

Approaching Open Source Science: Tools, Approaches

C. Titus Brown / ctb@msu.edu

Rough ToC

1. Sucking people into your open source science project: tool & development choices.
2. Automated testing: what it is, and what kinds you should use.

Talk is at <http://idyll.org/~t/swc.pptx>

Should you do “open source science”?

Ideological reason: reproducibility and open communication are (or should be) at the foundation of Good Science.

Idealistic reason: it's harder to change the world if you are trying to

- (a) Do good science
- (b) Keep your methods secret

Really practical reason: maybe other people will *help!*

So, your science software is open source... what now?

I should briefly mention that there is a largely
correct viewpoint that “closed-source science”
is oxymoronic.

;)

So, your science software is
open source... what now?

So, your science software is open source... what now?

Licensing is usually not your choice.

I've slowly shifted to preferring BSD over GPL, but many universities like GPL because they can dual-license & restrict for-profit resale.

(It's usually virtually impossible to get anyone to use your project even if you pay 'em, so why put restrictions on it!?)

So, your science software is
open source... what now?

How do you attract users??

Solving Social Problems with Technology

(That's a joke. It's generally held that you cannot.)

Free project hosting

Every open source project should have:

- A place to get the latest release.
- A mailing list.
- An openly accessible version control system.

A wiki and an issue tracker are useful if you have time/manpower. You don't.

Free project hosting

SourceForge, Google Code, github, bitbucket, etc. can provide you with downloads and source control. SourceForge and Google Groups give you mailing lists.

Primary consideration: distributed version control, or not?

Version control

(You all use version control, right?)

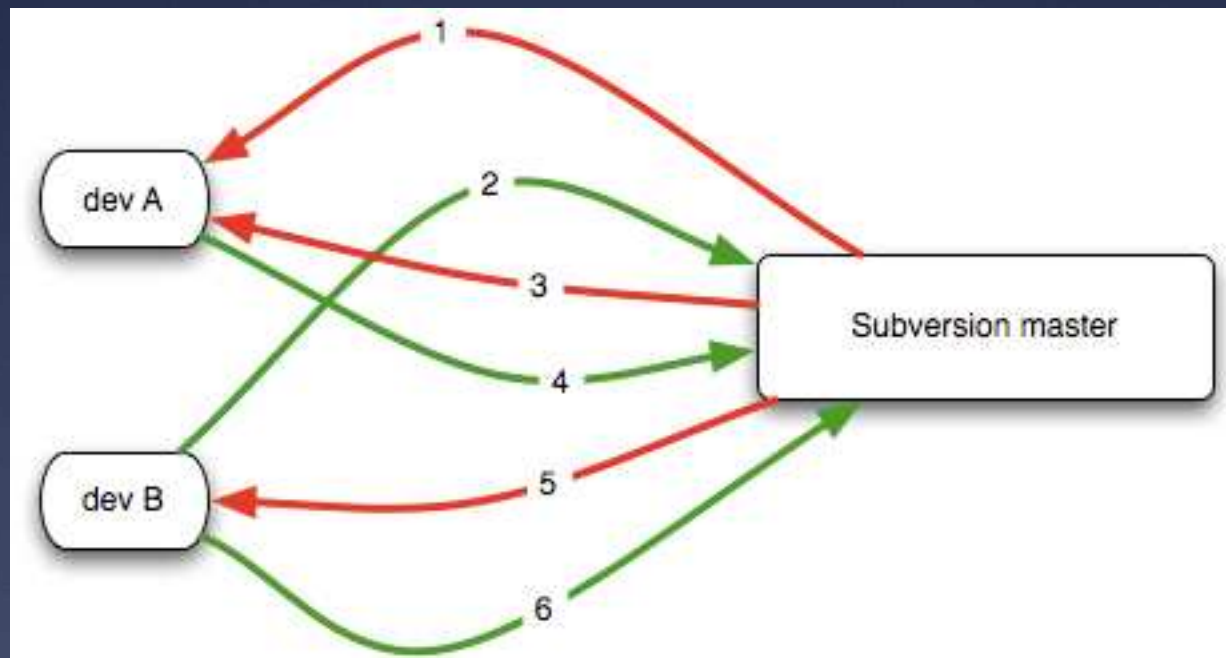
Version control

(You all use version control, right?)

