

The Rise of Open Source Development Tools

Tony Wasserman
Software Methods and Tools
tony@methods-tools.com

Abstract

For many years, the vast majority of tools used by professional software developers came from commercial vendors. Over the past few years, the availability of open source and non-commercial development tools has greatly increased, giving software development organizations a much broader range of alternatives for development projects.

While many of these development efforts have occurred under the leadership of a collection of open source projects, there is very little coordination among the various efforts. As a result, it is now relatively easy to assemble a collection of open source development tools, but it remains difficult to create an integrated environment of such tools. This paper describes numerous open source projects and some of the possible approaches for creating an integrated environment.

1. Historical Background

Open source software dates to the earliest days of programming. Numerical analysts published the source code of algorithms, making the code freely available to others. Software developed for the US Government was (and is still) available (in principle) to anyone who requests it. For all practical purposes, there was no commercial software industry before the early 1970's, so many of today's issues relating to software licensing and intellectual property were unexplored. (Note: the term "open source" is used here in line with the definition given by the Open Source Initiative at http://www.opensource.org/docs/definition_plain.html.)

Software systems, including operating systems and programming language compilers, were typically bundled with hardware. In 1969, though, IBM decided to unbundle its software products and services from their hardware; this event can be seen as triggering the development of a market for software products.

1.1. Unix tools

In the mid-1970's, AT&T Bell Laboratories created a non-commercial distribution of the Unix™ system, distributing it to universities and other research institutions, where it quickly became the *de facto* standard for learning about systems software in academic institutions. The University of California, Berkeley, with funding from the US Department of Defense Advanced Research Projects Agency (ARPA), created successive non-commercial distributions of Unix, augmented by numerous locally developed tools, including the *vi* text editor, as well as such development tools from AT&T Bell Laboratories, including *scs* and *make*. The source code was always part of the distributions, making it easy to port Unix and its tools to a wide variety of computer systems. Rosenberg [1] and McKusick [2] discuss this history in greater detail.

Kernighan and Plauger, part of the Unix team at AT&T Bell Laboratories, were among the first to discuss tools as a separate topic [3]. They followed what has become known as the Unix "toolbox" philosophy: "Make each program do one thing well.... Expect the output of every program to become the input to another, as yet unknown, program" [4, 5]. McIlroy introduced the notion of a "pipe" [6], which facilitated composition of tools. The Unix distributions included not just the underlying system, but also a broad collection of tools. All of the application programming interfaces (API's) and data representation formats were a standard part of the documentation

As the market for Unix-based commercial software tools developed in the 1980's, some products were designed to adhere to the principles of Unix tools. Wasserman and Pircher named this philosophy "open architecture" [7,8], and it has been widely adopted among developers of tools and environments.

1.2. Open Source Platforms

In addition to Unix, other open source development projects have existed for more than 20 years. Among these, the Emacs editor was developed beginning in 1978 under the aegis of the Multics project sponsored by ARPA at MIT [9]. The GNU Project, begun in 1984,

initially produced a later version (GNU Emacs) and the GNU C Compiler [10], and has led the GNOME project to create a user-friendly “desktop” environment on open source software (<http://www.gnome.org>). The Free Software Foundation (FSF) is now the organizational sponsor of the GNU Project. FSF and UNESCO maintain a directory of free software at <http://www.gnu.org/directory>.

The Unix system served as the intellectual foundation for several open source operating systems, including Linux (<http://www.linux.org>) (distributed under the GNU General Public License), FreeBSD (<http://www.freebsd.org>), OpenBSD (<http://www.openbsd.org>) and NetBSD (<http://www.netbsd.org>).

Beyond the operating systems themselves, though, are numerous other open source software components that provide the framework or platform on which applications run. The Linux effort, for example, includes more than 75 related software development projects (<http://www.linux.org/projects/software.html>).

Another widely used piece of open source software is the Apache HTTP Server, now a project of the Apache Software Foundation (<http://www.apache.org>). Apache, developed since 1995, has long been the most widely used web server, with many of its installations running on an open source operating system. The Apache Foundation also sponsors the Jakarta project, which creates and maintains open source solutions on the Java platform. Among the subprojects is Tomcat, the servlet container used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies, and Struts, a model-view-controller framework for Java-based web applications.

Other important platform components are relational database systems, exemplified by MySQL (<http://www.mysql.org>), and J2EE application servers, exemplified by JBoss (<http://www.jboss.org>). Both MySQL and JBoss have established commercial enterprises that complement the non-commercial, freely usable versions of the software. These businesses are successfully competing against traditional closed-source commercial software vendors.

