Five Things Test Engineers Should Know About Software

Author: Elijah Kerry
Elijah Kerry is a product manager for LabVIEW at National Instruments focusing on large, mission- critical development applications and software engineering practices. He holds a bachelor's degree in computer engineering from the University of Missouri, Columbia.

Modern test systems rely on software-based solutions to meet the needs of complex devices under test (DUTs) and satisfy demanding deadlines. As a result, many of the same development best practices and tools that are fundamental aspects of software engineering have become equally important for developing test applications. When developing software for a test system, keep in mind the following five basic practices to ensure you deliver a high-quality, reliable application on time.

1. If you are not using source code control, you are playing with fire.

Source code control is a fundamental tool for anyone who is developing software – no matter if you are a team of a hundred or a team of one. Without it, simple tasks such as sharing code or managing different versions can be difficult and pose risks that cause delays and lead to lost work. Software vendors offer numerous solutions ranging from Microsoft Team Foundation Server to Perforce to free open-source tools such as Subversion, any of which you can use with graphical code developed in NI LabVIEW software.

An advantage of having a source code control system is you can track, manage, and review changes to your application over time through a combination of diff and merge operations. With the LabVIEW Professional Development System, you can integrate graphical differencing and merge operations with source code control clients. Once set up, a diff or merge operation in source code control automatically displays a dialog in LabVIEW that steps you through the changes and identifies when it was modified and what was affected.

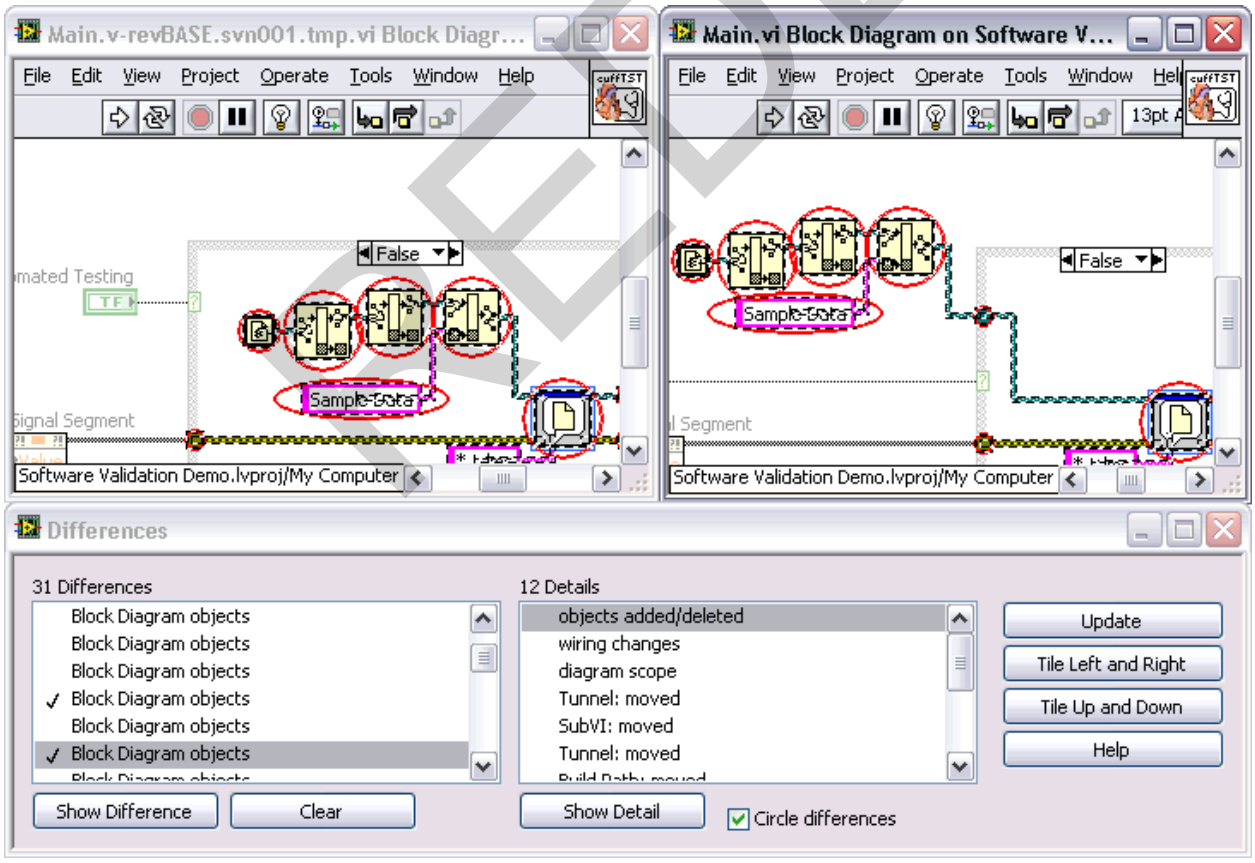

Figure 1. *You can invoke graphical differencing automatically from within source code control clients using the command-line interface that is provided with the LabVIEW Professional Development System.*

2. Development without requirements is prototyping.

Prototypes are an important part of the development process because you typically use them to demonstrate a crude idea or to prove the viability of a concept for new technology. However, the software you use in prototypes is often put together with little planning or regard for architecture, making it ill-suited for a final application. It is important to distinguish between the prototyping and development phases of the software life cycle – a key indicator is whether you have software requirements that include specifications for the overall architecture and a test plan.

