Creating a Free, Dependable Software Engineering Environment for Building Java Applications

Marcus Bittman
Allegheny College
Department of Computer Science
Meadville, PA 16335
814-332-3629
bittmam@allegheny.edu

Robert Roos
Allegheny College
Department of Computer Science
Meadville, PA 16335
814-332-2883
rroos@allegheny.edu

Gregory M. Kapfhammer
Allegheny College
Department of Computer Science
Meadville, PA 16335
814-332-2877
gkapfham@allegheny.edu

ABSTRACT
As open source software engineering becomes more prevalent, employing sound software engineering practices and the tools used to implement these practices becomes more important. This paper examines the current status of free software engineering tools. For each set of tools, we determined the important attributes that would best assist a developer in each stage of the waterfall model. We rated each tool based on predetermined attributes. We used the creation of a graphical user interface based email client in Java to assist in evaluating each tool. Our findings show that there is still a need for free tools to extract UML diagrams, test graphical user interfaces, make configuring Emacs easier, and profile Java applications. In other areas there are free tools that provide satisfactory functionality such as Concurrent Versions System (CVS), GVim, JUnit, JRefactory, GNU Make, Jakarta Ant, Javadoc, and Doc++.

Keywords
Open source, software engineering, development tools, freeware

1 INTRODUCTION

Current projects developed in the open source arena do not necessarily adhere to process models such as the waterfall or evolutionary approaches to creating software. In most cases, an open source software project is started by a single developer and then posted at a website such as [23] for further work by the open source community.

Although many projects have been successful (Linux [13] and Apache [2] are two examples), there is still a need for free tools to create dependable software. We have examined a variety of free tools to assess the current state of free software development using an Athlon class machine running Red Hat 7.0 [19]. The tools have been categorized according to the stages of an abbreviated waterfall model which include the following [21]:

- Requirements definition/system and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

We tailored our environment to tools for Java application development since the language is platform independent [25]. We implemented a graphical user interface (GUI) based email client with Java 2 Standard Development Kit (SDK) 1.3 to provide a means to evaluate each tool. In order to rate these tools without contaminating results, we used several different tools during each phase of the project's development for different tasks (see [4] for detailed results).

We rated the tools according to the following scheme. Values 2, 4, 6, and 8 are not described—they provide a means to fine tune scores (see Tables 1 and 2 for tool ratings).

- 0—the attribute is a failure
- 1—the attribute contains many problems and has one useful trait
- 3—the attribute has many problems and few useful qualities
- 5—the attribute is usable, but contains a significant number of problems
- 7—the attribute is fairly usable, but contains minor problems
- 9—the attribute is almost perfect except for one or two minor cosmetic problems
- 10—no problems encountered with this attribute

Section 2 discusses tools that were deemed the most mature, while Section 3 considers tools that need to be improved. In Section 4, we provide conclusions for this work.

2 THE MOST REFINED

The tools we found most useful were Emacs [6], GVim [7], Netbeans [16], GNU Make [22], Jakarta Ant [1], JUnit [3], JPython [9], JRefactory [20], CVS [5], Doc++ [29] and Javadoc [26].

Editors/Integrated Development Environments

Ideally, a developer would use Emacs or GVim to write and edit source code, while using Netbeans to create GUIs. Emacs was more customizable than GVim, but configuring Emacs was more difficult than GVim.

Build Systems

We found that GNU Make [22] executed commands faster than Jakarta Ant [1] due to Ant's usage of Java to issue rules. Since Ant utilizes Java commands, its build files are platform independent [1].

Testing Tools

JUnit successfully provided an interface to create a harness to test Java applications on the unit level, meaning that it establishes a model to compare an expected value to the value returned from a method [3].

Since traditional capture/replay tools are limited by component location, we utilized JPython to create replay scripts to automate the usage of a GUI [9]. JPython has the ability to utilize classes from Java's API and provides a compiler to turn JPython scripts into Java bytecode.

Refactoring Tools

JRefactory restructures Java code and performed well on the limited refactorings that we tested. They included moving a class and extracting an interface from a pre–existing class [20].

Version Control Systems

We found that CVS supports multiple user access to a single file very concisely [5]. It also contains provisions for accessing a repository through the Internet, thereby making development on a single project's source code possible from anywhere in the world.

Documentation Extraction Tools

While Javadoc extracts documentation from only Java source code [26], Doc++ can extract documentation from C/C++ and Java code [29]. Javadoc extracts HTML documentation and joins an application's documentation with the Javadoc from Java's API. Doc++ gives the user the ability to create HTML or $\LaTeX$ documentation. We recommend using Javadoc for HTML documentation and Doc++ for printed documentation.

