Configuring the Software Development Process on Linux

Dr. Adam Kolawa

ParaSoft Corporation
2031 S. Myrtle Ave.
Monrovia, CA 91016

Tel: 888-305-0041
Fax: 626-305-3036
Ak@parasoft.com

Gone are the days when you had the luxury of spending several years in development. Nowadays, your applications must be ready in two or three months. Put simply, if you don't have an effective development process, you will not survive.

Linux is an ideal software development platform for C, C++, and Java because it removes many of the obstacles typically associated with software development. Obviously, it's a good idea to catch errors before they crash a machine. Linux has the stability to meet this requirement. Because the kernel can be easily customized, you can prepare development machines for your specific needs. And often you'll find that Linux requires less horsepower and hardware than other OSes. All this means your code can be written and tested on Linux, then moved to another plat-form for deployment.

Fortunately, many of the tools you need come with Linux, and you can assemble a professional software development environment quite inexpensively. But first, you should understand the necessity of configuring a development process that not only produces software, but effectively controls errors while doing so.

Mention software to most people, and the word “bugs” comes to mind. Even computer neophytes are encouraged to believe that, like the common cold, bugs are inevitable. It's simply not true. Errors are caused by bad coding and development practices.

In a bad development process, bugs are not controlled; rather, they are introduced, then ignored until the final stages of the development process. This is more dangerous than most people realize: when you allow bugs to enter and remain in your code, the bugs build upon and interact with one another. This interaction usually has the critical effect of causing bugs to increase exponentially instead of increasing linearly with time and number of lines of code.

When this happens, the amount of bugs will dramatically increase as time and lines of code increase-- often to the point where they cause the project to be cancelled. The key to controlling bugs is preventing them from increasing exponentially. You can do this by integrating error-prevention and error-detection practices into your development process.

Study after study has confirmed that:

• Focusing on error prevention results in higher quality, shorter development schedules, and higher productivity.
• The longer a defect remains in the system, the more expensive and difficult it becomes to remove.

Moreover, one study done by Microsoft showed that it takes an average of 12 programming hours to find and fix a programming error. Another study by DeMarco and Lister found that professional programmers average 1.2 software defects for every 200 lines of code they write. Debugging commonly takes 60-70% of the overall development time. Do the math.

You'll find that letting errors enter and remain in your code will cost a lot of time and money. It's best to prevent errors in the first place and remove any existing errors as soon as possible, before they spawn more errors.

One critical part of implementing a software development process that controls errors is establishing an infrastructure that supports such a development process. We will now discuss how you might do this on Linux.

Source Code Repository
First of all, you need to establish a source code repository so developers can work off of a common code base. One of the best ways to do this on Linux is to configure what looks like a simple LAN, where each developer is connected to a Linux server which hosts a central repository for the entire source base. Development machines can be configured as needed, running Linux or Microsoft OSes, for example. Developers access the central repository from these machines via SAMBA, X-Term, dial-up access, or a Web browser. Where development is aimed at a specific version--or several versions--of an Intel-based OS, VMware can be used to instantly load the required environment on a development machine for testing, while retaining Linux as a “host” OS.

The software you select for this repository must provide a record of the evolution of the source base--the standard who, what, when, where, why questions from Journalism 101. With this capability, you significantly reduce the risk of change, making it possible to recover older, more stable versions. At the same time, it becomes less expensive to try different approaches to your goals. Tools in this category include GNU Revision Control System, GNU CVS (Version Control Sys-tem), and Rational Software's ClearCase (www.rational.com). (You'll find RCS and CVS in most Linux distributions. For a full list of GNU software and how to obtain it, see http://www.gnu.org/software/software.html#DescriptionsOfGNUSoftware.)

Nightly Build
Once you have a central code repository, you can establish an automatic nightly build of your application. A nightly build is a completely automated process that rebuilds the application over-night, every night, without any human intervention. At the same time each night, the nightly build should check code out from the code repository, rebuild the application, and run all available test suites. When the developers come in every morning, they should see a report that contains any failed test cases, and be able to immediately assess project progress and quality. Nightly-builds are usually set up with in-house scripts.

Bug-Tracking System
A bug-tracking system such as GNU GNATS (www.gnu.org/software/gnats/gnats.html) or Mozilla.org's Bugzilla (www.mozilla.org/projects/bugzilla/) has two main uses. Most important is to record and track all errors not detected by your test suite. The system should let staff report bugs, correlate them to source versions, and assign them to the appropriate developers. Loyally entering every bug found into the system facilitates problem tracking and provides valuable data about the types of errors that teams or developers tend to make-- data that can be used to hone error-prevention and error-detection efforts. The system can also be used to record feature requests that are not yet being implemented, as well as to prioritize, schedule, and track dependencies.

Automatic Development Tools
The most critical components of the development infrastructure are automatic development tools that help you prevent and detect errors as precisely, quickly, and easily as possible. These tools should be geared towards developers, not testers, and should provide the developer a simple way to start testing each piece of code as soon as it is compiled. The best way to achieve a high-quality application is for the developer to prevent and eliminate as many as bugs possible at each development stage. If developers do not constantly worry about bugs as they are developing, bugs left in the code will spawn more bugs, and when the application is finally tested, it will be considerably
more difficult to find and fix the bugs. In fact, when testing is delayed until the end of development, there is a good chance that bugs which could have been easily detected and fixed in the earlier stages of development will end up eluding tests and being passed on to customers. That’s why it is so critical for developers to use automatic testing tools from the moment that they compile each class or function.

Don’t just choose the cheapest or most popular error-prevention and error-detection tools. Take the time to evaluate different tools and choose carefully. The time spent evaluating tools can easily be regained if you find one tool that automates more processes than another, or prevents or finds more bugs than another. Compare how much user-intervention each tool requires. Look for features like automatic creation of test cases, harnesses, and stubs, easy ways to enter user-defined test cases, and automatic regression testing. Tools should be customizable, have interactive and batch mode, and integrate with other components in your development arsenal.

Conclusion

For your organization to survive, you need a development process that produces reliable applications as rapidly as possible. The first step in doing that is establishing a sound development infrastructure. The next step is making this infrastructure the backbone of a development process that controls bugs at every stage of development. Such a process would include practices like design reviews, code reviews, coding standards enforcement, unit testing, application testing, regression testing, and implementing “gates” that help developers assess when they are truly ready to leave one development stage and enter the next. For a discussion of how to implement such a process, see our “Debugging the Software Development Process” paper (available at http://www.parasoft.com/papers/dev_proc.htm).