The operating system, web server, DBMS, and application server, along with scripting languages (e.g., Perl, PHP, Python, Tcl, and Groovy), provide a powerful foundation for both traditional and web-based applications, making it possible for organizations to avoid the high costs of traditional commercial products for these services. By combining these software components with low-cost, commodity hardware, it becomes very inexpensive to deploy and run both traditional and web-based applications. What was missing, though, was a correspondingly attractive set of development tools for designing, building, and testing these applications.

1.3. Development Tools

Development tools were dominated by commercial efforts throughout the 1980’s and much of the 1990’s. Leaders among tool vendors included Microsoft, Oracle, IBM, Adobe, Borland, Macromedia, Rational Software (acquired by IBM), and Mercury Interactive. There were very few open source tools in the categories addressed by these and other commercial vendors.

2. Open Source Tools

In the past five years, though, the number of projects to create open source development tools has grown tremendously, driven by the overall success of open source platform and enabled by projects that allow developers from around the world to collaborate and contribute to the development of a tool. The economic downturn in developed countries, with the end of the “dot-com” bubble, led to a significant decrease in purchases of licenses for traditional commercial software tools. Even some commercial vendors have now released source code for some or all of their products. For example, Microsoft has established a Shared Source Initiative (<http://www.microsoft.com/resources/sharedsource/default.aspx>) and has released their Windows Installer XML under this Initiative.

2.1 Open Source Project Sites

Interest in open source development has led to the creation of numerous community sites that support collaborative software development projects, many of which involve development tools. These projects are typically characterized by a collaborative, consensus based development process, with a designated project leader and a small team of individuals who are authorized to “commit” their code to the evolving project. Others can review the code and can try out the software, making suggestions for changes.

Among the open source development sites are:

- 1) Apache – produces the Apache HTTP server and the various Jakarta Project solutions for the Java platform (<http://www.apache.org>)
- 2) Mozilla.org – manages the Mozilla, Firefox, and Camino web browsers, the Bugzilla issue tracking system, and the Thunderbird email client (<http://www.mozilla.org>)
- 3) SourceForge – the world’s largest site for open source development, currently hosting more than

- 80,000 separate projects, including some involving tools (<http://sourceforge.net>)
- 4) CVS Home – oversees development of the CVS version control system (<http://www.cvshome.org>)
 - 5) Eclipse – a universal, open, extensible tools platform created with support from more than 50 member companies; current projects include a C/C++ IDE, a modeling framework, and a web tools platform (<http://www.eclipse.org>)
 - 6) Mono – a Novell-sponsored project to create an open-source version of Microsoft .Net (<http://www.go-mono.com>)
 - 7) WebDAV – web-based distributed authoring and versioning (<http://www.webdav.org>)
 - 8) Tigris – open source tools for project management and UML modeling (<http://www.tigris.org>)
 - 9) JUnit – Java automated testing tool hosted on SourceForge (<http://www.junit.org>)
 - 10) NetBeans – the project that develops the NetBeans IDE (<http://www.netbeans.org>)

Several of these projects use CollabNet SourceCast (<http://www.collab.net>) or the VA Software SourceForge environment as the underlying enabling technology. These collaboration mechanisms provide a project repository along with version control, configuration management, mail, and project management support.

These projects largely exist independently of one another, even within a specific site. While a site typically has its own governance and leadership, as is the case with Apache, Mozilla.org, and others, the individual projects usually have their own missions and leadership. The situation is analogous to that of a large apartment block with many individual units. There are owners and managers, as well as some centralized services and policies, but the tenants of each apartment are then free to do as they wish within the established management structure.

2.2. Open Standards

Tool integration has long been a key issue in the creation of software development environments. The fundamental problem, of course, is to identify mechanisms by which tools could interoperate with one another, primarily by passing control or by sharing data. Tool developers (both open source and commercial) devise their own file formats, greatly complicating problems of integrating products, particularly when a tool developer changes those formats on successive versions of the tool.

Wasserman [11] defined five dimensions of tool integration: data, control, presentation, process, and platform. Thomas and Nejme [12] expanded that definition, making a valuable distinction between the

view of the environment user and that of the environment builder(s).

Initial efforts at tool integration included the ability of one tool to invoke another tool through remote invocation, message passing, and the aforementioned Unix pipe mechanism. The vast majority of these efforts were “point to point”, where two tool builders defined an integration mechanism specifically for those two tools. Integrating a larger number of tools was exponentially harder.

There were several initiatives to create integrated environments using a shared repository or a common framework. These included PCTE (Portable Common Tool Environment), sponsored by ECMA [13], IBM’s AD/Cycle with its repository [14], and a tool integration framework created by a startup company, Atherton Technology. ECMA and NIST sponsored a project that led to a reference model for such frameworks [15], known as the “toaster” model for the graphical representation of the model. None of these efforts proved successful, both for business and technical reasons. In the absence of significant customer demand, tool builders could not justify the effort required to integrate their tools into such frameworks.