Distributed version control (mercurial, git, etc.) is
awesome and life-changing.

Version control - subversion

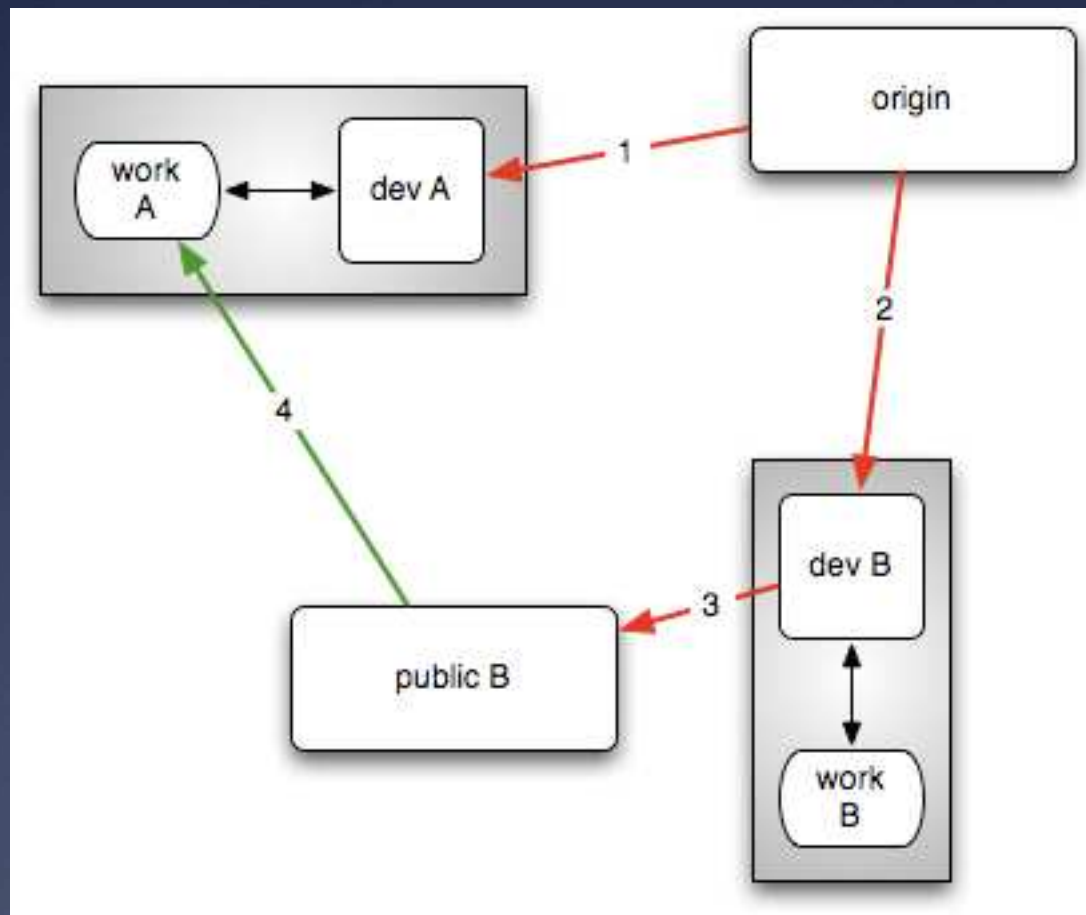
Centralized



All commits go through master.

Version control – git or hg

Decentralized



Distributed version control

Essentially decouples developers from master.