3 THE LEAST USEFUL TOOLS

The least useful tools were Argo [27], Dia [11], SuperWomble [8], and HProf [15]. These tools provide useful options, but are not up to the standards set by tools in the previous category.

Argo, a UML diagramming tool, was built with Java [27]. Since the application was built with Java, the interface can be sluggish. This tool's future usefulness will lie in developers' abilities to make the interface more responsive.

Dia, a lightweight diagramming tool, contains a toolkit for creating UML diagrams [11]. Unfortunately this tool does not include provisions to ensure that diagrams adhere to UML standards.

SuperWomble is a tool that extracts UML diagrams from Java bytecode [8]. This tool isn't highly developed because it is a research prototype. Therefore its use is limited.

HProf is a profiler for Java applications that provides the ability to track many different types of information such as CPU and memory usage [15]. It is deficient in its ability to display information in a readable format.

4 CONCLUSION

The greatest benefit of this research was the identification of the open source community's need for the following.

- a capture/replay tool to test and automate Java Swing GUIs, which could be created with aid from classes in the Java API
- a diagramming tool for UML that provides a more responsive user interface than Argo
- the creation of a GUI interface or a series of configuration scripts to configure Emacs—changing to XEmacs is a possibility [28].
- a profiling tool for Java applications that creates readable information (see [12] for a discussion of the JVMPI)

This research assessed the current state of free tools for software engineering. We conclude that there is a distinct need for tools to better assist developers in the specified areas of the software engineering lifecycle. Furthermore, there is an ongoing need for the assessment of free tools for software development as languages and processes evolve. This ensures that developers are afforded the most advanced suite of tools to assist in the efficient creation of dependable software.

Attributes / Tools	Argo	Dia	Emacs	GVim	Netbeans	GNU Make	Jakarta Ant	JUnit 3.4
Ease of Installation	9	10	3	9	6	9	7	9
Documentation/Help	5	6	7	8	8	10	9	7
UML Support	6	7	—	—	—	—	—	—
Diagram Portability	10	8	—	—	—	—	—	—
Savable Properties	10	9	—	—	—	—	—	—
Dependability	8	5	—	—	—	—	—	—
Skeleton Code Generation	n/a	n/a	—	—	—	—	—	—
Formatting Readability	—	—	9	7	8	—	—	—
Ability to Make Changes Quickly	—	—	7	9	4	—	—	—
Configurability/Customizability	—	—	9	6	9	—	—	—
Java Support	—	—	8	7	8	—	—	—
Ease of Use	—	—	—	—	—	9	9	—
Ability to Create a Test Harness	—	—	—	—	—	—	—	9
Ability to Test Many Different Methods with One Harness	—	—	—	—	—	—	—	7
Viewability/Detail of Results	—	—	—	—	—	—	—	9
Reuse of Test Cases	—	—	—	—	—	—	—	9

"n/a" – tool doesn't contain an attribute that can be assessed, but should contain this attribute
"—" – tool wasn't rated on the attribute and shouldn't contain the attribute

Table 1: Tool Ratings

Attributes / Tools	SuperWomble	Metamata Debugger	JPython	JRefactory	HProf	CVS	Doc++	Javadoc
Ease of Installation	4	10	9	7	n/a	7	4	n/a
Documentation/Help	4	8	5	7	5	9	7	9
Ability to Extract Comments	—	—	—	—	—	—	4	10
Comment Compatibility	—	—	—	—	—	—	10	10
Documentation Formatting	—	—	—	—	—	—	7	9
Formatting Options	—	—	—	—	—	—	8	9
Extractable Dependencies	4	—	—	—	—	—	—	—
Readability/Manipulation of Diagrams	3	—	—	—	—	—	—	—
Ease in Creation of Diagrams	7	—	—	—	—	—	—	—
Portability of Diagrams	4	—	—	—	—	—	—	—
Iteration Options	—	7	—	—	—	—	—	—
User Interface	—	8	—	9	—	—	—	—
Ability to Manipulate a GUI	—	—	7	—	—	—	—	—
Test Case Life	—	—	9	—	—	—	—	—
Options for Refactoring	—	—	—	6	—	—	—	—
Quality of Refactorings	—	—	—	9	—	—	—	—
Profiling Capabilities	—	—	—	—	9	—	—	—
Usability of Output	—	—	—	—	7	—	—	—
Ease of Configuration	—	—	—	—	—	5	—	—
Ease of Check Out/In	—	—	—	—	—	8	—	—
Access Control	—	—	—	—	—	6	—	—
Multiple Accesses	—	—	—	—	—	9	—	—
Version Control	—	—	—	—	—	9	—	—

"n/a" – tool doesn't contain an attribute that can be assessed, but should contain this attribute
"—" – tool wasn't rated on the attribute and shouldn't contain the attribute

Table 2: Tool Ratings Continued

REFERENCES

[1] Ant. The Jakarta Project, Jan 2001. `http://jakarta.apache.org/ant/`; accessed January 20, 2001.

