free software report
Volume 3, Number 1

# Testing, testing 1 -2 - 3 Can anybody hear me?

**Free Software, regression testing and standards; what do they have in common? This issue of the Free Software Report will introduce you to software testing, the problems inherent in the process, and will discuss a free software solution for this phase of software development.**

## The Problem:
### Software Testing or Regression Testing
By Jack Woehr

Users have come to expect high quality in software products and services. Regression testing is one of the major ways to improve quality. Freely available software now exists to automate regression testing; and free software increasingly comes with suites of tests that verify the quality of the software.

**In this issue:**



Regression testing operates like the family doctor who knows that his patient has had heart problems. The patient's heart is the first thing he checks on the next visit. It's a way of building up a set of tests, starting with tests that were written to isolate failures in the past, and adding more tests as new trouble reports are investigated and new features are added.

It's common that fixing a bug sometimes causes new bugs, or causes old bugs to reappear. This is called regression, because the software "regresses", becoming unable to do something that once worked. Regression testing is best suited to catching these changes that inadvertently destabilize a previously stable program. Once the test suite has grown large, it can also detect problems that there is no specific test for, because each test will necessarily depend on more than just one feature of the program, and will show a failure when these other "incidental" features break. The doctor may be listening for a heart murmur, but he'll also notice the sound of ragged breathing and be able to catch a lung problem.

The process of regression testing involves applying a test suite to a piece of software. Individual test cases in the test suite can produce one of four results: **expected passes, expected failures, unexpected failures, and unexpected passes.**

Continued on page 2

Continued from page 1

An **expected pass** is a test case which executes without error, and for which no error is expected, perhaps because the test case represents a well-known and long-fixed bug, or because it tests an important feature that is not known to have any problems.

An **expected failure** is a test case which produces an error which was anticipated by the author of the test cases. Perhaps the expected failure represents a known bug which is on the to-do list, or is testing a design feature not yet implemented.

**"The whole point of a standard is interoperability, but rigorous testing is required before different implementations of a standard become truly interchangeable and interoperable."**

An **unexpected failure** is a test case which returns an error in an instance where this result was not anticipated. This is where the test engineer and programmer must concentrate more effort; it probably represents a bug which has been inadvertently introduced.

Only slightly less exciting to the test engineer is an **unexpected pass**. This probably indicates that a previously tested problem has been fixed, but the test case was not modified to know that the fix had been applied. An unexpected pass might also indicate that a test case is poorly constructed, or that the wrong feature is being tested. Unexpected passes aid in debugging the test suite itself.

One difficulty of regression testing is that the collection of test cases represented by the test suite can grow quite large. Without early planning and appropriate automation, the tests will become unwieldy, and interpreting their results will take too long. In other articles in this newsletter, Rob Savoye and Michael P. Gerlek discuss a free software program that helps to automate the collection, organization, performance, and evaluation of test cases.

Tests become even more important where standards are concerned. The whole point of a standard is interoperability, but rigorous testing is required before different implementations of a standard become truly interchangeable and interoperable. Andy Oram, of O'Reilly and Associates, discusses a free software effort to produce a comprehensive test suite for implementations of the POSIX standards.

**As with every issue, we welcome any comments and suggestions you may have about the content of the Free Software Report. Our primary interest is to write about the topics you care most about. Please send your suggestions to fsr@cygnus.com.**

## The Standard: A Test-It-All Project

By Andy Oram

Kind of ironic, isn't it, that the main standard in contemporary computing doesn't have it's own test suite? Oh, lots of UNIX vendors run the System V Verification Suite (SVVS), but it doesn't adequately check for all expected behaviors. You can tell this because the utilities and libraries from different vendors have so many annoying little differences, which keep the job of porting from one UNIX system to another such a pain.

Anyway, we're not all System V. BSD, Linux, and GNU need their own test suite—something to mark differences, to monitor adherence to standards such as POSIX, and to provide a regression test for updates. And unlike SVVS, such a GNU test suite shouldn't be just something a vendor invests in. It should be lying around the net for the edification of all users.

**"...wouldn't it be nice to start up a little test and have it tell you whether the utility does the things you expect?"**

Think about the idea of a comprehensive test suite. Before you write a script depending on the behavior of some utility, wouldn't it be nice to start up a little test and have it tell you whether the utility does the things you expect? If you are planning to

Continued on page 3

Continued from page 2

adhere to POSIX 1003.1, wouldn't it be nice to know how POSIX-conformant your system is?

