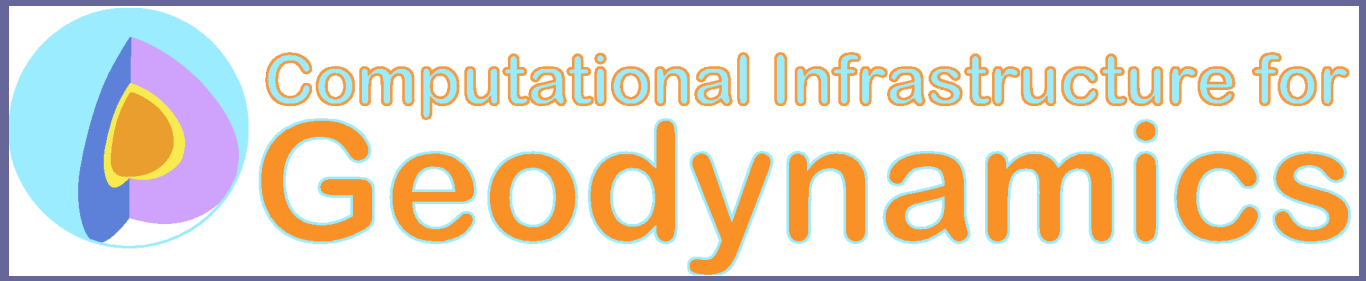
How to Give Away Scientific Software

Walter Landry



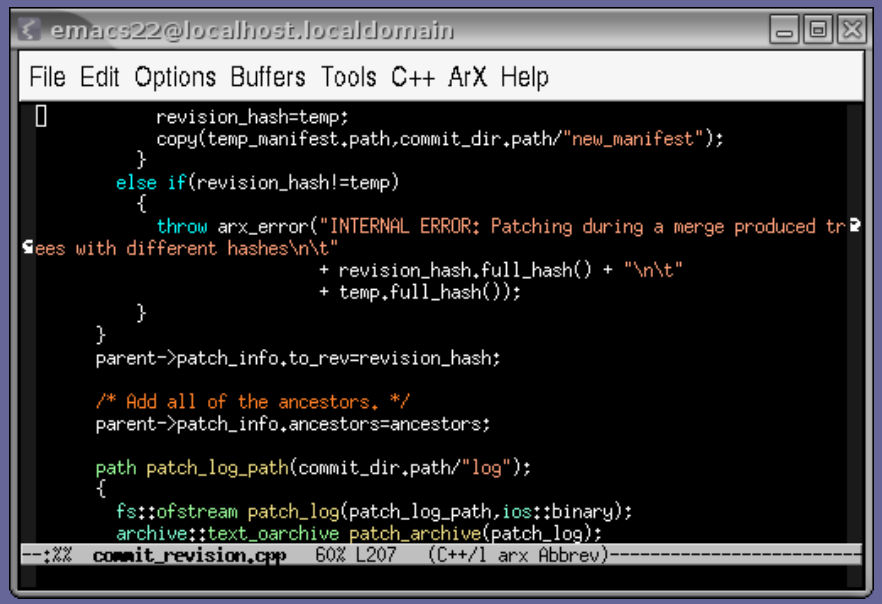
geodynamics.org

My Experience

- I have involved with distributing scientific and other software for about ten years
- I have been with the Computational Infrastructure for Geodynamics (CIG) for three years. CIG is an NSF funded center for developing, distributing, and supporting free software for the earth sciences.

Why give it away?

- There is more to life than just papers.
- The code used to generate those papers can be useful by itself.



Why give it away?