Over the past decade, however, the customer perspective has changed significantly. Customers have become much more concerned about long-term dependence on a single tool supplier. Even when one claimed adherence to standards, features in their products often went above and beyond the standard. Customers discovered that different versions of Unix or SQL databases were incompatible with one another, and that the cost of switching from one to another was often prohibitive.

The widespread acceptance of the Internet has also been a significant factor in encouraging the development of standards, since the World Wide Web and web-based applications could simply not exist in their present form without universal adherence to key standards. The World Wide Web Consortium (W3C) (<http://www.w3c.org>) has developed and sponsored the creation of “interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential.” The W3C has worked along with the Internet Engineering Task Force (IETF) (<http://www.ietf.org>), a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

OASIS (<http://www.oasis-open.org>) works similarly, serving as a non-profit consortium driving the adoption and standardization of e-business standards through a series of technical committees that follows a well defined process. It has produced standards for e-business, web services, and security, just to name a few. OASIS has also become the steward for the Universal Description,

Discovery, and Integration (UDDI) project, which was initially developed by three companies and endorsed by more than 30.

The Java community has moved in the same direction. Java was defined by Sun Microsystems with the goal of having the same software should run on many different systems. Adherence to the JavaVM specification was a key issue in the recently resolved dispute between Sun and Microsoft. Sun continues to control the language specification and lead many of the Java-related projects (<http://java.sun.com>), but has established the Java Community Process (JCP) (<http://jcp.org>) as a mechanism to allow broad participation in the definition and adoption of Java technology standards. The JCP has repeatedly evolved to expand the ways in which developers can participate in the creation of new standards, each of which is known as a Java Specification Requests (JSR). There are, at present, more than 200 of these, including the language and the JCP itself.

Sun has also established compatibility test suites for J2EE application servers and JDBC drivers. The goal is to strengthen the notion of the open standard and thereby to increase customer confidence that products verified for conformity to these suites will all yield consistent results.

Along the same lines, the Object Management Group (OMG) (<http://www.omg.org>), has sponsored the ongoing evolution of the UML notation, taking over from the initial work done by Rational Software (now part of IBM) and its partners. Part of this effort includes a diagram interchange specification, aimed at allowing users of a standards-compliant UML modeling tool to transfer and use models created in that tool in another standards-compliant UML modeling tool. OMG has also sponsored the development of the XML Metadata Interchange (XMI), a framework for defining, manipulating and integrating XML data and objects.

In summary, the high costs of incompatibility among tools and the difficulty of tool integration and interoperability have led to greatly increased efforts to create standards that can be endorsed and followed by tool suppliers. Development organizations have become much more savvy in evaluating tools based on their support for the relevant standards. Of course, proprietary interfaces still remain, but products using such interfaces are frequently accompanied by documentation on file formats, APIs, and messaging interfaces, much as was done with the original Unix distribution. The idea is that conformance to standards, combined with published interfaces, will make it much easier to build and maintain integrated development environments. This notion provides a technical approach and a valuable business case for tool developers to make sure that their tools work within established framework(s), providing consistency and interoperability with others.

2.3. Open Source Environments

From the perspective of a development organization seeking a set of tools (or an integrated environment) to support the various development tasks, the situation is just as difficult as ever, if not more so. Applications have become more complex, building on a complex runtime infrastructure, and often running on a network of machines. More tools are needed to specify, create and test the various components that comprise these applications.

A software development organization needs “horizontal” tools, “vertical” tools, and runtime support [11]. Horizontal tools span the development process and include tools for coordination, project and process management, version control, configuration management, and documentation. Vertical tools address specific tasks in the development process, and may include tools for modeling, GUI design, code analysis, and testing, in addition to a language-based IDE. Runtime support may include interpreters for scripting languages and runtime services from a language-based IDE, as well as a web server, application server, and content management system.

However, with both “closed” and open source tools, there’s no single source from which all of the needed tools can be obtained. Furthermore, there is no broad agreement on tool integration mechanisms that assures the interoperability of the tools. This situation means that a development organization must mix and match open source, commercial, and internally developed tools to create a best-of-breed environment. Some tool suppliers and open source sites have gone a long way to assure integration and consistent operation of their own tools, and have adhered to open standards. However, the degree of integration falls well short of that available in office suites (text editing, spreadsheets, presentation, drawing).

In many respects, then, the challenge of creating integrated development environments remains as great as it was two decades ago. Today’s challenge is slightly mitigated by the growth of open standards and common messaging mechanisms, but increased by the need to integrate a larger number of tools into the environment.