Requirement documents are an effective way to align the expectations of customers with developers, coordinate large teams, document the status of a project, and ensure that code is thoroughly tested. Common tools for storing and managing these documents include Microsoft Word, Microsoft Excel, Adobe Acrobat, Telelogic DOORS, and RequisitePro. NI provides the ability to automate integration between these interfaces and NI software products such as LabVIEW and NI TestStand to automatically track requirements coverage and generate reports for traceability and upstream coverage analysis.

3. You can measure the testability and quality of code.

Static code analysis refers to any tool or method that has preestablished criteria by which it can compare source code to see if it meets standards for style, organization, and technique. In addition, static code analysis can help demonstrate that code is poorly written and identify problem areas. You can also use code complexity metrics such as modularity and cyclomatic complexity to determine the size and testability of a project. This is helpful when you inherit code and are asked to fix bugs or add features.

To track progress and find problems early, combine regular reporting of the code analysis metrics with frequent peer reviews. You can automate static code analysis of LabVIEW code with the LabVIEW VI Analyzer Toolkit, which offers the ability to customize more than 80 tests, including analysis of performance, complexity, documentation, and even spell-checking. A wizard is also available for creating new tests using LabVIEW VI Scripting.

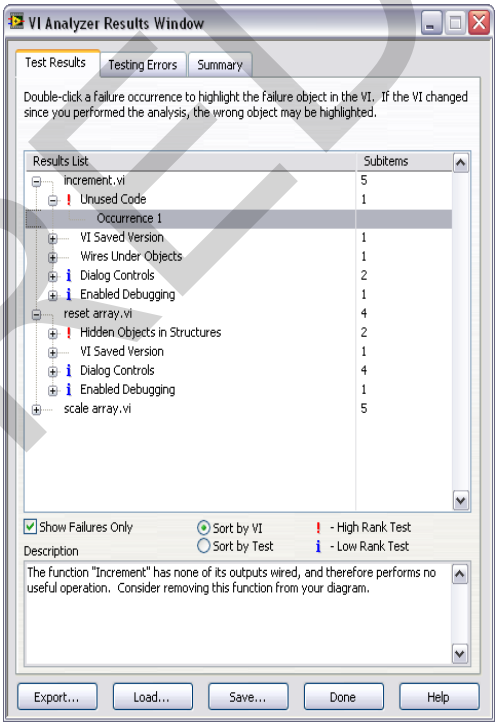

Figure 2. *VI Analyzer provides an interactive dialog and reporting tool for examining the quality and testability of VIs.*

4. You may think your code works, but you have to prove it.

You know when something is broken because it won't run. What is more difficult is proving to someone else or an external authority that something works correctly. The audience for this type of demonstration varies, but it could be the customer, a quality assurance group, or even a government regulatory agency.

Testing and debugging software is an inseparable part of the development process, but you can use automated tools, such as the LabVIEW Unit Test Framework Toolkit, to address the challenge of testing complex software. Automating this process reduces the amount of time you spend performing tests and makes more exhaustive testing possible. This not only helps ensure that the highest-quality software possible is produced but also saves money by catching problems earlier and reducing test time.

Functional code validation and testing is a well-recognized part of the software engineering process and standard practice for anyone who has to prove that the code works. Proving software works is more complex than showing that the application runs; it requires validating that it works correctly. This requires documentation and test results that demonstrate the application behaves in the way it was designed.

5. Reuse is not a myth, but it requires planning.

The growing complexity of test systems is converging with shortened release cycles for many DUTs, which has prompted a strong need for reuse libraries. Reuse means that both hardware and software can span multiple test systems and easily be adapted for new DUT iterations. It also means that separate development teams can use preexisting drivers and APIs to maximize efficiency and further decrease the programming phase of the life cycle. However, many programmers often struggle to incorporate practices to successfully reuse code – typically because of poor planning and the inability to adapt to changing requirements and easily introduce those changes to large numbers of programmers and applications.

A widely used example of a reuse library is the NI collection of more than 8,000 instrument drivers at ni.com/idnet. The success of these reusable instrument libraries relies on the clearly defined APIs for instrument communication and the encapsulation of low-level functionality in private libraries. However, the VI Package Manager offers a more sophisticated solution for distributing reuse libraries across an enterprise and managing the versions that are used across different projects.

For more information on these and other best practices for large application development with LabVIEW, visit **ni.com/largeapps.**