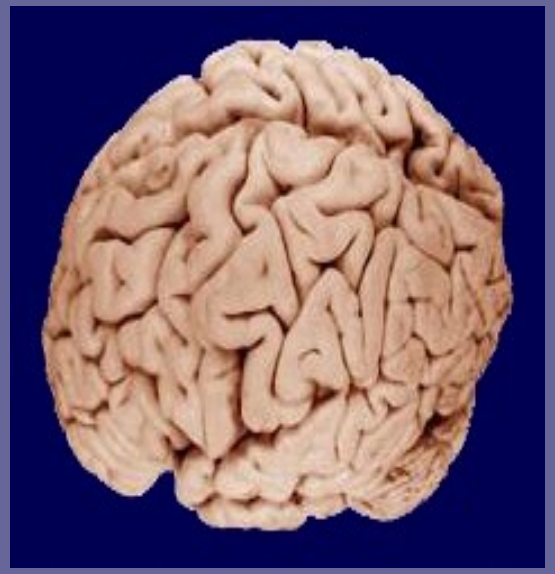
- You want to advance the state of the public art. Lots of people use bad algorithms to write bad papers.

Why give it away?

- Your funding agency requires it.
- Some agencies look more favorably upon proposals that promise to release their code.

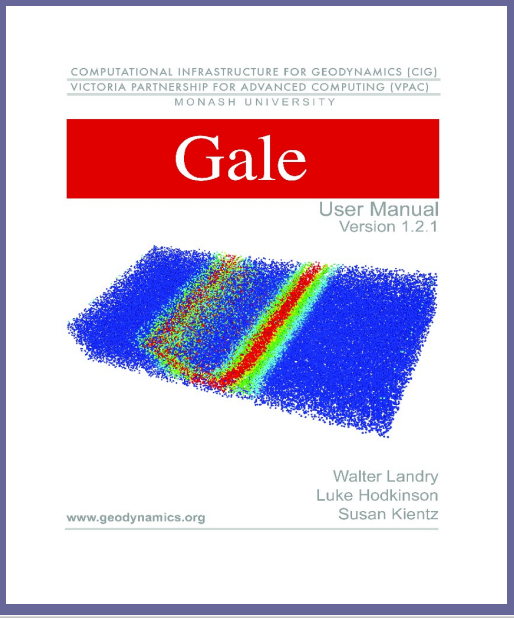
Why give it away?

- You definitely do not employ all of the smartest people in the world.
- You may hope to get other people to contribute to your project, thus helping you out.



Why doesn't everyone do it?

- It can be a lot of work
  - Writing documentation
  - Making it easier to use
  - Answering questions
  - Getting all of your licenses in order
  - Going to meetings



Why doesn't everyone do it

- You probably will not get that many contributions. You are not writing something as generally useful as the Linux kernel or a web browser.
- For example, Perl has perhaps a half dozen active contributors. Python even fewer.
- CIG's main codes get hundreds of downloads, but perhaps 5-10 actual users.
- So expect to do most of the work yourself.

Why doesn't everyone do it?

- Competitors can use your code and leapfrog you
  - This is often a more theoretical than practical problem. Understanding other people's code is hard.
  - However, I have seen people deliberately remove features prior to release. Presumably they were more motivated by  than 

The Steps to Releasing Code

- Development
- Promotion
- Support

It can be as little or as much work as you want. However, if you don't do much, don't expect many people to use the code.

Development

- In general, when developing the code, you need to:
  - Get the licensing in order
  - Get the code in order
  - Get the documentation in order

Licensing

- This is the part that everyone hates.
- It can easily be the most time consuming part of the whole process

License Zoo

- We have to start by considering the panoply of licenses in use.
- There are many, many, many different licenses that claim to be “open source.”
- Most of them are a pile of poo.
- Unfortunately, you can find many of them at opensource.org :(

License Zoo

- There are really only three licenses that you should think about
  - BSD/MIT/Public Domain
    - Do whatever you want (more or less)
  - GNU LGPL
    - Designed for libraries
    - You can link it with any code, but if you modify the library, you have to provide the modifications to the library.
    - This allows commercial codes to at least use the code without revealing their own secrets.

License Zoo

- GNU GPL
  - If you link the code against something else, then you have to provide the sources for that other thing
  - Effectively cuts off commercialization of the code, because anyone who gets the binaries also gets the source for everything AND is allowed to redistribute it to anyone.

