



LabVIEW Developer Days

Build Code. Form Communities. Gain Confidence.



Software Engineering for LabVIEW

High-Volume Production Test



Structural Health Monitoring



Medical Devices



Space Exploration



Large Physics Applications



Avionics Applications



Large System Development Powered by LabVIEW

Lawrence Livermore National Labs

Developed automated maintenance process for world's most energetic laser array at the National Ignition Facility using NI LabVIEW and PXI

- LabVIEW increased productivity by 3X over Java and C++
- Developed complex application consisting of over 1,000 VIs
- Applied software engineering practices to ensure quality



An overhead view of one of the main laser chambers

"The value in using the graphical dataflow language is the speed in which a team can deliver a robust solution while still using proper software engineering practices."

- Glenn Larkin, LLNL

Business Impact Case Study: Philips Home Healthcare Solutions

Accelerated Product Development

Full Regulatory Quality Compliance

Productive Test Platform Team



By implementing software engineering best practices and adopting the NI platform, the team increased its productivity by **347 percent** and reduced the company's cost of quality, saving it **\$4.5 million USD** annually.

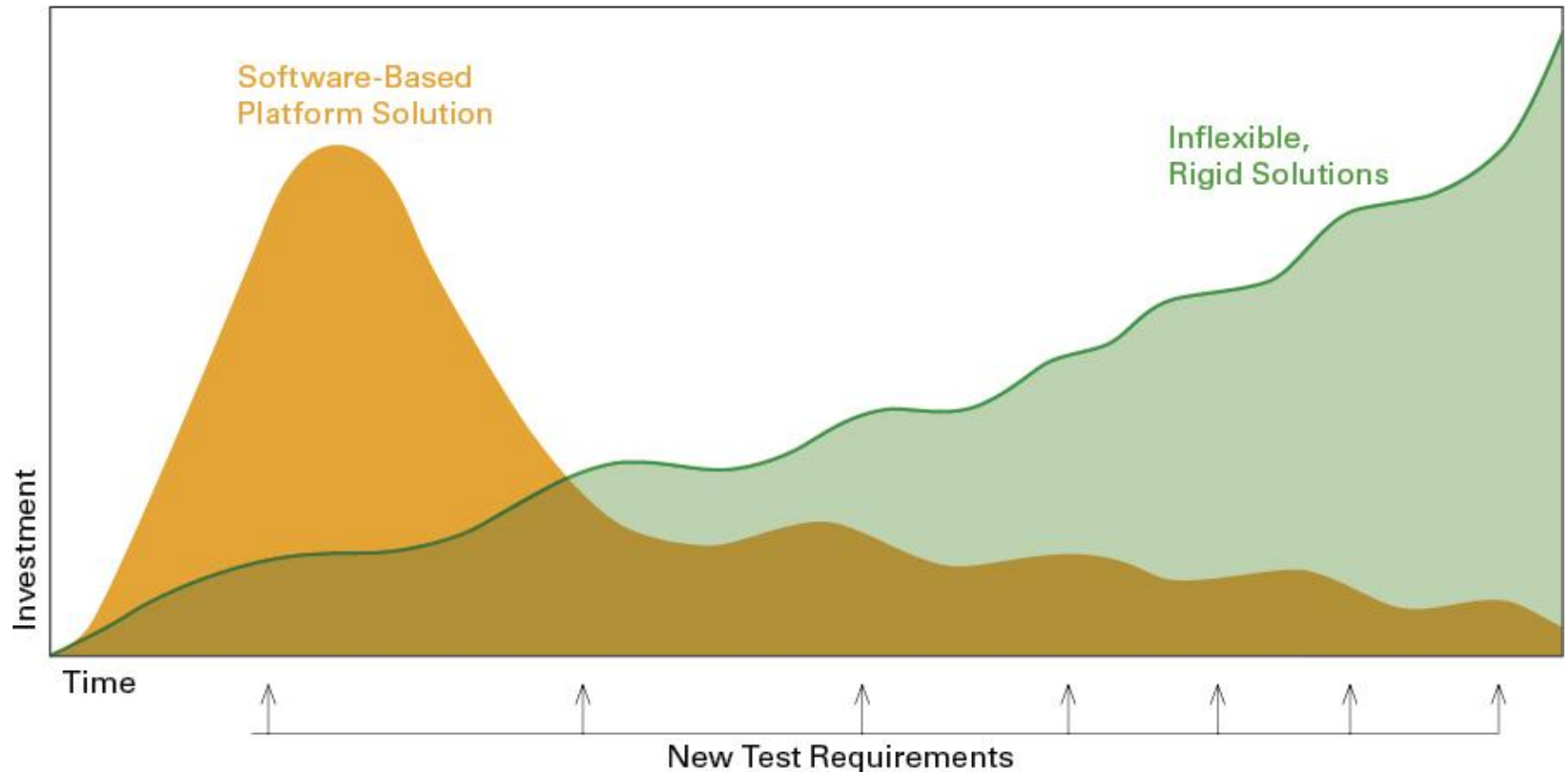
Philips HHS Accelerates Product Development and Decreases Test Costs by **81 Percent** Using the NI Platform

Goal is to Detect Defects Early and Manage the Risks Associated with Change

Development Phase	Cost Ratio
Requirements	1
Design	3-6x
Implementation	10x
Development Testing	15-40x
Acceptance Testing	30-70x
Post Release	40-1000x

Based on an analysis of 63 software development projects at companies including IBM, GTE and TRW

Investing In People and Processes

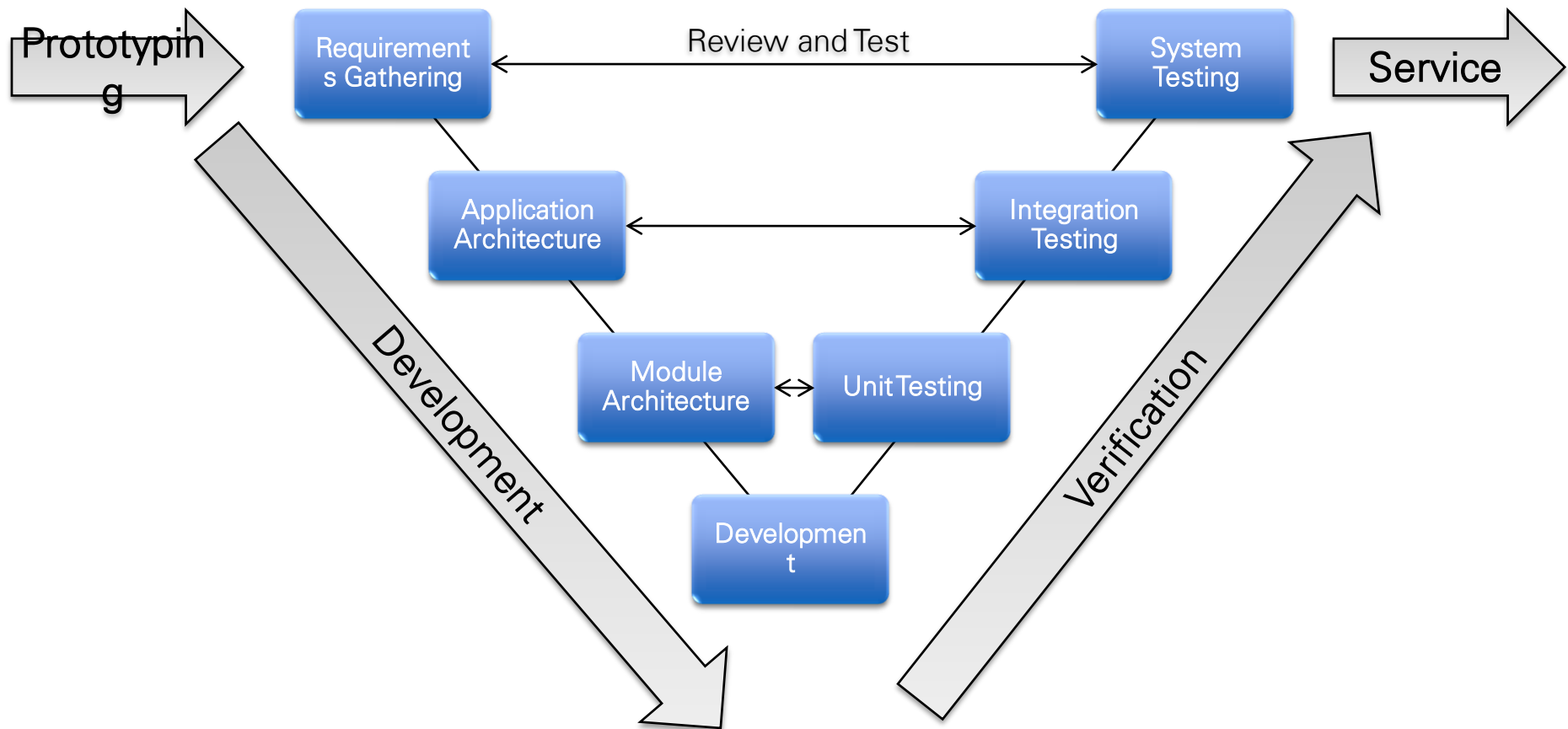


Examples of Software Engineering Debt

(just *some* of the most common LabVIEW development mistakes)

