Editor'sPicks

BY DAVID PAIK, PhD, EXECUTIVE EDITOR

Do you have a few favorite books that you recommend to anyone with an interest in biomedical computing? Are there software products or Web sites that you love to evangelize? We'd like to open this forum up to the community for mini-reviews of biomedical computation—books/papers/Web sites that have caught your interest. Email us at editor@biomedicalcomputationreview.org.

Open-Source Software

ESR's Cathedral or Bazaar

The paper's cryptic title, *The Cathedral and the Bazaar*, dares the reader to delve in and read further. Eric S. Raymond (a.k.a. ESR) is a long-time advocate for open source software and the author of this classic essay. At its most literal level, it documents the history of email software, but this is only the framework for the true topic: a first-hand experiment with the open-source development process that has made Linux so successful.

Raymond describes the traditional closed-source development process as a cathedral, with a centralized, selective, authoritative hierarchy that releases ordained versions of software only when ready. On the other hand, he describes open-source development as a bazaar, with tasks promiscuously delegated among developers with different agendas and approaches.

He expands on the topic by enumerating 19 lessons learned from his experience with fetchmail, about half of which can be generalized beyond programming and software design. Most interesting are Raymond's insights into the culture of the open-source movement. He describes how motivation, teamwork and communication differ from that seen in the "cathedral" setting.

DETAILS

Eric S. Raymond
The Cathedral and the Bazaar

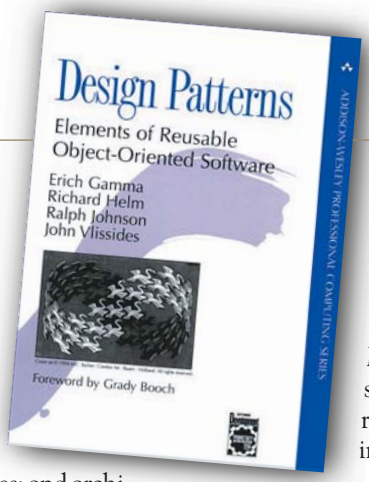
http://www.catb.org/~esr/writings/

Since open-source development is a growing trend in biomedical computing, many of these observations and lessons apply directly. But there are other ways in which this essay may also be fitting. The open-source movement shares numerous similarities with the culture of big science projects, many of which involve biomedical computing in one form or another. Research teams of the future, as outlined in the NIH Roadmap, stress interdisciplinary and collaborative effort that echoes many characteristics of the bazaar model. Many of the 19 lessons could apply to big science projects as well.

One of Raymond's 19 lessons explains why open-source software improves so rapidly. Dubbed "Linus's law" (after Linus Torvalds, who launched the Linux movement), the lesson states: "Given enough eyeballs, all bugs are shallow." In essence, given enough co-developers and beta-testers, a problem (or "bug") will be characterized and fixed by someone. It's an embodiment of the Delphi effect, which states that averaged expert opinion is more reliable than a single individual. If the same lesson proves true for the biomedical computation infrastructure currently under development, that will be good news for the field as a whole.

"If you push mass and velocity high enough you get surprises like nuclear explosions or Linux."

—Eric S. Raymond



Software Engineering

Finding Patterns

Don't reinvent the wheel—sound advice whatever your profession. Novelists have reused archetypal plots and characters throughout history; graphic artists reuse design templates; and architects reuse architectural motifs. The same is true for programmers: certain design problems recur, and there's no need for every programmer to tackle them all over again. Examining reuse at a higher level of abstraction, this book is more about *idea* reuse than *code* reuse. This book is particularly relevant to this community because it promotes the use of solid design concepts that may prove valuable for building a biomedical computing infrastructure.

By relying on reference books such as *Design Patterns* and its progeny, designers can spend their time coming up with new ideas rather repeating others' mistakes.

Instead of discussing programming language syntax or features, this classic text presents design patterns that are commonly used motifs in object-oriented programming. After a short introduction, the book catalogs 23 of the most commonly used design patterns, although it does not claim to be an exhaustive list. The patterns are then divided on two axes. The purpose axis distinguishes between creational (dealing with object creation), structural (dealing with composition of classes/objects), and behavioral (dealing with class/object interaction and distribution of responsibility). The scope axis distinguishes between patterns that deal with classes and objects. Some of the patterns are obvious enough that many designers use them without needing to use this book, whereas other patterns may be less commonly understood.

Design Patterns has become a powerful and important book because it starts to map out the universe of patterns in a single reference collection. Since 1995, when this book came out, others have compiled lists of anti-patterns: programming motifs that should be avoided. Sometimes, studying poor form can be just as instructive as studying good form.

By relying on reference books such as *Design Patterns* and its progeny, designers can spend their time coming up with new ideas rather than reinventing the wheel and repeating others' mistakes.

DETAILS

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software,
Addison-Wesley Professional, 1995.

Development Environment

Fixing Windows (With Cygwin)

Every once in a while, something so ridiculously useful comes along that you feel the urge to go out and tell everyone you know about it. For me, that something is Cygwin, a UNIX-like environment that runs within Microsoft Windows. Mac OS X and Linux may be compelling alternatives, but for diehard Windows users, Cygwin can't be beat for ease of use.

Cygwin is free software and the installer is simple, available at http://cygwin.com. It gives Windows a command line shell that allows the user to use most of the standard UNIX utilities you would expect. Because it is not an emulation layer, programs running within Cygwin run just as fast as native Windows code. Some interesting alternatives exist, such as VMware, coLinux and Bochs, but none of these have the native integration and ease of use of Cygwin. Installing is as simple as running the setup program from the Web site and selecting the individual packages that you want to install; the default selection is rather minimalist.

Cygwin brings tremendous new functionality to Windows.

Cygwin brings tremendous new functionality to Windows, including a full X Windows port that allows X11 programs to run seamlessly alongside Windows programs. One can run various servers include the Apache web server, FTP, SSH/SCP, NFS, and mail. Programming tools include gcc/gdb, perl, python, emacs, vi, make, cmake, doxygen, OpenGL, LAPACK, cvs, subversion and swig. Other miscellaneous applications include PostgreSQL, gnuplot, octave, grep, tar, cron, wget, ghostscript and TeX.

There are, however, a few limitations and caveats. Windows and UNIX handle newlines and carriage returns differently and so care must be taken in I/O programming. Also, the Cygwin API library is covered by GNU General Public License (GPL), so if you publicly release programs compiled under and linked against Cygwin, you must make source code available.

Cygwin has fundamentally changed my computing habits, providing a pretty good best-of-both-worlds compromise. Nothing is perfect but this does make significant improvements to Windows. □

DETAILS

Cygwin
http://www.cygwin.com