License Zoo

- There are a few other decent licenses, but they accomplish basically the same thing with different words
  - Apache v2.0
  - IBM's CPL
  - CeCILL

Which License for You?

- The internet is filled with reasoned and unreasoned arguments over the merits of BSD vs GPL.
- Although any of the three licenses are OK, I recommend that you put code under the GPL.

Which License for You?

- There was a recent incompatible update of the GPL and LGPL to version 3. To maintain compatibility, I recommend licensing using the standardized terms

  This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Get the Licensing in Order

- First off, who owns the copyright for the code?
  - You may have to find all of the authors
- For each author, how were they employed?
  - If they worked directly for the US government (e.g. USGS, not a contractor), then the code is public domain. Easy!
  - They may still have to ask for permission (e.g. they work at a national lab).

Get the Licensing in Order

- If the author worked for a university, the university *probably* owns it.
  - Sometimes authors make separate deals with the university, so ask them first.
  - Sometimes they have not made a separate deal, but they still want to stop you.
- So then you get to figure out what the university's policy is.

Get the Licensing in Order

- Some universities are quite nice about it.
  - Rensselaer officially sanctions the MIT license.
  - UT Austin officially sanctions the GPL.

Get the Licensing in Order

- Other universities seem to be more in the crowd than the  crowd.
- They like to stick “no commercial use” clauses in their licenses.
- This causes incompatibilities with other licenses, hindering development

iGeos Debacle

- Because of the no commercial use and other clauses, they have to have developers licenses for their GUI toolkit Qt
- So anyone who wants to contribute to the code has to pay about $5000.
- If it were GPL'd it would be free.

Eos, Vol. 89, No. 29, 15 July 2008

Open-Source Software Framework Integrates Data Analysis

PAGES 261–262

An open-source software framework suitable for integrating many research tasks could be advantageous in various areas of geophysics. Two examples of the most successful and broadly used of such frameworks are the Seismic Un*x, which is used in seismic reflection data processing (SU) [*Stockwell*, 1999], and Generic Mapping Tools (GMT), used to produce various map projections in PostScript [*Wessel and Smith*, 1995]. Such large, shared projects bring clear benefits of versatile and uniform software support to the research community yet require decades of consistent development effort. However, SU and GMT are still limited to their respective application areas, and their interactive visualization capabilities are limited.

This report outlines our open-source software framework [*Morozov and Smithson*, 1997; *Chubak and Morozov*, 2006], which is called IGeoS (Integrated Geoscience, formerly referred to as SIA) and which has been in development for more than 12 years. Started as a seismic (reflection and wide-angle) processing package, the system uses dynamically linked custom binary executable codes for each task, similar to commercial seismic processing packages and unlike SU. Very few restrictions are imposed on the system's data types (seismic, borehole logs, gravity, Earth models, and so forth) and processing logic, which results in broad functionality. The system has been used in a variety of tasks, including signal processing in earthquake and wide-angle refraction; reflection seismology; seismic ray tracing, tomography, and migration; potential fields; visualization; interactive modeling and inversion; real-time data management; and Web services (such as data processing and delivery via the Internet).

The system now has reached substantial maturity and includes more than 200 plug-in tools, a full-featured graphical user interface (Figure 1a), support for parallel computations, and a 3-D/2-D visualization component. The system incorporates seamless interfaces to SU, GMT, and other software, and it includes a global seismic travel time calculator, 2-D ray-tracing programs, and seismic waveform modeling in one, two, and three dimensions. The IGeoS design framework is also convenient for developing new geophysical applications, with built-in tools for community collaboration and automatic code updates. Because of the system's object-oriented design, the incorporation of new and existing (including legacy) algorithms is typically easy, with benefits including common parameterization, input/output, parallelization, integration with other tools, and uniform code maintenance.

Integration of 3-D/2-D Visualization With Data Analysis

Visualization is the key to modern interpretation; however, interactive visualization also requires programming skills and efforts that are rarely available in focused research projects. Different data types also use different display methods, and multiple data sets often need to be combined in common 3-D displays. Our solution to these challenges was to build a single, all-purpose visualization server based on OpenGL graphics libraries, which provides 3-D/2-D displays for many tools in the system. With this component, interactive graphical displays can be built for various data analysis/interpretation tasks (Figures 1 and 2).