- ✓ No source code control (or Project)
- ✓ Flat file hierarchy
- ✓ 'Stop' isn't tested regularly
- ✓ Wait until the 'end' of a project to build an application
- ✓ Few specifications / documentation / requirements
- ✓ No 'buddying' or code reviews
- ✓ Poor planning (Lack of consideration for SMORES)
- ✓ No test plans
- ✓ Poor error handling
- ✓ No consistent style
- ✓ Tight coupling, poor cohesion

Software Engineering V-Model

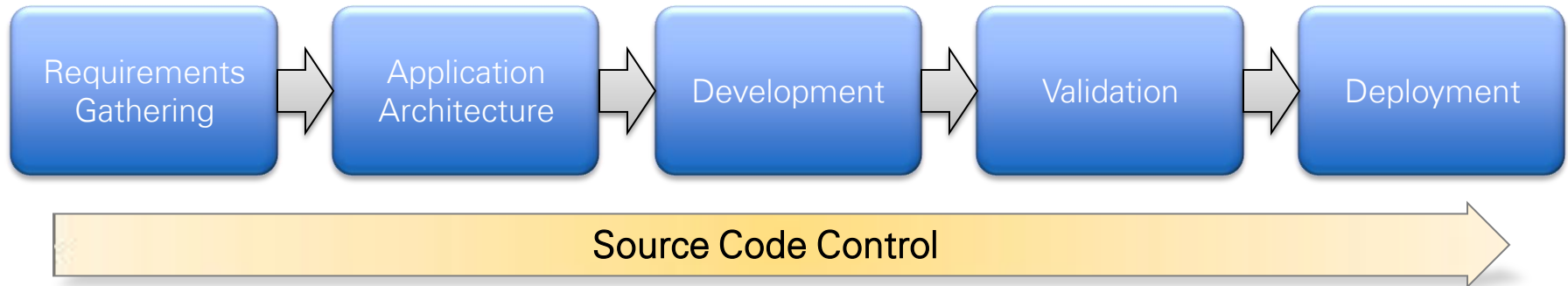


Agenda

Configuration Management and Source Code Control
Creating Reuse Libraries with VIPM
Requirements Tracking and Management
Static Code Analysis and Code Reviews
Dynamic Code Analysis
Regression Testing with the Unit Test Framework

Configuration Management and Source Code Control

Software Configuration Management Tools

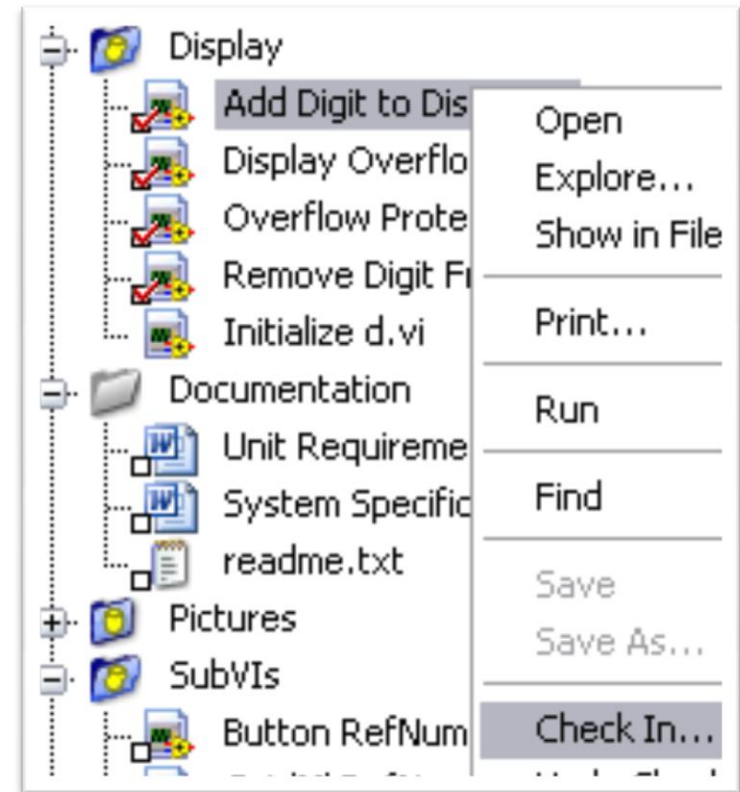


- ✓ Provides remote, backed-up repository of code
- ✓ Manage multiple versions and branches
- ✓ Track and identify changes and their impact
- ✓ Communicate and share code across a team
- ✓ Reduces pain throughout process

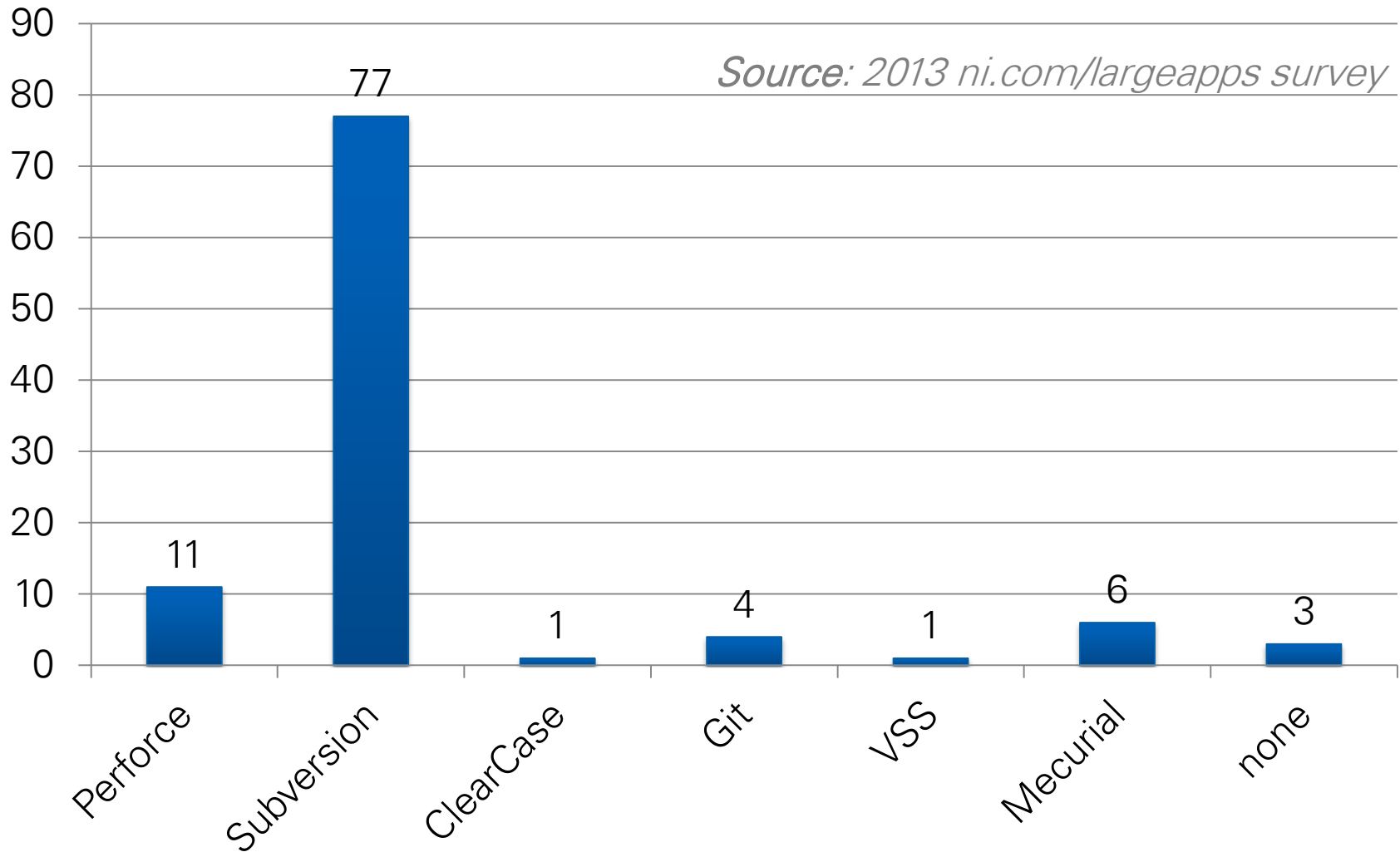
A Wide Variety of SCC Tools Are Available

Any of them can be used together with LabVIEW

- Perforce
- Subversion
- Git
- Mercurial
- Microsoft Visual SourceSafe
- Microsoft Team System
- Rational ClearCase
- PCVS (Serena) Version Manager
- MKS Source Integrity
- Seapine Surround SCM
- Borland StarTeam
- Telelogic Synergy
- ionForge Evolution