Cygnus Support is organizing people to create just such a test suite. It's a big, ongoing job. As this test suite progresses, we'll have more and more basis for comparing and rating different UNIX systems. The suite will cover utilities and daemons, common libraries, and system calls. It will use Cygnus's free testing framework, DejaGnu, as the organizing principle.

The idea of a comprehensive UNIX test suite came up in an unusual way. I was exploring the idea of producing books about Linux, part of my job as acquisitions editor at O'Reilly & Associates. Linux distributions at that point, early 1993, were much less stable than they are today. One couldn't really tell what would be among the libraries and utilities from one release to the next.

I wanted to put out a version of our popular book, UNIX in a Nutshell, for Linux (This may indeed happen some day. We're concentrating on some other Linux projects first). I needed some assurance that at least SLS, the main distribution at that time, really worked the way we said it did. This doesn't mean that our current versions of UNIX in a Nutshell are thoroughly tested, indeed, somebody finds an error every once in a while. But dealing with something as loosely organized as the Linux development effort, I wanted to hold it to a higher standard. I wanted a test suite.

Traditionally, free software developers have not been very aggressive in pursuing testing. Indeed, a quality assurance mailing list about Linux failed to gain members and quickly died. Cygnus Support, however, understood the need for testing. They developed a large number of test cases for the GNU compilers and debugger they distribute (`gcc`, `g++`, and `gdb`). These test cases are written under the DejaGnu testing framework.

Cygnus released DejaGnu about the time I got interested in a Linux test suite. In March 1993, I contacted the developer of DejaGnu, an extremely energetic guy named Rob Savoye, and our project grew wings. Our first approach had been to Linux developers, but they were too busy with other tasks to start a formal test effort. So in April we posted an appeal on various newsgroups, Linux-related, BSD-related, and test-related, to volunteers who wanted to learn the DejaGnu system and contribute tests. A dozen responded, of whom several are still involved in the project. A few people donated large suites of tests written in various languages, which confirmed our suspicion that some of the work we needed had already been done and was lying in desk drawers waiting for our appeal.

The main burden of the testing is still mainly on Cygnus staff. Cygnus, which makes quarterly distributions of the entire GNU toolchain to support customers, is firmly committed to automated software testing utilizing the DejaGnu framework. Cygnus plans to continue expanding the scope of the test suites that they distribute with DejaGnu.

Andrew Oram is an acquisitions editor at O'Reilly and Associates. He spent ten years as a technical writer in the computer industry and co-authored the O'Reilly book "Managing Projects with Make." O'Reilly and Associates publishes definitive guides on UNIX, the X Window System, and a number of free software tools such as perl.

**Did You Know?**
*The Tactical Flag Command Suite for all U.S. Navy vessels was built with GNU C? TFCC is on all aircraft carriers, battleships and amphibious command ships. It is solely for the use of the embarked Admiral and his staff. It runs on Sun and HP workstations.*

**Did You Know?**
*Floppy and tape distributions of NetBSD, a free BSD UNIX compatible OS are available with complete source code for 386 and 486 PC's from Winning Strategies, Inc.*

Continued from page 3

## The Solution: DejaGnu

By Rob Savoye

As software authors take on larger jobs, software gets more complicated to write and debug every day. With this increase in program complexity comes the need for improved automation of software testing.

DejaGnu is a software testing utility designed to simplify the running of large collections of regression tests. DejaGnu provides both a utility to run the tests ('runtest'), and a framework in which new tests can be written.

POSIX 1003.3 is a new standard for test frameworks. DejaGnu is the ONLY POSIX 1003.3 conforming testing tool in existence that supports this interactive testing process. The standard defines how test results are displayed to the user.

DejaGnu's unique, standardized output display makes it easy to analyze test results. The advantage of these standardized displays is twofold: it makes it easy to write text processing filters to produce reports from the DejaGnu output files, and it makes it easy for humans to read those output files. Quick recognition of what happened during the test run speeds up the fixing and retesting cycle.

DejaGnu is based on a well-known scripting language, Tcl, and a C-based application, Expect. Tcl is a C-language library, developed by John Ousterhout at the University of California at Berkeley, that provides a scripting language which can be "dropped in" to control any application. Expect, developed by Don Libes at the National Institute for Standards and Technology, is a program that provides automated control over application programs that were designed to interact with humans. Expect uses Tcl as a scripting language: Tcl commands control the starting, stopping, and input of the application program. Other Tcl commands match up the application's output against sets of expected responses and timeouts. The match results control the flow of the Tcl script. Tcl also allows the user to define new functions in a script as if they had been built-in, to read and write files, and to access other system facilities.