Seismic travel time modeling represents perhaps the most impressive improvement arising from this software integration approach. Wide-angle seismic ray tracing and inversion is the key operation used in the interpretation of deep crustal structures. The ray-tracing code "rayinvr" by *Zelt and Smith* [1992] appears to be the most broadly used such code, and several interactive interfaces [e.g., *Song and ten Brink*, 2004] were created for it. However, these interfaces are still limited in their capabilities and, most important, in their ability to incorporate other types of data.

By using the object-oriented rayinvr ray tracer in our system, the power and convenience of analyzing crustal structures improve dramatically (Figure 2). Model depths, velocities, and densities can be displayed in color and edited by using the computer mouse. Velocity columns and contours at any points can be analyzed interactively, and waveform synthetics can be produced for the selected columns. Rays are traced, and the changes in travel times and amplitudes are displayed during model editing, instantly. Reflection travel times and stacked sections can be incorporated in common displays (Figure 2b). Two-dimensional gravity is modeled concurrently with the seismic properties (Figure 2a, top). Teleseismic receiver functions can also be modeled and analyzed in the same ray-tracing session (Figure 2c). Finally, the model can be displayed and ray traced in full 3-D geometry (Figure 2d). The ray tracer also offers several technical enhancements for crooked-line and marine-land recording, and it supports travel time and attenuation tomography and migration.

Note that the various displays in Figures 1 and 2 are created entirely by the user, by describing the sequences of processing tools and their parameterizations (similar to those used in SU or GMT) and without computer coding. The displays are constructed by combining the objects posted by the selected tools, including 3-D geographical base maps obtained directly from GMT databases (Figure 2d). Other types of data (such as heat

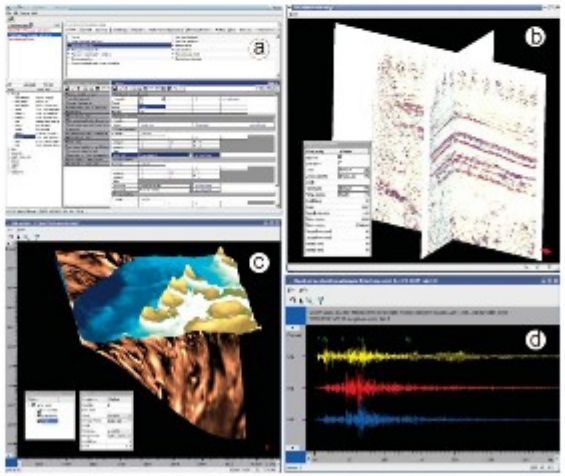


Fig. 1. Examples of interactive graphical displays. (a) user interface, (b) three-dimensional (3-D) reflection seismic sections; (c) 3-D potential-field data; and (d) real-time earthquake data display. Many display options can be selected from the menus.

Get the Licensing in Order

- The best way to keep universities from trying to put restrictions on your contributions is to get parts of the code licensed under the GPL.
- This could be as simple as using a GPL'd library.
- Alternately, one of the authors can “insist” on licensing their changes under the GPL.

Get the Licensing in Order

- This really works. I have personally interacted with Intellectual Property offices from several universities. Once I made it clear that the code could not really be commercialized, they were happy to allow me to use the GPL.
- This is the main reason I recommend the GPL.

Third Party Libraries

- Scientific codes often use external libraries.
- Make sure that those libraries have reasonable licenses!!!!!
  - You may have to rewrite code to remove dependencies.

Numerical Recipes

- A common offender is Numerical Recipes code.
- They wrote a book and provided sample codes, but get upset if you use that in source code that you distribute.
- A good source of replacements is the GNU Scientific Library (GSL). Netlib also has a number of replacements.

Third Party Runtimes

- Your code may depend on third party run times.
  - Matlab -> Octave
  - IDL -> GDL
- Making sure that they work with free run times will make it easier for people to try out your code.