Popularity of SCC Options Amongst LabVIEW Programmers



Demonstration

Using Source code Control with LabVIEW

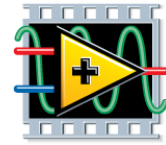
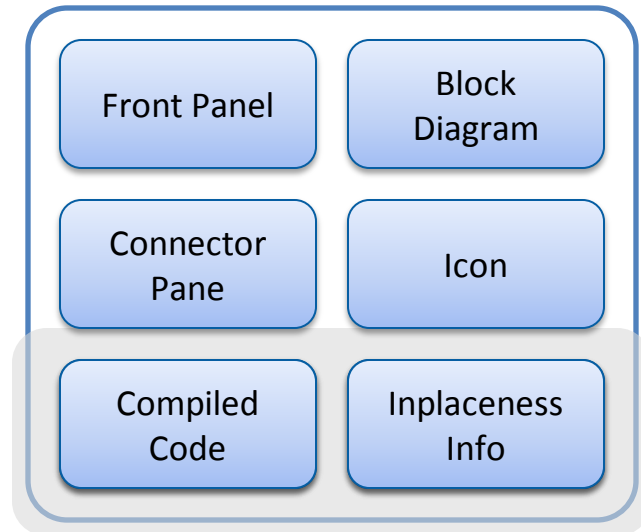
Separate Compiled Code From Source File

Improved Source Code Control Integration



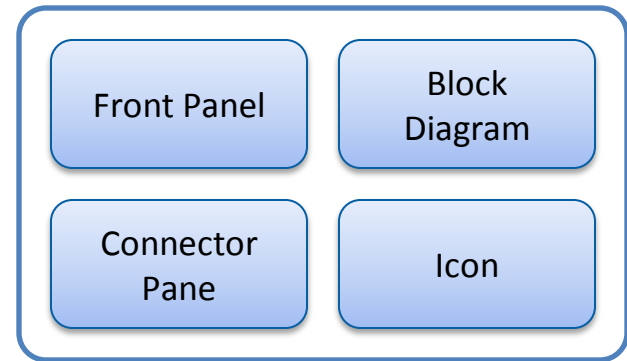
2009

.vi file format



2010

.vi file format*

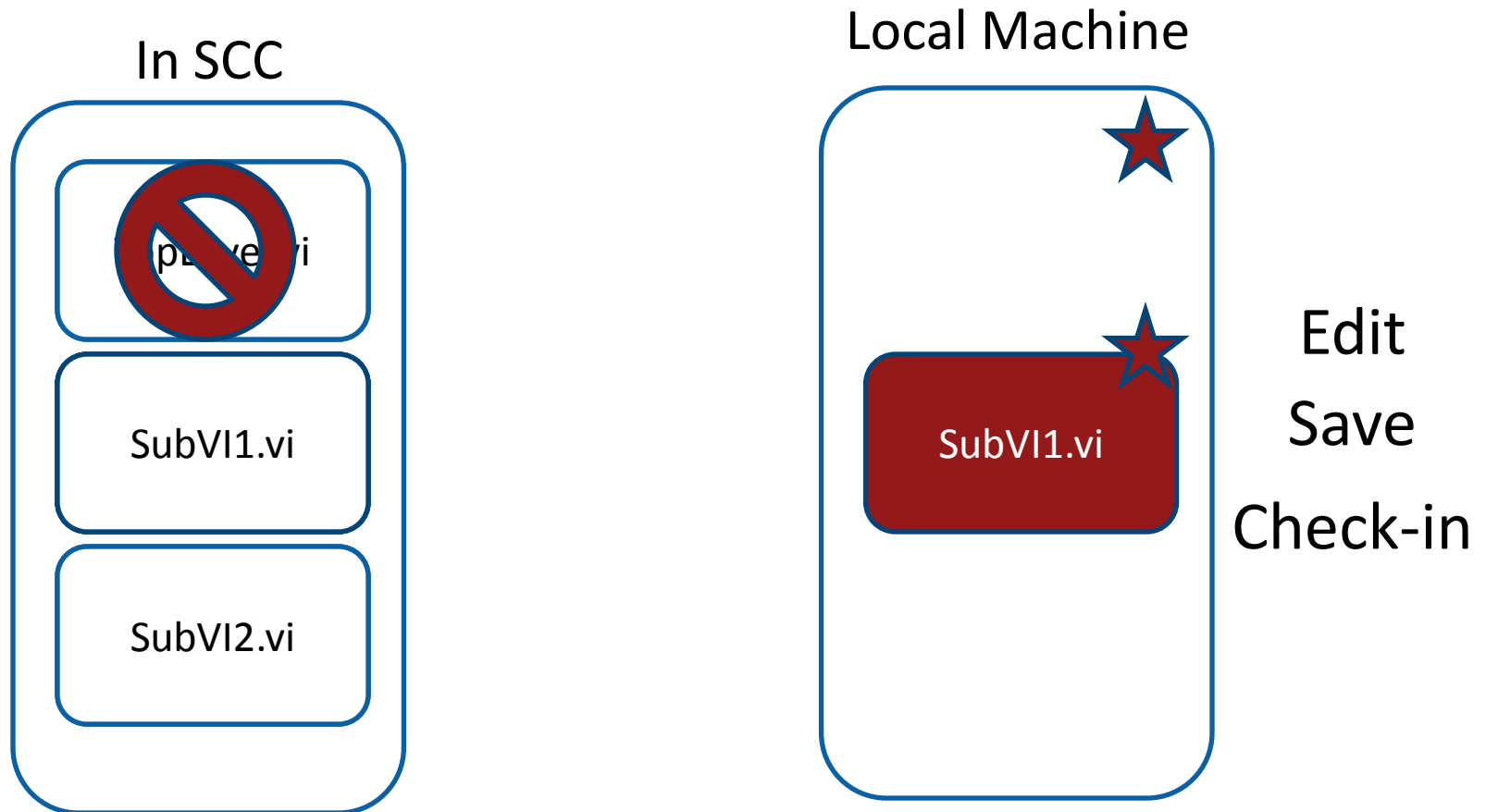


A separate object file is created to store and retain this information

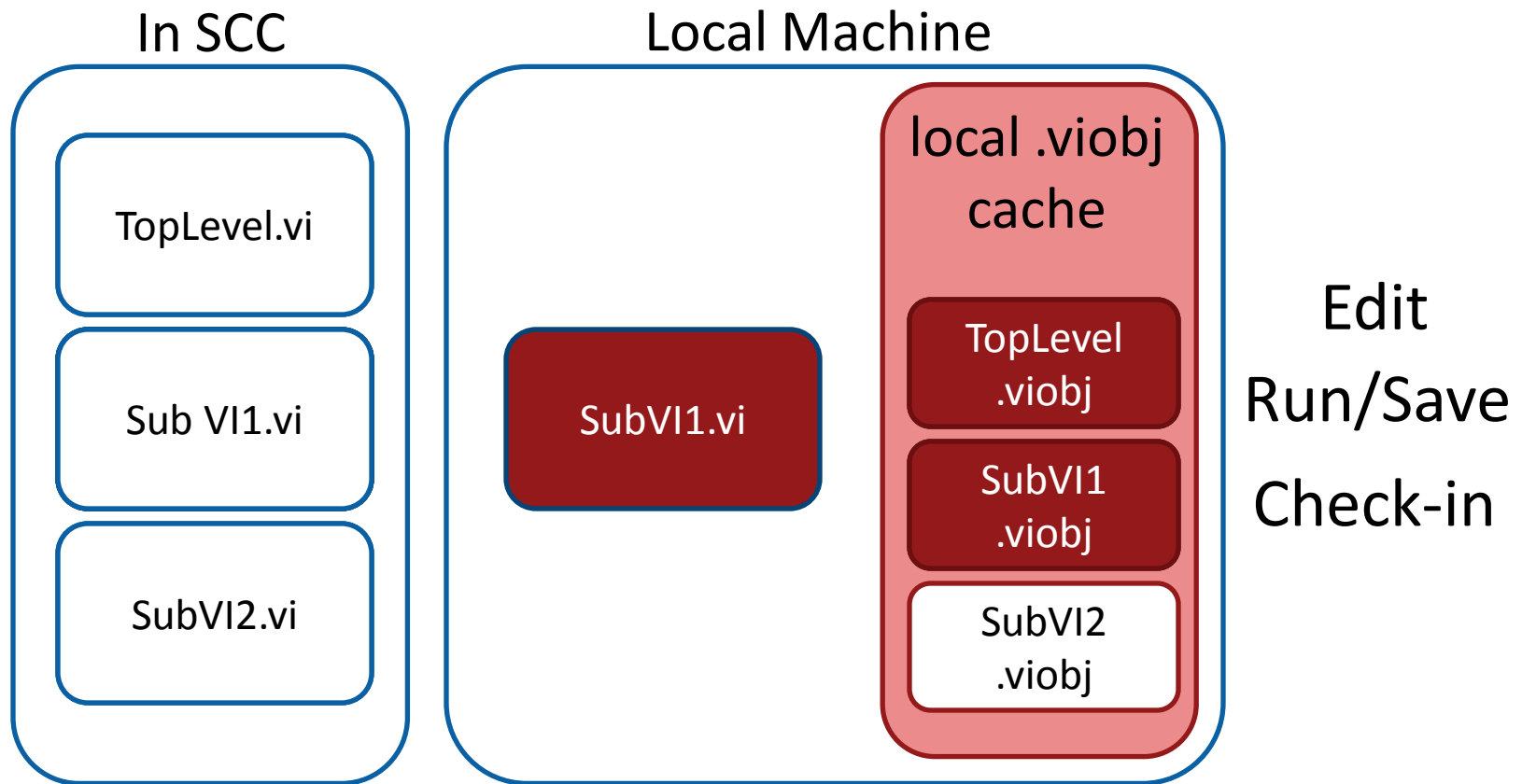
Eliminate the need to re-save and re-submit files to source code control unless the graphical source code has been changed by the developer