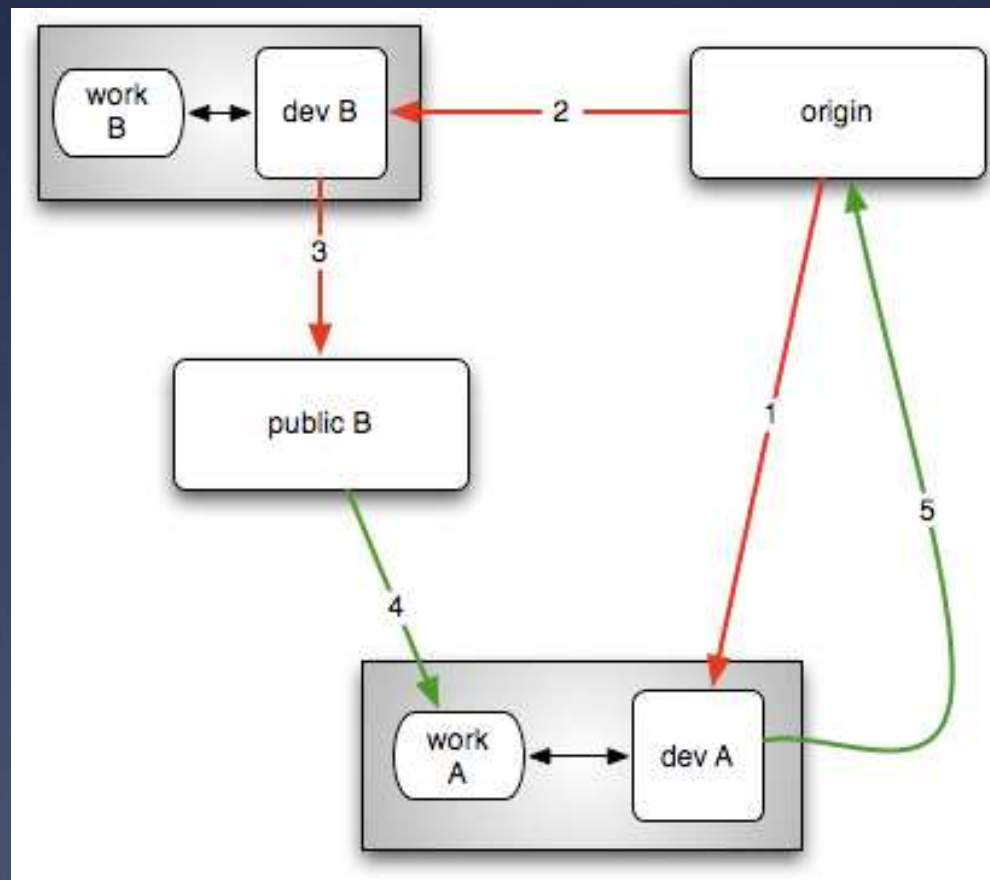
Mixed blessing – “code bombs”, effective forks of the project, complicated merges.

Frees you from permission decisions: “do what you want. If I like it, I’ll merge it.”

Github, gitorious, bitbucket, SourceForge and (?)
Google Code all support DVCS.

Distributed version control

Process can still be under central control...

