Five Easy Pieces of Software Testing

By Steve Barry

So, you have decided to start testing.

Testing is an admirable undertaking. It is also a little bit daunting. Ideally, your test effort should mirror development every step of the way. It should cover every aspect of project work, from the initial project idea to the day it is removed from the users' computers. It should be infallible and uncompromising.

Testing should be all of these things, it usually isn't. Usually, it is difficult to impress upon the "Powers That Be" (management, users and even yourself) the importance of thorough testing. Not to mention, the seemingly monumental effort required getting a testing department up and running.

In order to ease the transition from untested development to proper testing practices, the following five "pieces" have been put forward as examples of what you can do right now.

Write up a broad test strategy
A testing strategy is a document that broadly describes your approach to testing. Keep it brief and open-ended. This document is not meant to be your testing bible, but merely your testing desk reference. It tells you what are important issues such as types of development, lists of software and hardware. It is the basic understanding that precedes testing. If you can, develop it in conjunction with management. This way you address their concerns, thereby getting their buy-in to the testing effort.

Assign testing duties to various members of your project team
Keep the assignments small; the intention is not to load them down with work, but rather to get them involved in the testing effort. Have somebody organize a walk-through and have someone else do an informal review of specifications on any existing documentation.

For existing software, set up an approval form
Give the approval form to the users to evaluate your products. This allows the user input in the revision of the software. It also gives the user the opportunity to understand your approach to testing.

Keep the information stored in any data method that makes sense to you. Over time you will develop these metrics to map the course of the development of the software. It will also help you predict the effort required for development of new products.

It is never too late to develop the requirements for software
Have somebody (preferably a user) review the existing products and write down what it "does". This way, any revision to the software can be mapped against its existing requirements. Make sure that both the developers AND users sign off on the requirements.

By developing the requirements for a project, even after it is released, you instill the importance of mapping out exactly what software does (or is supposed to do) from the beginning.

Look into a bug tracking system
I usually hesitate to recommend automation to a fledgling tester/test group. The one exception is good bug tracking software. When it is used correctly, it will never lead you astray. It gives you a platform for formalizing your bug definitions, severity and reporting structure.

This software does not have to cost an arm and a leg, either. In fact, many well-developed packages are available as shareware. If you are developing in UNIX or LINUX, for example, try gnat or wreq. Both are widely available on the Web.

Keep in mind that good testing is a full time job. Dedication and proper training, along with these simple techniques, will put you on the path to good training practices.