**this feature is not on by default and needs to be enabled from the VI Properties dialog*

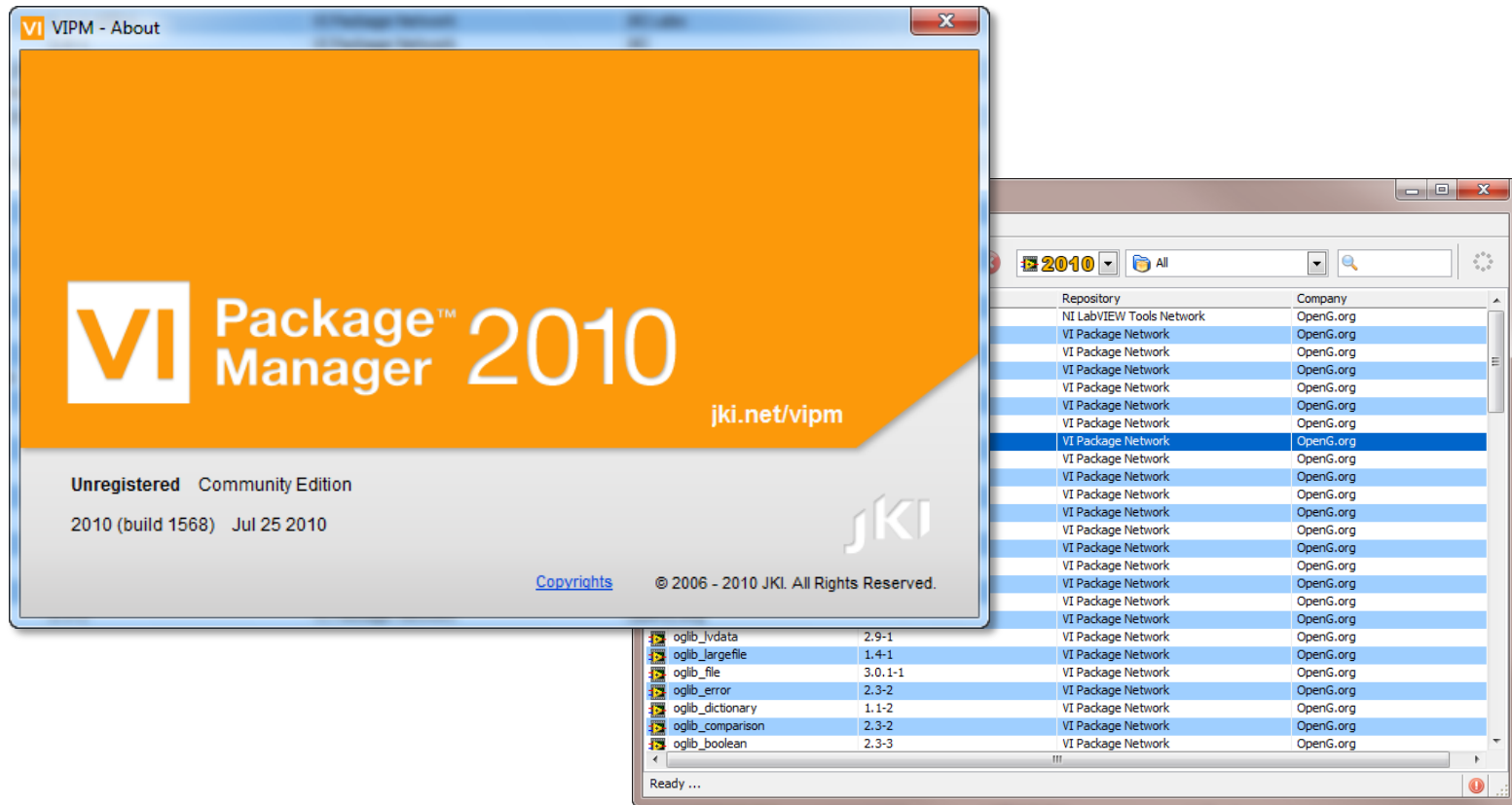
Source Code Control Scenario: Today



Source Code Control Scenario: 2010



VI Package Manager

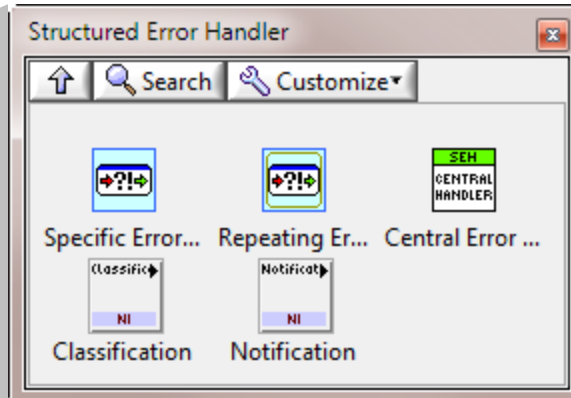


Build and manage packages of LabVIEW code

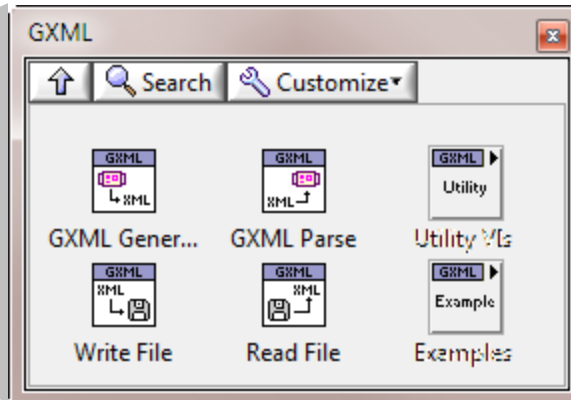
Install and Manage VI Packages



Structured Error Handler



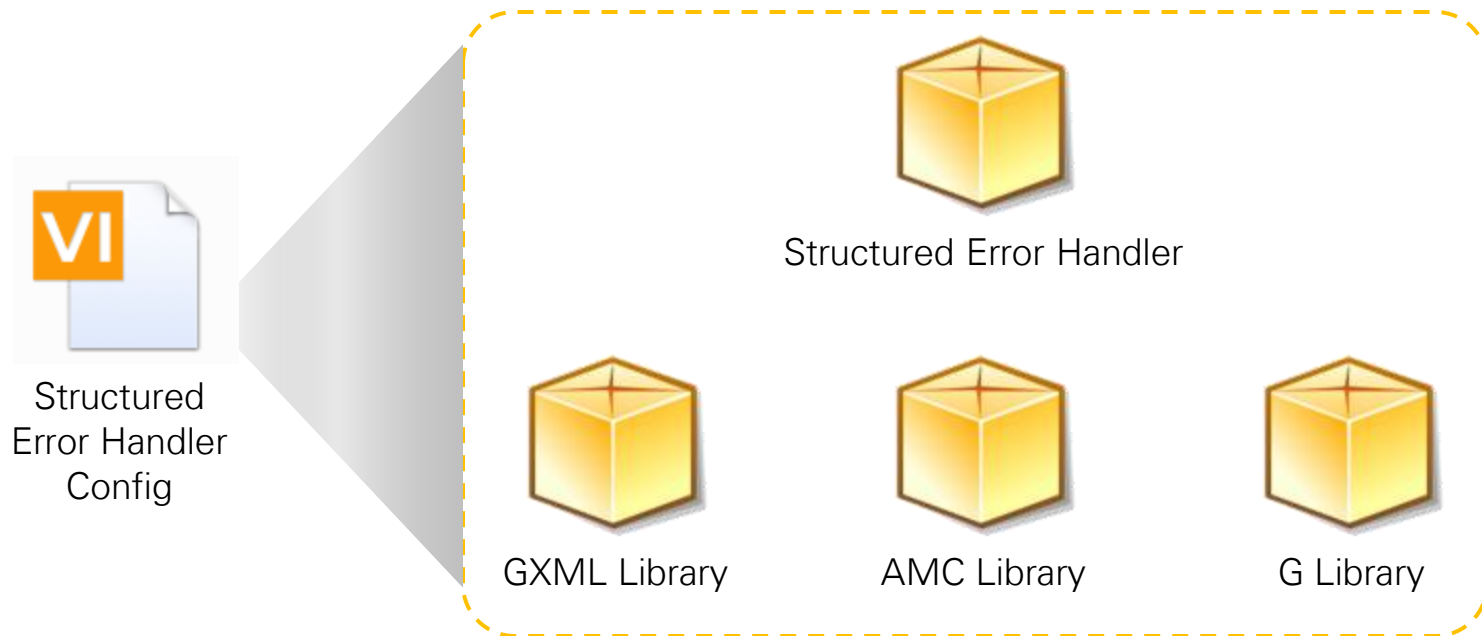
GXML Library



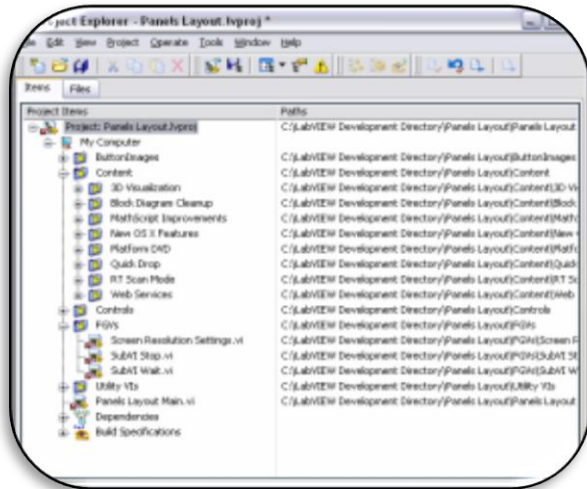
Create VI Configuration Files

A single file that contains multiple packages.

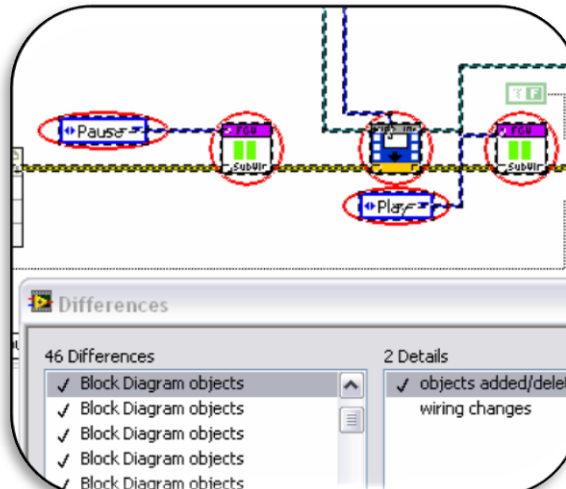
Easily share and distribute code that depends upon multiple libraries.



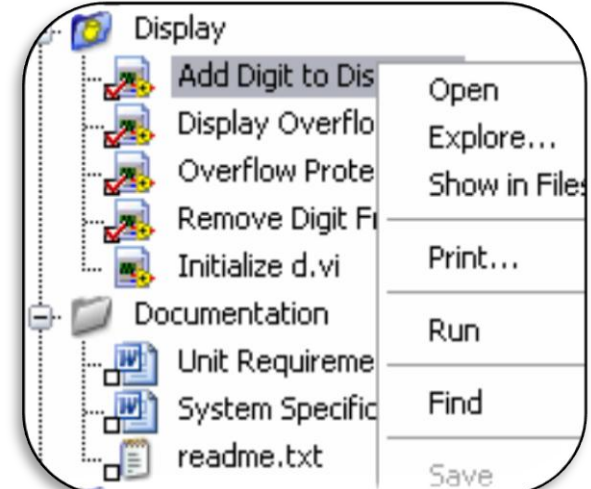
Software Configuration Management for LabVIEW