DejaGnu itself is a collection of functions that are written in Tcl and loaded into Expect. Test cases are created entirely in Tcl, using Tcl commands provided by Expect for process control.

DejaGnu's most obvious use is to test other software running on the machine where DejaGnu is running. For example, DejaGnu can run on a Sun SPARCstation and test a program that is running on the Sun. Most testing frameworks provide this capability; indeed, many useful test frameworks are simply Unix shell scripts.

DejaGnu stretches this paradigm to include testing of software that runs on other machines, including embedded systems. As a result, DejaGnu is particularly useful for cross-developers and embedded system designers. Test cases written in the DejaGnu framework can control execution, input, and output matching for programs that run on remote target boards as well as on the local machine. DejaGnu is an automated communications application which can access remote machines, download programs, execute them and examine the results. This allows, for instance, running tests on a Sun which check that an embedded application running in a 386 board provides correct responses to a wide variety of inputs.

In this way, DejaGnu can test systems of programs that include embedded components, such as a medical imaging system. If such a system contains a Unix application and embedded hardware for machine control, DejaGnu handles both. The application and the

Continued on page 5

**MAUG**

**The Free Newsletter of the MIPS Architecture User's Group**

This newsletter is published to provide an uncensored, vendor-independent forum for discussing matters of concern to programmers building MIPS-based embedded systems. It consists entirely of submissions from subscribers, who are encouraged to share information that would be of use to other MIPS users to ask questions. This newsletter is free and is distributed by Philip Bunce pjb@carmel.portal.com or 408-625-1247. To subscribe send email to maug@carmel.-com; subject "info". All legal submissions containing non-proprietary info that are received in pure ASCII machine-readable form will be accepted for inclusion. This can include submissions from vendors describing features of their products, but please keep them technical.

Continued from page 4

embedded firmware are started up, run and controlled in parallel, and problems in their interaction show up immediately.

At Cygnus Support, we use DejaGnu to control the GNU C compiler in compiling small test programs, control the GNU debugger to download the test programs, then test not only that the test program was compiled and downloaded correctly, but also that the debugger itself properly debugs the embedded application. These tests are run "native" on almost a dozen workstation hosts, and "cross" from these hosts to half a dozen embedded target boards, to provide quality assurance for Cygnus's quarterly software releases. We wrote DejaGnu because we didn't see any other way to fully test forty configurations of our development software in a new release every three months.

For some people, the most notable feature of DejaGnu and its test suites is that they are freely available. This not only contributes to DejaGnu's popularity, but also provides a continual stream of improvements to DejaGnu itself. No other regression test available today offers the versatility and flexibility that comes built-in to DejaGnu.

Many parts of the GNU system come with freely available DejaGnu test suites, including the C Compiler, the C++ compiler, the CHILL compiler, the debugger, the assembler, and the binary utilities. These test suites can be used to test any similar program (or a new port of such a program), whether it's a freely available program or a proprietary one. Since the test suites are freely available to everyone, new tests are being contributed all the time by grateful users, and the suites become both more generally useful, and more complete, over the years.

DejaGnu has come to the rescue for many software developers. Manual testing is now a thing of the past. Cross developers and embedded system designers can spend more of their time developing and designing. More importantly, embedded products can be completely tested at a low cost before they are released, rather than having customers find the problems after release.

You'll still be eagerly welcomed if you want to participate in this historic effort to place the Unix universe under the regimen of regression testing. The project has been termed —FROST— FRee OS Testing project. Our goal is to develop a free conformance test suite for the POSIX.1 and POSIX.2 standards.* To join the Internet mailing list, send mail to listserv@-cygnus.com. Include the text "subscribe FROST".

> ***Did You Know?***
> *Cygnus supports over 70 platform configurations of the GNU compilers, fully tested by DejaGnu. DejaGnu executes 8,000 test cases in 16,000,000 documented tests for a typical release. New tests and test cases are being added every quarter.*

* POSIX.1 covers all the system calls. POSIX.2 covers all the file utilities, many of them interactive.

# Practical Application:
## Testing Parallel Fortran Performance with DejaGnu

Michael Gerlek Interviewed
by Terri Thather

The following is an interview with Michael P. Gerlek, a third year graduate student at the Oregon Graduate Institute, completing his Ph.D. in Computer Science. Michael is part of a research team using DejaGnu.

**Q: Tell us about the project that led you to look into using DejaGnu?**
A: I work under Dr. Michael Wolfe as part of his research team, the Sparse Group. Our group currently has several Ph.D. students and a visiting researcher from Japan.