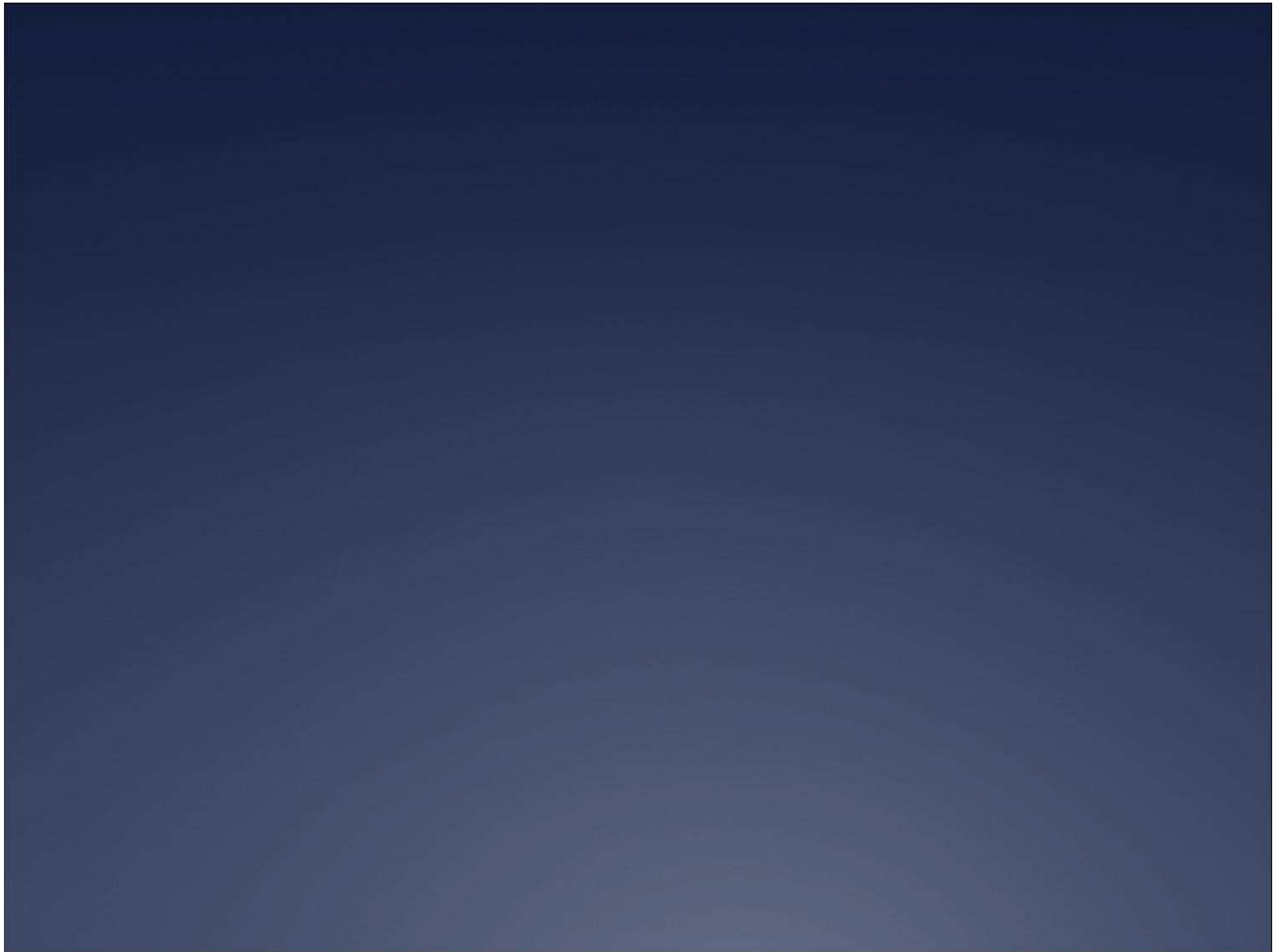


“Open source” vs “open development”

Do you want participation?

Participation comes at a cost, in both support time and attitude.

- Loss of control.
- Annoying questions about design decisions.
- Frank (“insulting”) discussion of bugs.
- Time-consuming code contributions.



The Stunning Realization

An anecdote:

“There’s no such thing as too many
corrections”

The Stunning Realization

How do we know that our code works?

The Stunning Realization

How do we know that our code works?

And what does “works” mean, anyway?

The Stunning Realization

We don't teach young scientists how to think about software.

We don't teach them to be suspicious of their code.

We don't teach them good thought patterns, techniques, or processes.

We basically shove them in front of {Fortran,C,Perl,Python} and tell them to generate results.

The Stunning Realization

We don't teach young scientists how to think about software.

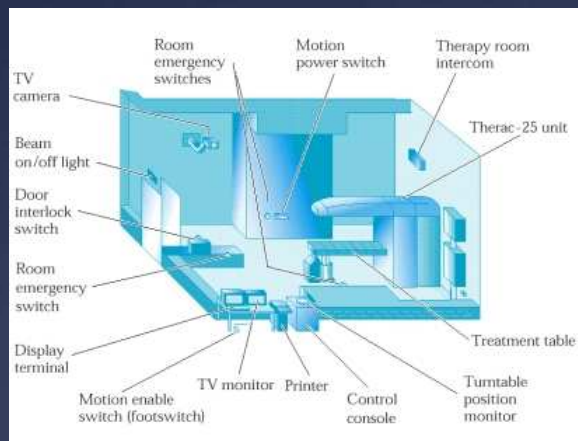
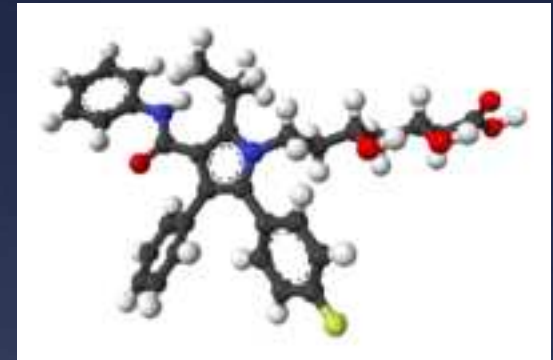
We don't teach them to be suspicious of their code.