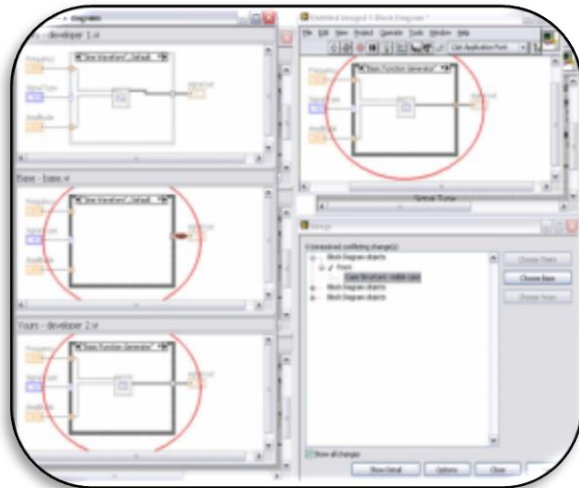
System Level View



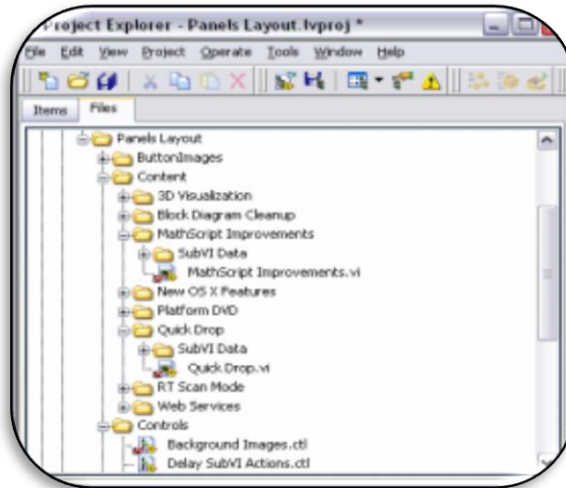
Track Changes



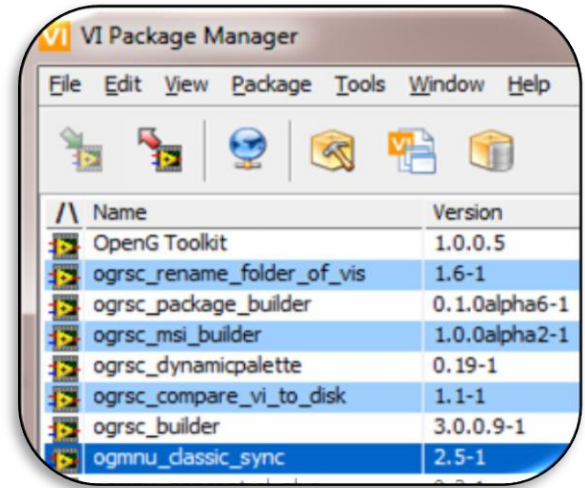
Integrate with SCC



Merge Graphical Code



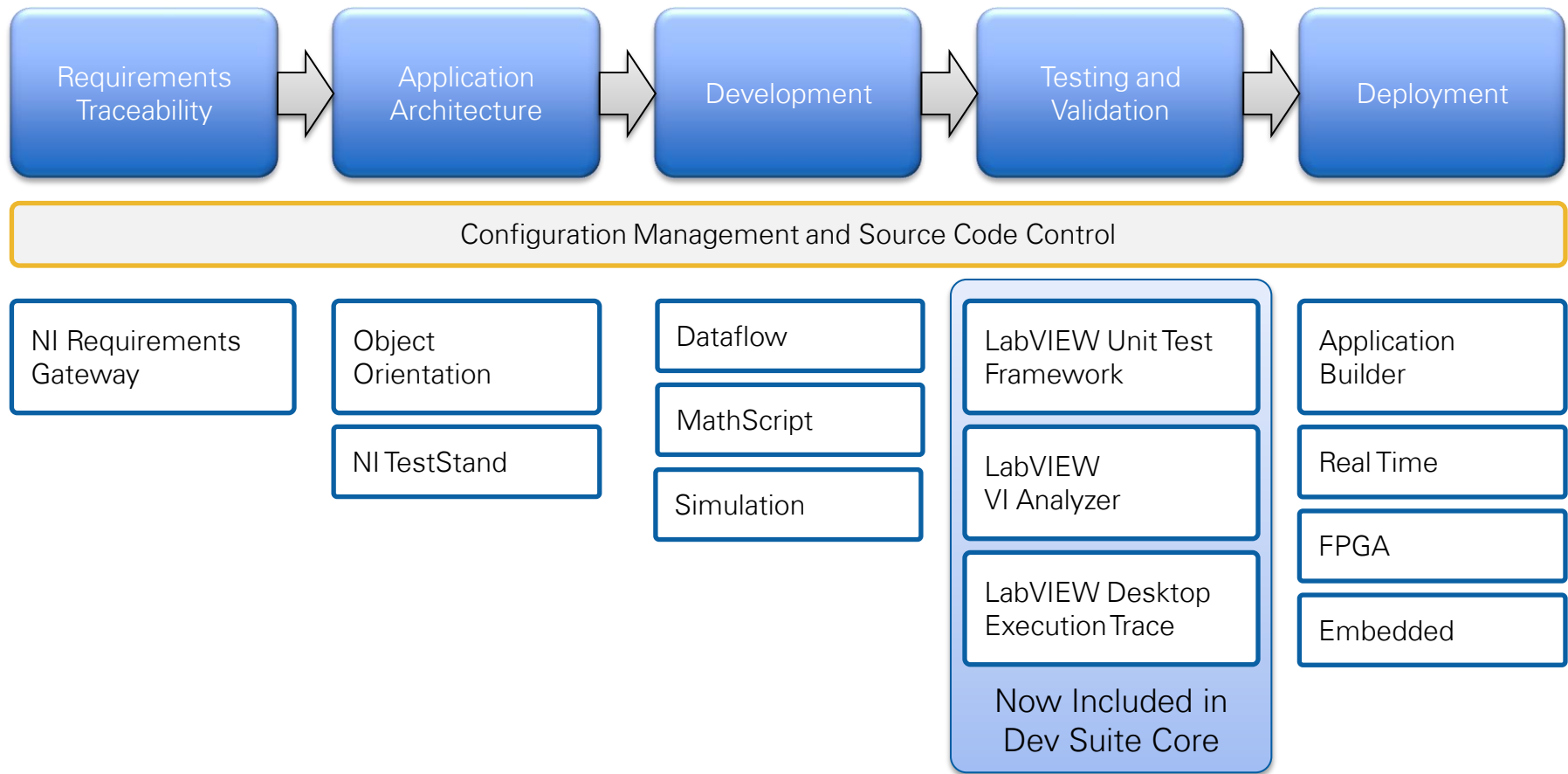
Manage Files and Links



Manage Reuse Libraries

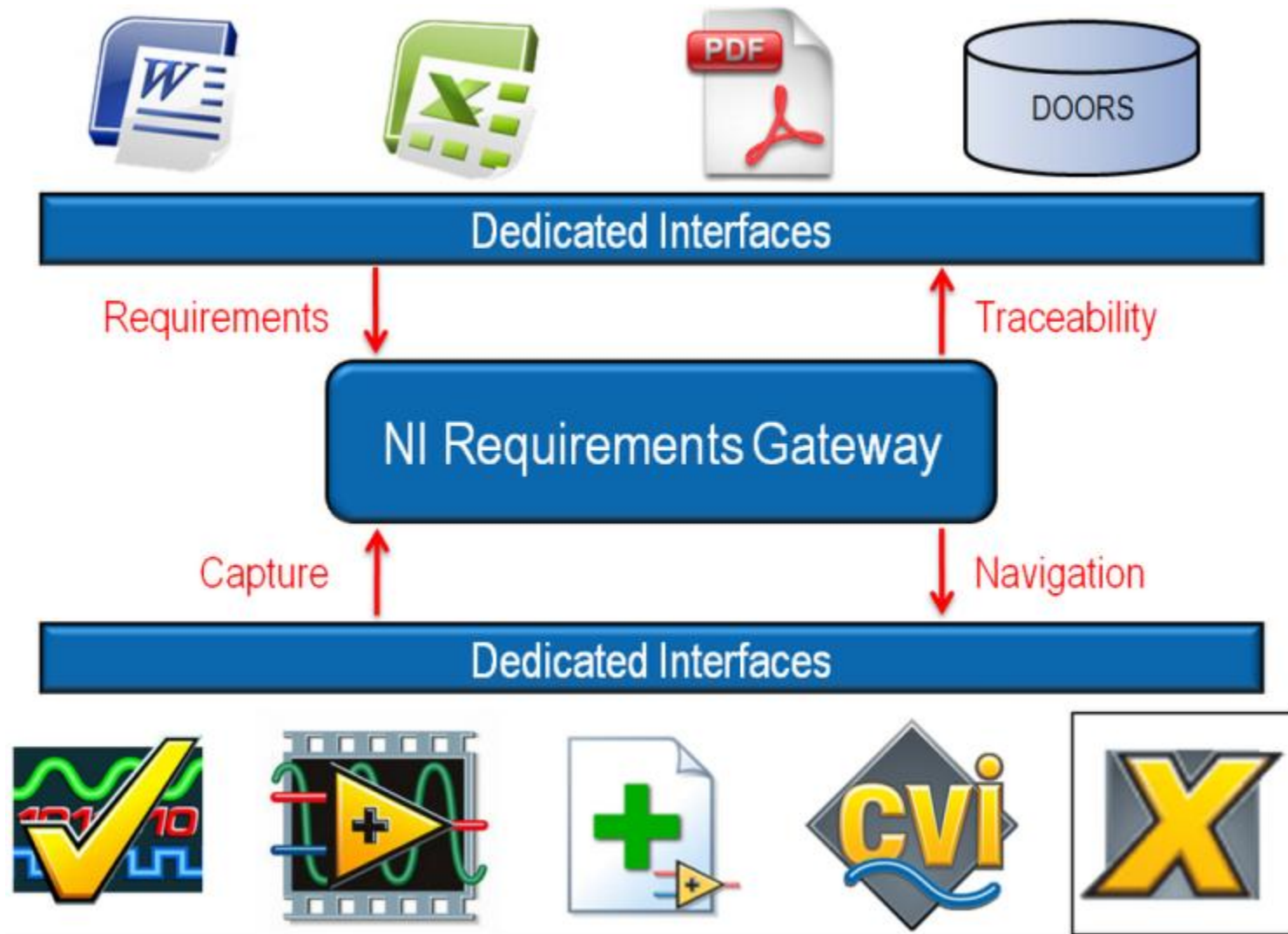
Software Engineering Tools

Software Engineering Tools for LabVIEW



These tools improve and automate life-cycle management, code quality analysis and developer productivity in order to deliver high-quality systems in less time.

Requirements Traceability Solution from NI



Demonstration

Requirements Management with NI Requirements Gateway

Establish or Adopt Development Guidelines

Front Panel Style

- Fonts and Text Characteristics
- Colors
- Graphics and Custom Controls
- Layout
- Sizing and Positioning
- Labels