We're doing research in the field of high performance compilers—that is, advanced, state-of-the-art compilers for high performance architectures, ranging from high-end uniprocessor workstations to highly parallel computers. Some of the areas we're currently working in include languages and optimizations for parallel computers, dependence analysis and advanced loop transformations, back-end optimizations for superscaler RISC chips, and inter-procedural analysis and optimizations.

Our research test bed is Nascent, a Fortran 90/HPF compiler written in C++. I'm de facto "toolsmith" for the project and have the responsibility for maintaining some semblance of version control, installing new software (e.g. `gcc`), playing with makefiles, and so on.

Although it will only be of inter-

Continued on page 6

Continued from page 5

est to researchers in the field, we intend to release the source code to Nascent at some point in the future.

**Q: How did you hear about DejaGnu? Why would you trust that it was quality software?**
A: Several months ago we had talked about needing regression testing for Nascent, so I was keeping my eye out for something to use. We don't have the money for commercial packages, and wouldn't be interested even if we did: we're a small group, and we prefer to be able to tinker with our own tools as needed. I don't have the time for, nor interest in, writing my own from scratch.

I heard about DejaGnu in the usual way—I saw a reference to it on the Internet. Since I was already familiar with Cygnus I had no qualms about its quality. The FSF produces very good tools that are supported, improved, widely used, and well documented. We already use quite a bit of FSF software here at OGI: `gcc, Gnu make, emacs.` Now that I think about it, we use piles of free software of one sort or another in our group: `bash, gnu tar, gzip, f2c, xv, mosaic...` one takes it for granted. The stuff's just as good as anything vendor supplied, and often better. Plus, I had heard Cygnus was using DejaGnu internally—obviously they wouldn't use it if it wasn't up to snuff.

**Q: After using DejaGnu did you find that it measured up with what you expected from a regression test? In what practical application did you use it?**
A: I'm currently developing Nascent's code generator and I have two ways of checking my progress as I add more code and more features. One is to inspect generated assembly code by hand—a hassle, but necessary. The other is to execute the generated code and examine the output it produces. Since I'm only interested in classic Fortran 77 code, I can use a known compiler (Sun's f77) to also generate output and use that as a baseline. With DejaGnu, I can automate this: for each file in a set of Fortran programs, my test script runs Nascent, runs the resulting program, runs f77, runs that resulting program, diffs the outputs, and reports back.

I found it fairly easy to get going. I'd never seen any Tcl before, but that wasn't a problem. I began by pattern matching off the test suites that are available for `gcc`.

As I get more experience with this in the next couple months, I'll write a full set of scripts to do classical compiler regression testing for the whole group. This will simply run Nascent over a (large) set of inputs, covering each of Nascent's different analyses. This will just be an automated way of making sure that nothing suddenly, and perhaps very subtly, stops working. It'll be run in batch over night, I guess.

It has occurred to me that these scripts will make it easy to collect performance data for Nascent. To investigate the effects of the compiler optimizations we're developing, we run the compiler over various inputs and collect lots of statistics. Without too much work, DejaGnu could provide us with some hooks to automate this data collection process for large input sets.

**Q: What platform did you use it on?**
A: Our principal development platform is a network of Sparcstations running SunOS. We started the project using `gcc` on the Suns so we wouldn't have to worry about eventual portability to other environments. As I said, we plan to release Nascent to others in the compiler community someday.

Several of us have recently starting running Linux at home on our PC's. I was pleased to discover Nascent compiled and ran on my 486 with virtually no changes needed, courtesy of `gcc`. When I first got DejaGnu, I installed it at home and played with it there first. After I got a

Continued on page 7

**ANDREW CONSORTIUM UPDATE**
The Andrew Consortium* Newsletter is available via anonymous ftp from:
`emsworth.andrew.cmu.edu` (128.2.203.197) as file `newsletters/93Dec.d (or newsletters/ASCII/93Dec.ascii)`
**Topics include:**
Release plans
6.1 released to members
6.2 will be the last C release
7.0 is the first C++ release
**New and revised features in 6.1**
Dynamic loading and C++
Printing directly to post script
Widgets are Just Resource Bundles
CIL and OpenDoc
* The Andrew User Interface System is widely used in research labs as a base for user interface development. It's recognized for its open architecture and extensive set of existing high level media objects, especially text. The Andrew Toolkit has been the basis for at least two commercial products, Raport and the R/S6000 info system.

Continued from page 6

test suite prototype running, I just moved it over to the Suns, no problem.