Get the Licensing in Order

- If you mess up the licensing at the beginning, it may never be fixed.

Export Control

- Export control is a completely orthogonal issue from the license. It directly affects your ability to distribute the code.
- There is some confusion in the courts as to whether code is protected by the First Amendment.
- Encryption software is expressly allowed with, at most, some minimal paperwork.

Get the Code in Order

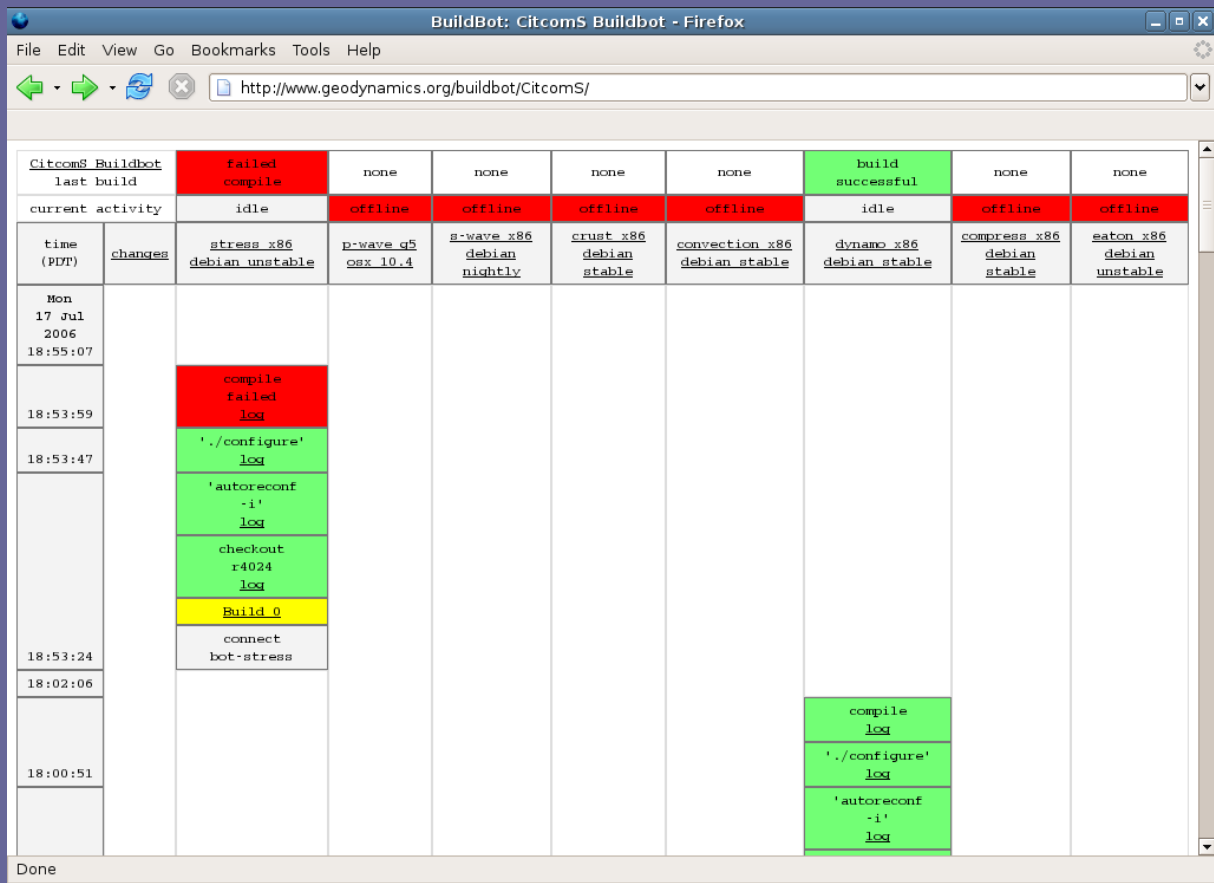
- Case 1: The code is already written
  - Put it in Version Control before you do anything!!!
    - My personal recommendation is Mercurial (hg), although SVN, git, bzr, and CVS are all better than nothing.
    - Make it publicly available.
      - Most people will not use it, but some will.
      - Especially if you make a fix but do not immediately (that day) make a new release. Otherwise you are discouraging contributions.
      - It can simplify your own development: you don't have to authenticate just to get the code on your test machines.
      - It discourages you from waiting too long for “perfect” code, since people can already see what you are doing.