We don't teach them good thought patterns, techniques, or processes.

(Actually, CS folk don't teach programmers this, either.)

Fear does not seem to be a sufficient motivator

Pfizer backtracks on the benefit of atorvastatin over simvastatin: “Programming error” in flawed study cited [June 2007, WebMD]



Software glitch in Therac-25 responsible for patients receiving lethal radiation dosages [Boston Globe, November 1987]

A Sign, a Flipped Structure, and a Scientific Flameout of Epic Proportions

(lack of software testing leads to retraction of 3 Science, 1 Nature, and 1 PNAS article...)

Interlude

You all use version control, right?

And (if you're using a compiled language)
you have a build script, right?

And other people (e.g. fellow student) can
build & use your software, right?

If so, good.

Else, go straight to jail, do not pass go.

Automated testing

The basics of automated testing:

Write 'test' code that runs your 'main' code and verifies behavior.

Automated testing

Example: regression test.

1. Run program with a set of parameters & record output.
2. At some later time, run the program with the same set of parameters, compare output to recorded output.

Answers question: “did my program change?”

Automated testing

Example: regression test.

Answers question: “did my program change?”

Different from “is my program correct”, but if your results change unintentionally, maybe you should ask why?

(Synergy with version control)

Automated testing

Example: functional test.

1. Read in known data.
2. Check that known data matches your expectations.

Answers question, “does my data loading routine work?”

Automated testing

Example: functional test.

Answers question, “does my data loading routine work?”

If your data loading routine doesn't work, you should probably fix it.

Be brave; test with “tricky” data. Use testing to increase your confidence.

Automated testing

Example: assertion.

1. Put “assert parameter ≥ 0 ” in your code.
2. Run your code.

Answers question, “do I ever pass garbage into this function?”

Automated testing

Example: assertion.

Answers question, “do I ever pass garbage into this function?”

You’ll be surprised at how often conditions that “should never happen”, do happen.

(#1 trick. Walk away with this in your bag o’ tricks, & you’ve learned something.)

Automated testing

Example: assertion.

C: `assert(condition);`

Python: `assert condition, message`

Perl: `assert(condition);` or `'die unless'`.

There's no excuse – `assert` is in your language.

Automated testing

There are several other kinds of automated testing (acceptance testing, GUI testing) but they usually don't apply to scientists.

Automated testing

You don't need to use specialized testing tools, honestly.

Regression tests: 'diff' and shell scripts.

Functional tests: write a specialized entry point.

Put 'assert's in your normal code.

Automated testing

You don't need to use specialized testing tools, honestly.

OK, I lied.

Automated testing

Code coverage: what lines of code are executed?

- Discover dead code branches.
- Guide test writing to untested portions of code.

Need a specialized tool, per-language.

Automated testing

Continuous integration

Have several “build clients” building your software, running tests, & reporting back.

- Does my build & run on Windows?
- Does my code run under Python 2.4? Debian 3.0? MySQL 4?

Answers question, “is there a chance in hell anyone else **could** use my code?”

Automated testing

So, why does automated testing help with
“correctness”?

It locks down “boring” code

(== code that you understand)

and lets you focus on “interesting” code.

Automated testing

So, why does automated testing help with
“correctness”?