- Paths versus Strings
- Enumerated Type Controls versus Ring Controls
- Default Values and Ranges
- Property Nodes
- Key Navigation
- Dialog Boxes

Style Checklist

- VI Checklist
- Front Panel Checklist

- Block Diagram Checklist

Block Diagram Style

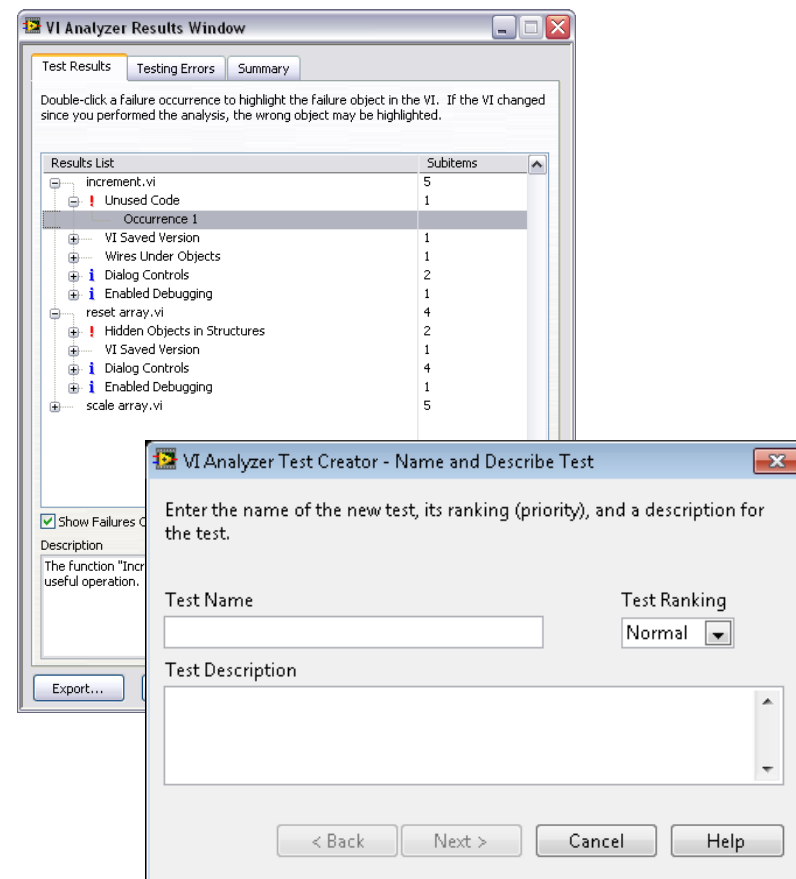
- Wiring Techniques
- Memory and Speed Optimization
- Sizing and Positioning
- Left-to-Right Layouts
- Block Diagram Comments
- Call Library Function Nodes and Code Interface Nodes
- Type Definitions
- Sequence Structures

Icon and Connector Pane Style

- Icons
- Example of Intuitive Icons
- Connector Panes

Preparing for a Code Review with VI Analyzer

- Automate code analysis with 80+ configurable tests
 - Performance
 - Style
 - Complexity
- Interactively inspect failures
- Generate custom reports
- Code complexity metrics
- Write your own tests with VI Scripting *LabVIEW 2010*



