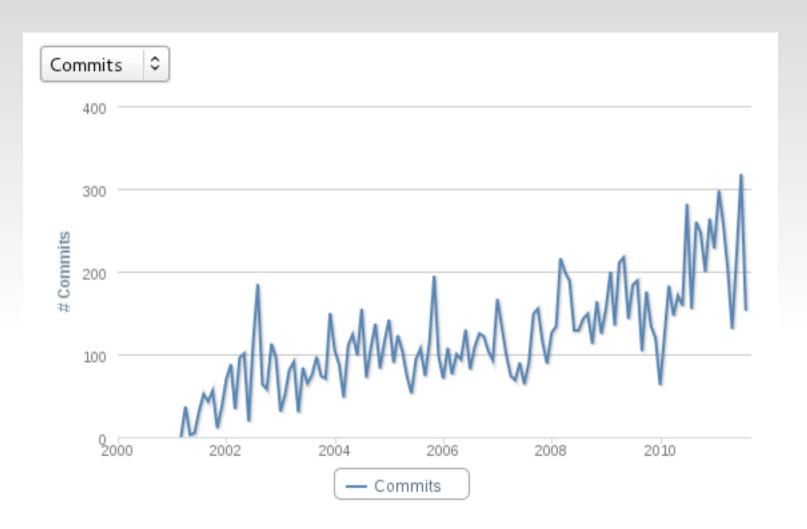
Lines of Code



Languages



Coot repo commits/month



CCTBX repo commits/month

Productivity Gains

- Not writing papers
- Not going to meetings
- Not handling the Mac or WinCoot builds
- Using Scheme and Python
 - Especially for the GUI
- Emacs
- Subversion
- Testing/test suite
- Working together

Summary/Take Home...

- Use SVN (or bzr) or something similar
- Writing new code is much more fun than debugging old code
 - So: Be disciplined: Test-driven development
 - Use Standard Template Library
 - Writing extension language code (scheme/python) is much more fun than C++
 - Spec features, not release and NOT both

"Release Early, Release Often"

- This is ridiculous
 - (in our field)
- Should be:
 - "Release when it's 'done done', release often"