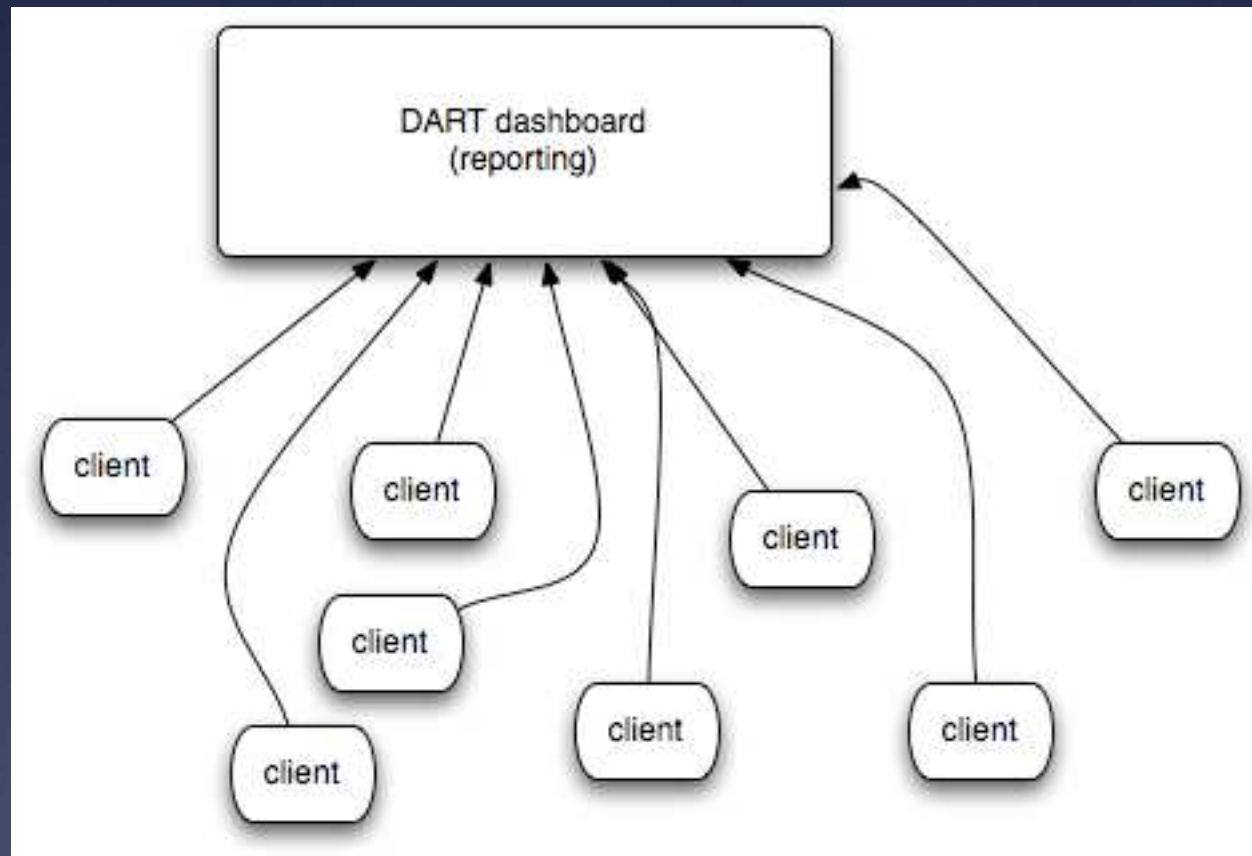
1. Freedom to refactor/tinker/modify.
2. Maintenance.
3. Other people can also refactor/tinker/modify.

Distributing “continuous integration”

Take continuous integration mentality one step further – why not automate *user reports*?

Distributing “continuous integration”

“Dashboard” concept.



Many “anonymous” clients reporting to central server.

The Insight Journal

A neat idea.

VTK/ITK-based software has a common tool stack:
cmake&ctest.

The Insight Journal takes such submissions, builds them, runs the tests, and makes test & code review part of the review process.

It's almost like they care!

Concluding thoughts

- * If sucking people into your open source project is important to you (and it should be), then you can help make that happen by choosing technology.
- * An overlapping and complementary goal should be to “write correct software.”
- * Automated testing can help with that, too.

Acknowledgements

- * GrigGheorghiu
- * AlexandreGouillard
- * Greg Wilson

Questions?