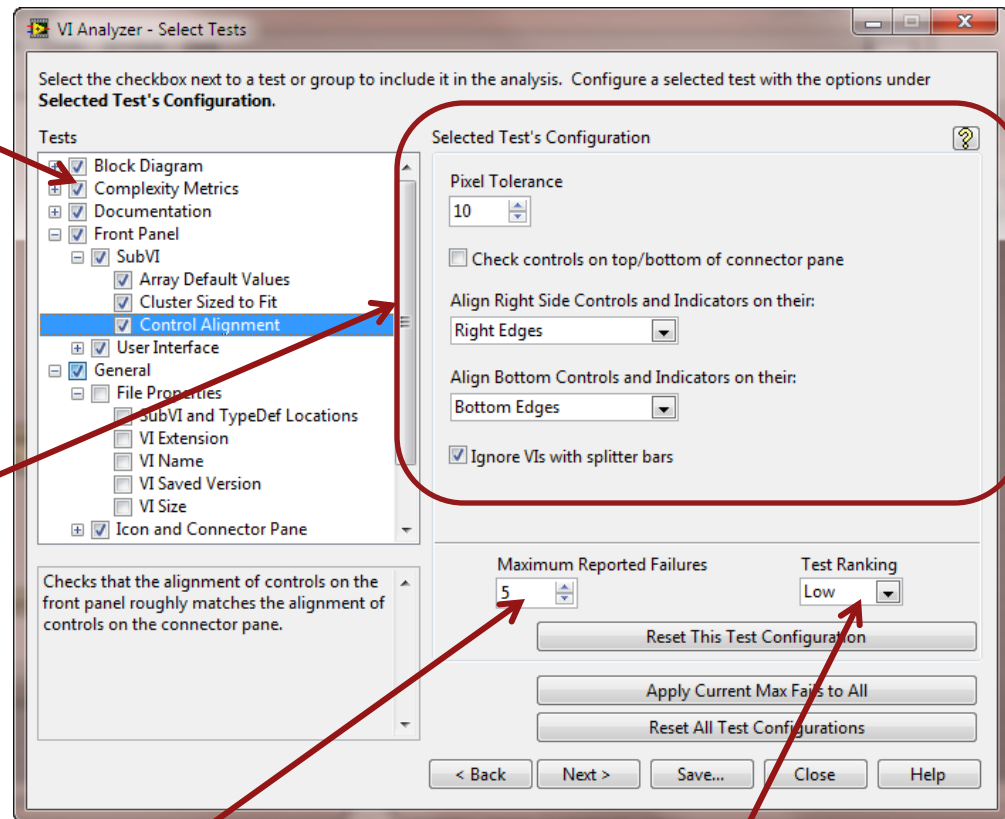
Demonstration

Static Analysis with VI Analyzer

Creating Custom Tests

Select tests to run

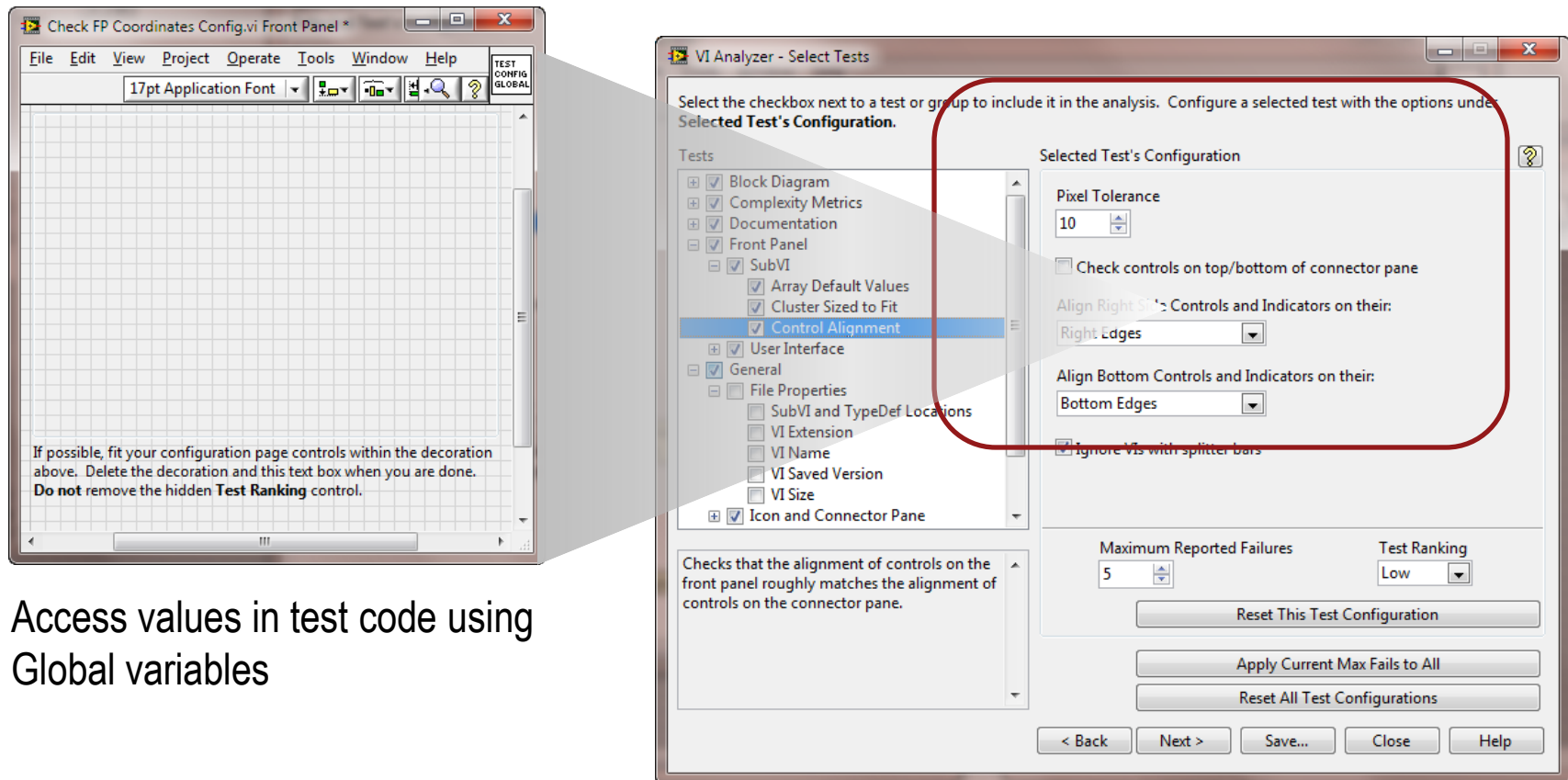
Test specific
configuration
settings



Number of failures to
report

Test Priority

Define Configuration Options



LabVIEW 2013 Desktop Execution Trace Toolkit

New Desktop Execution Trace Toolkit

- Reinvented user interface based on user feedback
- Capable of handling much larger traces
- Improved filtering and sorting options
- Comparison tool for diff'ing trace data

The screenshot displays the Desktop Execution Trace Toolkit interface. The top menu bar includes File, Home, View, and Data. The ribbon contains tabs for Navigate, Filter, Display, Search, Split Pane, and Compare. The main table lists execution events with columns for #, Time, VI, Event, Thread ID, CPU ID, and Highlight. The details pane at the bottom shows an error message: "Error: 1 (LabVIEW: An input parameter is invalid. For example if the input is a path, the path might contain a character not supported by the target device.)".