Get the Code in Order

- Make sure it builds on a machine other than your local machine
  - Gigantic Makefiles
  - Autoconf/Automake
  - Scons
  - PETSc's BuildSystem

Get the Code in Order

- Incorporate and run tests and benchmarks
  - Buildbot: Automated regression tests.
  - So you know immediately if you broke something.
  - Useful for rapid development if your tests are good.



Get the Code in Order

- Binaries?
  - It can be quite difficult to get ready-to-run binaries working.
  - Binaries for parallel machines is a lost cause.
  - The most popular binary for CIG's Gale code is Windows.

Get the Code in Order

- Case 2: The code is not yet written
  - Make sure that people want it.
    - You may build it and then no one come.
    - Have a list of people who will actually use the code to help you during development.
    - It does not work to have someone who is just knowledgeable about the field.
    - CIG sponsors many workshops every year to make sure our priorities are aligned with the users.

Get the Code in Order

- This gets particularly tricky with portals, where you provide a simplified web access to a complex code.
- This is like Google, where you do not have to worry about managing 100,000 machines, web crawlers, ranking software, etc. You just type in your search terms..
- Unfortunately, users with enough interest to try it out while it is being developed are probably interested enough to just run the code themselves.

Get the Code in Order

- Choose a language: C, C++, Fortran, Python, Perl, Ruby, ...
  - This may be determined by which libraries you will be using.
  - Don't choose something platform dependent
  - Choose what you know
  - Fortran 90 used to be difficult to support, but gfortran has matured.
  - Mixing languages is great in theory, painful in practice. Do not do it unless you have to.

Get the Code in Order

- Build on other people's work as much as possible
  - PETSc
  - VTK
  - HDF5
- Get other people to do the work for you.
  - CIG partners with RPI, VPAC, USGS, Monash, Texas A&M, UT Austin
- Now see Case 1

Get the Documentation in Order

- README's are nice
- Examples are also good
- Tutorials are even better
- 100+ page professionally edited manuals with slick graphics that provide both a newcomers introduction to using the code and a thorough explanation of every feature and how to use it are best.