**Q: What strong/weak points do you see in DejaGnu?**

A: One strong point is that it's Tcl based. That makes it easy to do many different sorts of testing from one big framework. We're just using it as a batch testing tool, so its pretty simple to write a test script. For interactive testing, though, DejaGnu and Expect make a powerful combination. Another strong point is the obvious, it's supported and maintained software from a very reputable source. Example: I had a problem with an overly-aggressive error reporting function, and sent Rob (Savoye) email. I quickly received a reply explaining what the problem was (he already knew about it) and a patch. That's the way things often work in the better regions of the free software Unix community.

A weak point is that it's tied to 'configure'. For sites that don't use configure already, this will be a bit unfamiliar. I think it tends to obscure the principal aspect of writing the testsuites themselves. Once its in place, though, it's simple.

The manual still needs work too: the provided example could be expanded quite a bit. Rob has assured me that he's working on it. The manual does include appendices detailing Expect and Tcl; that's a big plus.

If you're considering using any software, free or commercial, for a serious long-term project, you want to make sure of several things: it will be maintained and improved for the foreseeable future, it is well documented, you're not the only one using it, and so on.

The term 'free software' has a bad connotation. It's unfortunate that things like `gcc, perl`, the Linux kernel, and so on are called "free software". I prefer to think not of how much it costs, but who wrote it. There are many people in the Unix community who have written quality programs and have donated them to the net. These people have well-known reputations, and you know you can trust their software.

I'm not into the whole GPL controversy. I honestly haven't been convinced one way or the other. But I have no problem using free software under whatever terms it comes with.

**Q: Do you see free software as the wave-of-the-future?**

A: No, but I do expect to see the importance of open source software to continue to increase and I look forward to it. I'm writing these comments using emacs under X11 on my Linux'ed 486. Tomorrow I will use a commercial product to prepare slides for a presentation, simply because the quality and fucntionality of that product is much better than anything else I've seen. I think there's room enough for both types of software.

### The Fine Print



**More Reading On Free Software and Related Topics**

**"The Meaning of Freeware"**
Rich Morin
Unix Review December '93
**"Surviving Software Testing"**
Carl Dichter
Unix Review Vol 11 No. 2 '93
**"What's GNU?"**
Jack Woehr
Embedded Systems Programming Magazine
January '94, February '94
**"Operating Systems"**
SunExpert Magazine
October '93
**"Cruising the Freeway"**
Embedded Systems
Programming Magazine
February '94

## Access Information

**Andrew Consortium**
Carnegie Mellon University
5000 Forbes Avenue Smith Hall 106
Pittsburgh, PA 15213-3890
+1 412 268 6710 voice
`info-andrew-request@andrew.cmu.edu`
**Cygnus Support**
1937 Landings Drive
Mountain View, CA 94043
+1 415 903 1400 voice
+1 415 903 0122 fax
`info@cygnus.com`
**Free Software Foundation**
675 Massachusetts Avenue
Cambridge, MA 02139 USA
+1 617 876 3296 voice
`gnu@prep.ai.mit.edu`
**Don Libes**
National Institute of Standards and Technology
Gaithersburg, MD
For a copy of Expect, send email to
`library@cme.nist.gov`
**MAUG**
100 Dolores St. Suite 242
Carmel, CA 93923
408-626-4068 voice
`maug@carmel.com`
**O'Reilly and Associates**
103 Morris St. #A
Sebastopol, CA 95472
+1 707 829 0515 overseas/local
800 998 9938 US/Canada
+1 707 829 0104 fax
**John Ousterhout**
Computer Science Division/Dept. of EECS
University of California
Berkeley, CA 94720
+1 510 642 0865 voice
+1 510 642 5775 fax
`ouster@cs.berkeley.edu`
**Winning Strategies, Inc.**
61 Crestwood Drive #18
Daly City, CA 94015
`jtc@wimsey.com`

The Free Software Report is published quarterly. A 4-issue subscription is available for $60.00 payable in U.S. Dollars through a U.S. bank. Send a check or money order to: **The Free Software Report, 1937 Landings Drive, Mountain View, CA 94043, USA**

Please supply your name and title, company name, and address for receiving the Free Software Report. The most recent and subsequent issues will be mailed as soon as we receive your order.

Name____________________ Title____________________

Company or Organization________________________________

Address________________________________________

City__________________ State__________ Zip __________

Country__________________ Amount Included: $__________

Comments ______________________________________

________________________________________________

e-mail__________________________________________

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**The Free Software Report**
**1937 Landings Drive**
**Mountain View, CA 94043**

**BULK RATE**
**U.S. POSTAGE**
**PAID**
**MOUNTAIN VIEW, CA**
**PERMIT NO. 521**