[2] The Apache Software Foundation. Apache.org, March 2001. `http://www.apache.org`; accessed March 11, 2001.

[3] K. Beck and E. Gamma. JUnit, September 2000. `http://www.junit.org`; accessed September 24, 2000.

[4] M. Bittman. Creating a Free, Dependable Software Engineering Environment. Technical Report CS01–03, Allegheny College, March 2001. `http://ace.allegheny.edu/~bittman/`.

[5] CVSHome.org. CVS Home, October 2000. `http://www.cvshome.org/`; accessed October 10, 2000.

[6] GNU Org. Free software, Sep 2000. `http://www.gnu.org`; accessed September 17, 2000.

[7] S. Guckes. The Vim (Vi Improved) Home Page, September 2000. `http://www.vim.org`; accessed September 24, 2000.

[8] D. Jackson. Womble, Oct 2000. `http://sdg.lcs.mit.edu/womble/`; accessed October 11, 2000.

[9] JPython.org. Jpython, Jan 2001. `http://www.jpython.org/`; accessed January 8, 2000.

[10] P. Kinnucan. Java Development Environment for Emacs, December 2000. `http://sunsite.dk/jde/`; accessed December 21, 2000.

[11] A. Larsson and J. Henstridge. Dia's homepage, November 2000. `http://www.lysator.liu.se/~alla/dia/`; accessed November 25, 2000.

[12] S. Liang and D. Viswanathan. Comprehensive Profiling Support in the Java Virtual Machine. *5th USENIX Conference on Object–Oriented Technologies and SYstems*, 1999.

[13] Linux.org. Linux online, March 2001. `http://www.Linux.org`; accessed March 11, 2001.

[14] MetaMata.com. Metamata Development Environment Personal Edition, January 2001. `http://charlie.metamata.com/java/servlet/automation.Charlie?l=i`; accessed January 13, 2001.

[15] N. Meyers. Perfanal: A Performance Analysis Tool, Sep 2000. `http://developer.java.sun.com/developer/technicalArticles//Programming/%perfanal/index.html`; accessed September 31, 2000.

[16] NetBeans.org. Netbeans, September 2000. `http://www.netbeans.org`; accessed September 24, 2000.

[17] G. Pennington. Jprof, Sep 2000. `http://starship.python.net/crew/garyp/jProf.html`; accessed September 31, 2000.

[18] RCS. Cyclic RCS page, October 2000. `http://www.cvshome.org/cyclic/cyclic-pages/rcs.html`; accessed October 10, 2000.

[19] Red Hat, Oct 2000. `http://www.redhat.com`; accessed October 16, 2000.

[20] C. Seguin. Software, Sep 2000. `http://users.snip.net/~aseguin/chrissoft.html`; accessed September 31, 2000.

[21] I. Sommerville. *Software Engineering.* Pearson Education, Harlow, England, 2001.

[22] R. M. Stallman and R. McGrath. GNU Make, Jan 2001. `http://www.gnu.org/manual/make/html_chapter/make_toc.html`; accessed January 20, 2001.

[23] Source Forge. Breaking Down the Barriers to Open Source Development, March 2001. `http://www.sourceforge.net`; accessed March 11, 2001.

[24] Sun Microsystems. Java Virtual Machine Profiler Interface (JVMPI), Feb 1999. `http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html#hprof`; accessed January 11, 2001.

[25] Sun Microsystems. Java, Sep 2000. `http://java.sun.com`; accessed September 24, 2000.

[26] Sun Microsystems. Javadoc, Sep 2000. `http://java.sun.com/j2se/1.3/docs/tooldocs/javadoc/index.html`; accessed September 24, 2000.

[27] Tigris.org. Argouml, September 2000. `http://argouml.tigris.org`; accessed September 24, 2000.

[28] XEmacs. XEmacs, March 2001. `http://www.xemacs.org`; accessed March 14, 2001.

[29] M. Zckler and R. Wunderling. Doc++, Sep 2000. `http://www.zib.de/Visual/software/doc++/index.html`; accessed September 24, 2000.