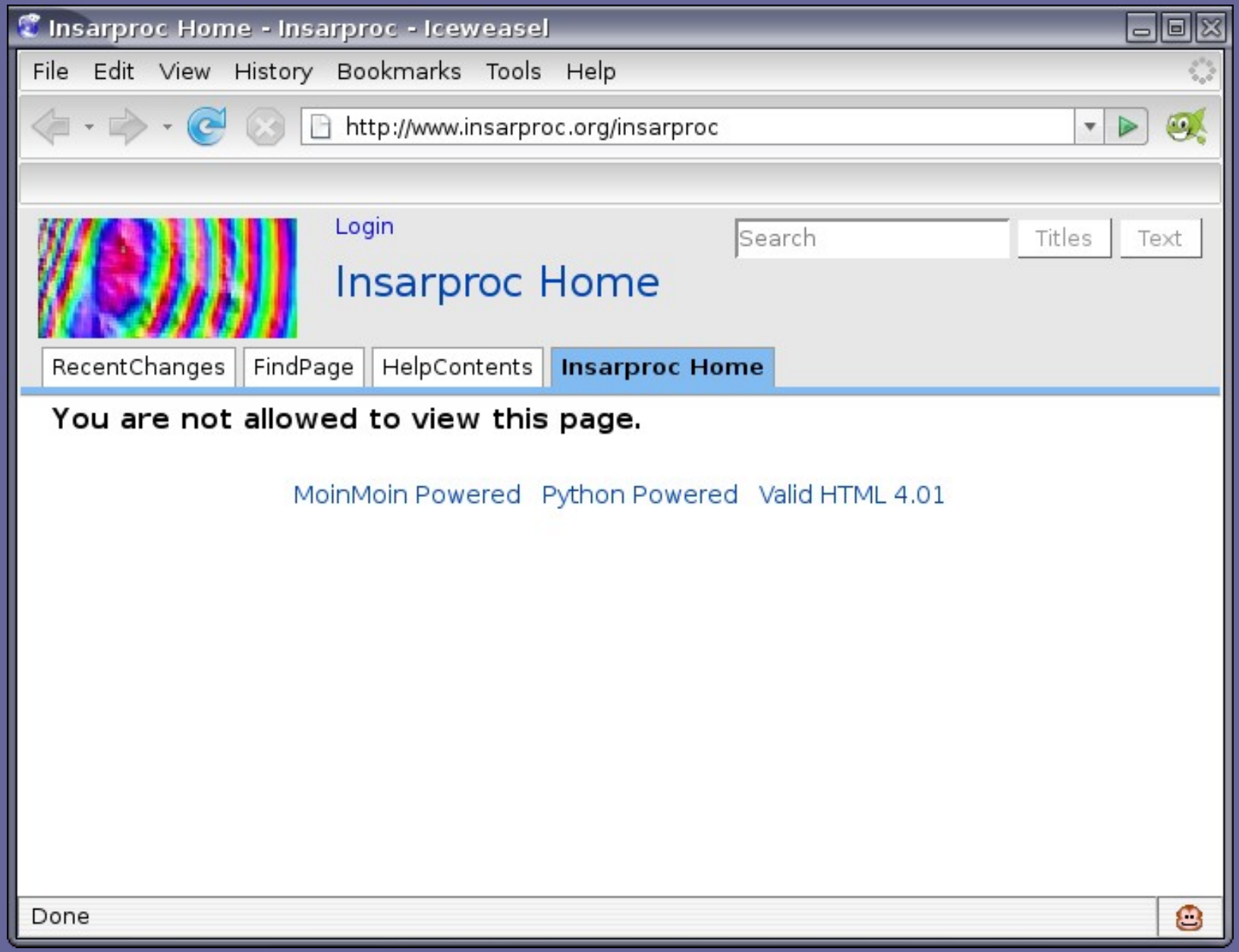
Get the Documentation in Order

- CIG uses LyX to produce the manuals
- It is a GPL'd, cross platform GUI front-end to LaTeX.
- We have had some complaints by LaTeX die-hards, but it has worked well overall.

Promotion

- Write papers with the code
- Present at conferences
- Give tutorials
- Make your website searchable
  - Everything is public by default
  - Do not require registration to download code. Just track IP's. You can still get nice maps.

What were you thinking?



Support

- There should be a clearly labeled email address for people to send questions, bug reports, etc.
  - Making it a mailing list is better. Then it can be archived and searched.
  - Some people are intimidated by mailing lists. Providing a private email can induce more communications, though I am conflicted on this.
- When someone does not understand something, you probably need to tweak your documentation.

Bugs

- Ideally, there should be a bug tracker that is actively maintained.
- If no one uses the bug tracker, then get rid of it.
- CIG uses Roundup, but if I had to do it all over again, I would use Trac.

- Trac provides a bug tracker, web front-end to the repository, project goals and timeline, and a wiki.
- It is an excellent all-in-one solution for single projects, but not so good for multiple projects (though there is progress with Tram).
- Trac's wiki is ok, but not as powerful as some.

- CIG uses Plone for its wiki.
- Plone provides a fairly nice, customizable interface.
- It has all of the fine-grained permissions needed for running multiple projects in parallel.
- It is a major resource hog, usually using all of our servers' memory.
- Upgrades are fearsome.

Wiki

- We have it set up so that individual groups have workspaces for discussion.
- But the main web site can not be modified by a random person.
- One group uses it a great deal, others hardly use it at all.