One particularly promising effort at tool integration comes from the Eclipse Platform subproject, which “provides the core frameworks and services upon which all plug-in extensions are created. It also provides the runtime in which plug-ins are loaded, integrated, and executed. The primary purpose ... is to enable other tool developers to easily build and deliver integrated tools.” (<http://www.eclipse.org/platform>)

The Eclipse Platform provides “common facilities required by most tool builders including a standard workbench user interface and project model for managing

resources, portable native widget and user interface libraries, automatic resource delta management for incremental compilers and builders, language-independent debug infrastructure, and infrastructure for distributed multi-user versioned resource management.” Other tools are integrated using a plug-in mechanism that allows these tools to specify how they are to be integrated, including the other resources that they need [16,17].

This plug-in approach has been very successful with image manipulation and document publishing tools, such as Adobe Photoshop and QuarkXPress, respectively. It is too soon to tell if it will be equally successful when applied to a platform, as is being done in Eclipse. Nonetheless, it is the first innovative approach to tool integration in many years.

3. Summary and Directions

The rapid increase in open source development projects and tools has major implications for the development of tools and integrated environments. The many open source development sites and their openness to broad participation combine to provide settings that encourage innovation and experimentation in the creation of tools and environments. Frameworks such as the Eclipse Platform make it straightforward for an individual or small team to create a tool that fits into a framework using the plug-in mechanism.

It seems likely that evolution toward integrated environments will follow the path established by many of the open source efforts, with many separate projects sharing common goals for interoperability and use.

Apart from the technical issues, there are numerous business considerations. First, there has been a significant drop in the price of commercial development tools. Fifteen years ago, the average cost of an enterprise quality modeling tool was \$8000 US; today, it is closer to \$2000. In many categories, tools and platform software are gradually being commoditized by comparable software developed on open source projects. This trend is likely to continue, with such tools becoming the preferred choice in both industrial and research environments.

4. References

[1] Rosenberg, D.K., *Open Source: the Unauthorized White Papers*, M&T Books, Foster City, CA, 2000.

[2] McKusick, M.K., “Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable”, in *OpenSources: Voices from the Open Source Revolution*, ed. C. diBona, S. Ockman, and M. Stone, O’Reilly & Associates, Sebastopol, CA, 1999, pp. 31-46.

[3] Kernighan, B.W. and P.J. Plauger, *Software Tools*, Addison-Wesley, Reading, MA, 1976.

[4] Salus, P.H. *A Quarter Century of UNIX*, Addison-Wesley, Reading, MA, 1994.

[5] McIlroy, M.D., E.N. Pinson, and B.A. Tague “Unix Time-Sharing System: Foreword”, *The Bell System Technical Journal*, July–Aug, 1978 vol 57, number 6 part 2, p. 1902.

[6] Mahoney, M. (ed.), “An Oral History of Unix,” <http://www.princeton.edu/~mike/unixhistory>.

[7] Wasserman, A.I. and P.A. Pircher, “A Graphical, Extensible Integrated Environment for Software Development,” *ACM SIGPLAN Notices*, vol. 22, no. 1 (January, 1987), pp. 131-142. (Proc. ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments)

[8] Wasserman, A.I. and P.A. Pircher, “Visible Connections,” *Unix Review*, vol. 4, no. 10 (October, 1986), pp. 62-73.

[9] Greenberg, B.S., “Multics Emacs: The History, Design and Implementation.” <http://www.multicians.org/mepap.html>

[10] Stallman, R., “The GNU Operating System and the Free Software Movement,” in *OpenSources: Voices from the Open Source Revolution*, ed. C. diBona, S. Ockman, and M. Stone, O’Reilly & Associates, Sebastopol, CA, 1999, pp. 53-70.

[11] Wasserman, A.I., “Tool Integration in Software Engineering Environments,” in *Software Engineering Environments*, ed. F. Long. Springer Verlag, Berlin, 1990, pp. 137-149. (Lecture Notes in Computer Science, no. 467)

[12] Thomas I. and B. Nejme, Definitions of Tool Integrations for Environments *IEEE Software* vol. 9, no. 2 (1992), pp. 29-35.

[13] Campbell, I. “Portable Common Tool Environment,” *Computer Standards and Interfaces*, vol. 8, no. 1 (1988), pp. 66-74.

[14] Mercurio, V.J., B.F. Meyers, A.M. Nisbet, and G. Radin, “AD/Cycle Strategy and Architecture,” *IBM Systems Journal*, vol. 29, no. 2 (1990), pp. 170-188.

[15] National Institute of Standards and Technology. Reference model for frameworks of software engineering environments, 3rd ed. Special Publication 500-211, 1991.

[16] Gallardo, D., E. Burnette, and R. McGovern, *Eclipse in Action: A Guide for the Java Developer*, Manning Publications, Greenwich, CT, 2003.

[17] Object Technology International, Eclipse Platform Technical Overview. February, 2003. <http://www.eclipse.org/whitepapers/eclipse-overview.pdf>