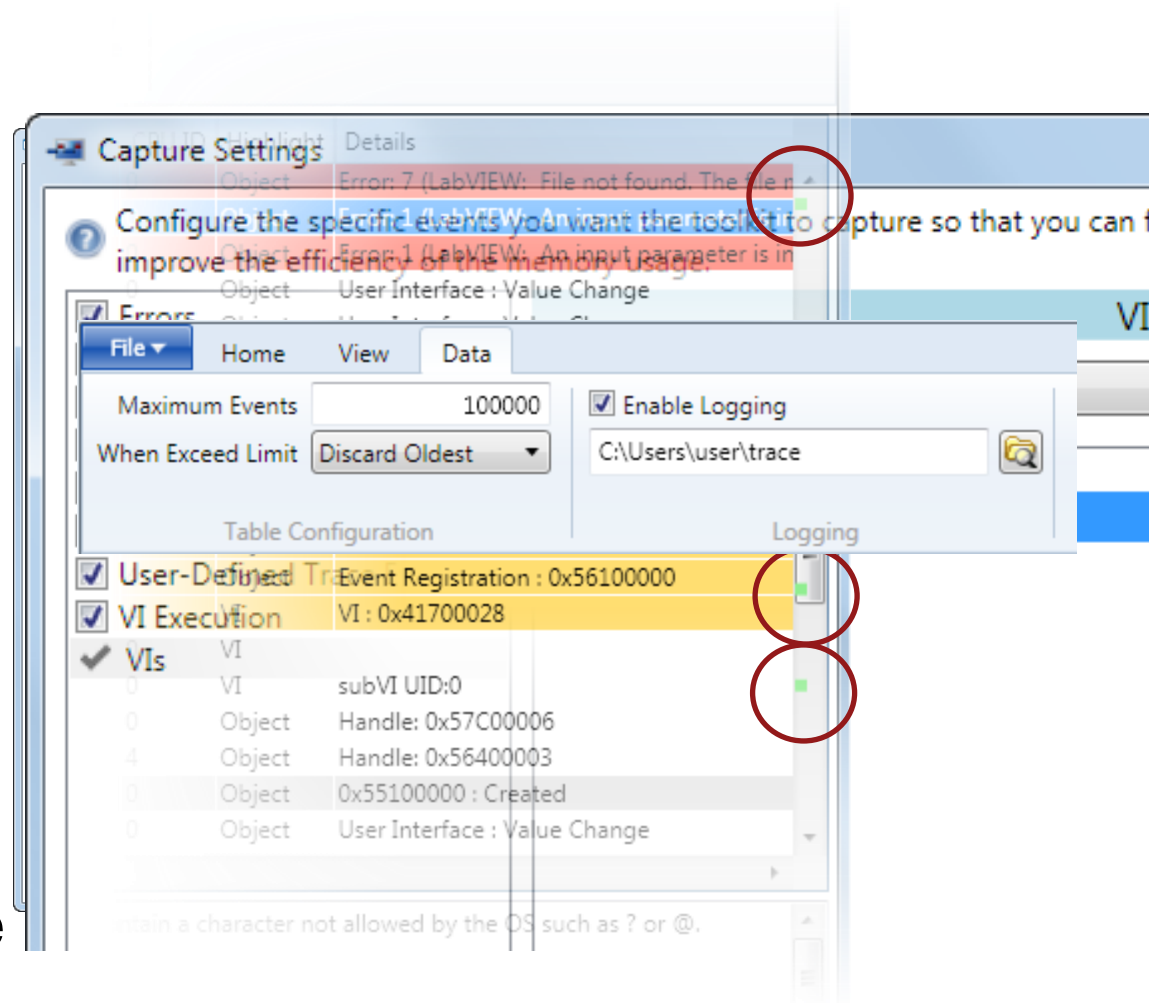
#	Time	VI	Event	Thread ID	CPU ID	Highlight
43	11:57:46.0155391	Desktop Execution - Generate Trace Events.vi	User Defined	5	4	Object
44	11:57:46.0155444	Desktop Execution - Generate Trace Events.vi	End Event Structure	5	4	Object
45	11:57:47.3737473	Desktop Execution - Generate Trace Events.vi	Trigger Event	0	5	Object
46	11:57:47.3737874	Desktop Execution - Generate Trace Events.vi	Begin Event Structure	5	4	Object
53	11:57:48.2033815	Desktop Execution - Generate Test Data.vi:500001	VI Call	5	5	Object
54	11:57:48.2037859	Desktop Execution - Generate Test Data.vi:500001	User Defined	5	5	Object
55	11:57:48.2037911	Desktop Execution - Generate Test Data.vi:500001	VI Return	5	5	Object
59	11:57:48.2456207	Desktop Execution - Generate Trace Events.vi	End Event Structure	5	5	Object
60	11:57:55.1743040	Desktop Execution - Generate Trace Events.vi	Trigger Event	0	5	Object
61	11:57:55.1743478	Desktop Execution - Generate Trace Events.vi	Begin Event Structure	6	1	Object
62	11:57:55.1763830	Desktop Execution - Generate Trace Events.vi	Error	6	0	Object
63	11:57:55.1764073	Desktop Execution - Generate Trace Events.vi	Error	6	0	Object
64	11:57:55.1764192	Desktop Execution - Generate Trace Events.vi	Error	6	0	Object
65	11:57:55.1764292	Desktop Execution - Generate Trace Events.vi	End Event Structure	6	0	Object
66	11:57:55.8778367	Desktop Execution - Generate Trace Events.vi	Trigger Event	0	1	Object
67	11:57:55.8779077	Desktop Execution - Generate Trace Events.vi	Begin Event Structure	5	5	Object
68	11:57:55.8779468	Desktop Execution - Generate Trace Events.vi	End Event Structure	5	5	Object
69	11:58:51.4485425	Desktop Execution - Generate Trace Events.vi	VI Return	6	0	VI
70	11:58:51.4485559	Desktop Execution - Generate Trace Events.vi	VI Stop Execution	6	0	VI
71	11:58:51.4779710	Desktop Execution - Generate Trace Events.vi	Reference Leak	0	4	Object
72	11:58:51.4779768	Desktop Execution - Generate Trace Events.vi	Reference Leak	0	4	Object

Details: Error: 1 (LabVIEW: An input parameter is invalid. For example if the input is a path, the path might contain a character not supported by the target device.)
NI-488: Command requires GPIB Controller to be Controller-In-Charge.)
Call Chain:
-- Desktop Execution - Generate Trace Events.vi
VI Path: C:\Program Files\National Instruments\LabVIEW 2013\examples\Desktop Execution Trace\Desktop Execution - Generate T

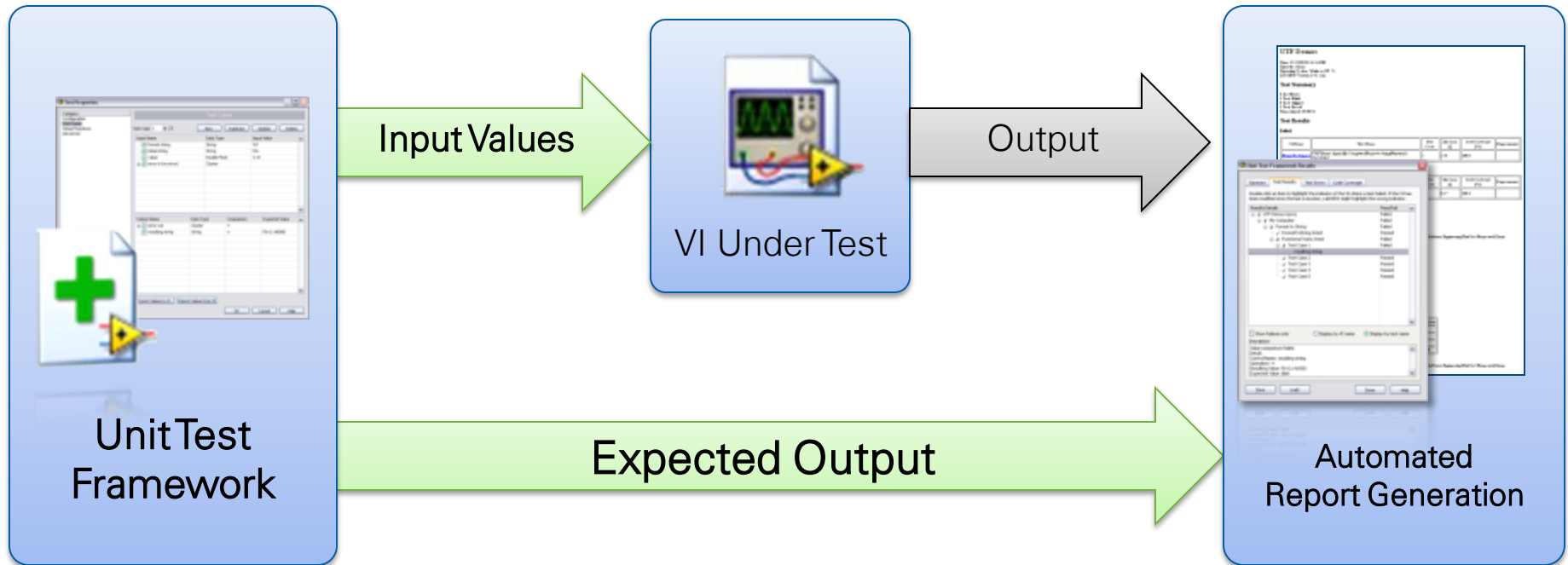
LabVIEW 2013 Desktop Execution Trace Toolkit

Feature Highlights

- Compare different sessions to examine behavioral changes
- User-requested trace configuration options
- Bookmarks make navigating multiple traces simple
- Automatic logging allows extended trace sessions



LabVIEW Unit Test Framework



Test vector = Input value(s) + Expected output(s)

LabVIEW 2013 Unit Test Framework

Test Properties : Calculate Blood Pressure - Range Exceeded.lvtest

Category
Configuration
Test Cases
Setup/Tear Down
Test Vectors
Advanced

Test Case 5

Comment This case tests the same peak pressures with different heart-rates to ensure the algorithm still correctly computes the blood pressure.

Heart Rates
Peak Pressures
Peak Amplitudes
error in (no error)

mean
Systolic
Diastolic
error out
Output

Input Value

Input Value
67.41573
62.433817

Output Name Data Type Comparison

Output Name	Data Type	Comparison
VI under Test		
Systolic	Double Float	=
Diastolic	Double Float	=
Output	Cluster	=
Pressure	Array[Double Float]	=
Amplitude	Array[Double Float]	=
mean	Double Float	=
error out	Cluster	=

62.433817

Cancel Help



LabVIEW Idea Exchange

Demonstration

Regression Testing with the Unit Test Framework

Keep in Touch

Join the Advanced Development Community:

ni.com/community/largeapps

Download this code and slides from

bit.ly/lv_swe