Table and figure legends

Table 1. Problems that multiple support staff (AE Specialists) reported as common with LabVIEW users

Problem (P = performance-related)	Description	# who reported
Too Many Variables P	Too many variables in a VI can lead to race conditions, The suggested limit ranges from 2 to 5 variables per VI—more than this raises the risk of inconsistent use.	8
Build Array in Loop P	When a build-array node is inside of a loop, it builds a new copy of the array every iteration. This can cause slow performance and memory issues.	6
Multiple Array Copies P	When an array is forked and passed to multiple nodes, a new copy of the array is made—and large arrays forking multiple times can cause memory issues	5
No Wait in Loop P	If there is no wait inside of a loop, it could cause synchronization issues and also usually makes front panel elements unresponsive	4
Unconnected Front Panel elements	When elements on the front panel are not connected to anything in the block diagram, this can lead to unexpected behavior or confusing VIs	3
Redundant Operations P	When there is large data, such as a large array, having operations that are redundant can waste time and memory in the VI (e.g., adding 0 to each element of an array)	3
No Queue Constraint P	When there are no constraints on a queue, the queue can grow infinitely large. This can cause performance problems due to memory and loop synchronization issues.	3
Infinite While Loop P	A loop that runs infinitely (sometimes due to the lack of any rule for termination) may cause issues with expected behavior and execution time.	3
Non Reentrant VI's P	Non-reentrant VIs have a data space shared between multiple calls. If a non-reentrant VI is in a structure that can be parallelized, blocking causes poor execution speed.	2
String Concatenation in Loop P	When string concatenation occurs inside of a loop, it builds a new copy every iteration. This can cause slow performance and memory issues.	2
Sequence Structure P	Sequence structures remove a lot of the power of LabVIEW, limiting the compiler's optimization options and potentially reducing readability.	2
Multiple Nested Loops	Multiple nested loops reduce readability.	2
Only Loop Once	If a loop only iterates once, then having the loop structure reduces readability	2

Table 2. Effectiveness and efficiency results

	With tool	Without tool
Success rate at finding problem	100% (13/13)	38% (5/13)
Average time for finding problem (minutes)	4.49 (13 cases)	6.85 (5 cases)
Success rate at finding solution (after finding problem)	92% (12/13)	60% (3/5)
Average time for finding solution after finding problem (minutes)	0.83 (12 cases)	2.37 (3 cases)

Table 3. Prevalence of smells detected by SDPA in Real World Examples Corpus

Smell	Total number of instances	Unique VI's with smell
No Wait in Loop	37	19
Terminals in Structure	34	18
Build Array in Loop	34	12
Uninitialized Shift Registers	14	6
Multiple Array Copies	11	5
Sequence Structure	6	4
Non-reentrant VI's	3	2
Too Many Variables	4	3
Infinite While Loop	3	3
No Queue Constraint	2	2
String Concatenation in Loop	2	2
Redundant Operations	0	0
Total:	150	30

Table 4. True and false positives for SDPA without and with profiling enabled

Technique	Total smell instances found	True positives	False positives
SDPA alone	150	102	48
SDPA with profiling	112	88	24

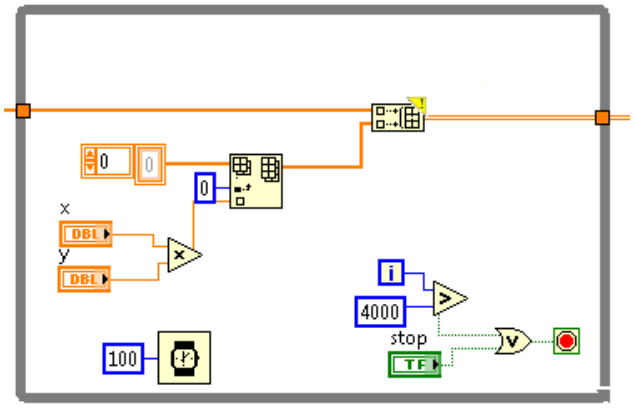

Figure 1. Example of a "Build Array in Loop" smell. Our prototype adds a small yellow triangle to the "Build Array" node (top center node).

Time	VI	Event	Threac	CPU I
19:14:47.806976	MasterSlavePattern 1.vi	VI Start Execution	7	0
19:14:47.806984	MasterSlavePattern 1.vi	VI Call	7	0
19:14:47.807009	MasterSlavePattern 1.vi	Obtain Notifier	7	0
19:14:47.807028	MasterSlavePattern 1.vi	Wait on Notification	7	0
19:15:08.634203	MasterSlavePattern 1.vi	Notifier Wake	7	1
19:15:08.634232	MasterSlavePattern 1.vi	VI Return	7	1
19:15:08.634235	MasterSlavePattern 1.vi	VI Stop Execution	7	1

Figure 2. Output from LabVIEW Desktop Execution Trace Toolkit

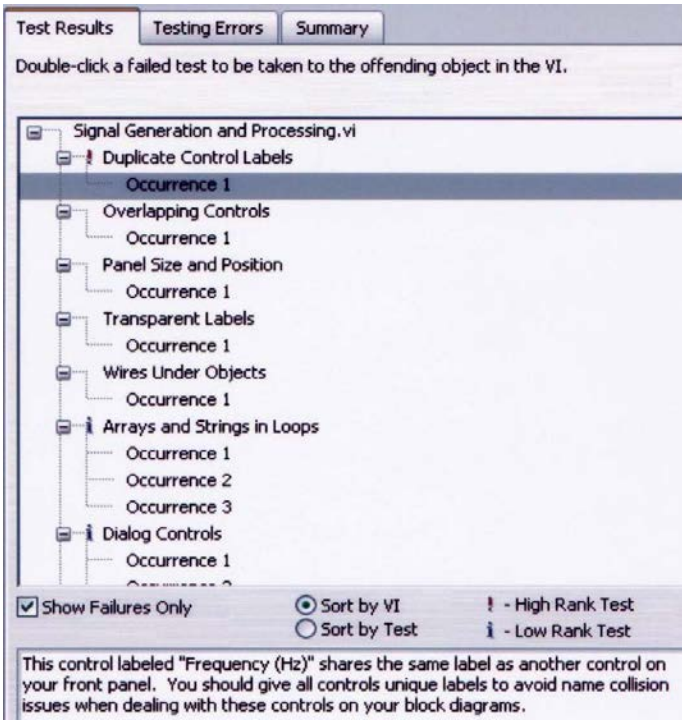

Figure 3. Output from LabVIEW VI Analyzer