

Delivering Higher Quality Products Faster on AdvancedTCA Systems

Bulent Kasman

Chief Architect

Wind River

Device Test Product Division

The Need

- Diagnosing software faults during system integration and product validation and provide fault resolutions in a cost effective manner.
 - It is very expensive to troubleshoot deployed devices and deliver patches.
- Why is this a problem?
 - Current root cause analysis techniques and tools are inefficient in system integration, product validation, and field trial phases.
 - Which contributes to the high integration and test costs, schedule delays, poor quality.

How is this problem addressed today?

- Remove all defects during unit testing.
 - Do more unit testing.
 - Adopt newest development processes like TDD or Agile.
 - Which means adding more testing.
 - Design and code inspections.
 - Static code analysis tools.
- Design for supportability
 - Specify design and coding guidelines for adding log statements in code.
 - Impossible to measure and enforce.
 - Can't be turned on due to big performance impact.
 - Usually limited to error logging. Little value without the context information.
 - Use core dumps to log catastrophic failures.
 - Can't always be used in chassis based devices.

Added Complexity...

- Complex topology and software environment
 - Very difficult to test components in isolation.
 - High dependency on timing makes it impossible to turn-on “logging” without significantly altering the system behavior.
- Open source and third party components.
- Different teams in the company follow different development guides.
- Source code developed over a long period
 - Logs are not used consistently.
- Problems that are not reproducible on developer systems are not resolved.
 - Leads to lower quality of the shipping product.
- Product is already late to the market!!!

What seems to be needed is...

- Capability to add/change code of a running device “live.”
 - Change the code during test, to collect data or to force a change
- Critical requirements for dynamic instrumentation
 - Must be able to do this while the application is running.
 - Must have minimal overhead.
 - Must be precise
 - Must be safe.
 - Must be able to do this remotely.
 - Must be easy to use.

How to do it?

- There are some technologies with varying levels of capabilities
 - Open source technologies, such as Dynamic Probes, DTrace, System Tap, etc.
 - There are a few commercial products.
 - Wind River offers Sensorpoint and Patchpoint technologies, which is what I will talk about here.
 - These technologies are leveraged by Wind River Test Management and Wind River Test Diagnostics products.

Sensorpoint & Patchpoint Technology

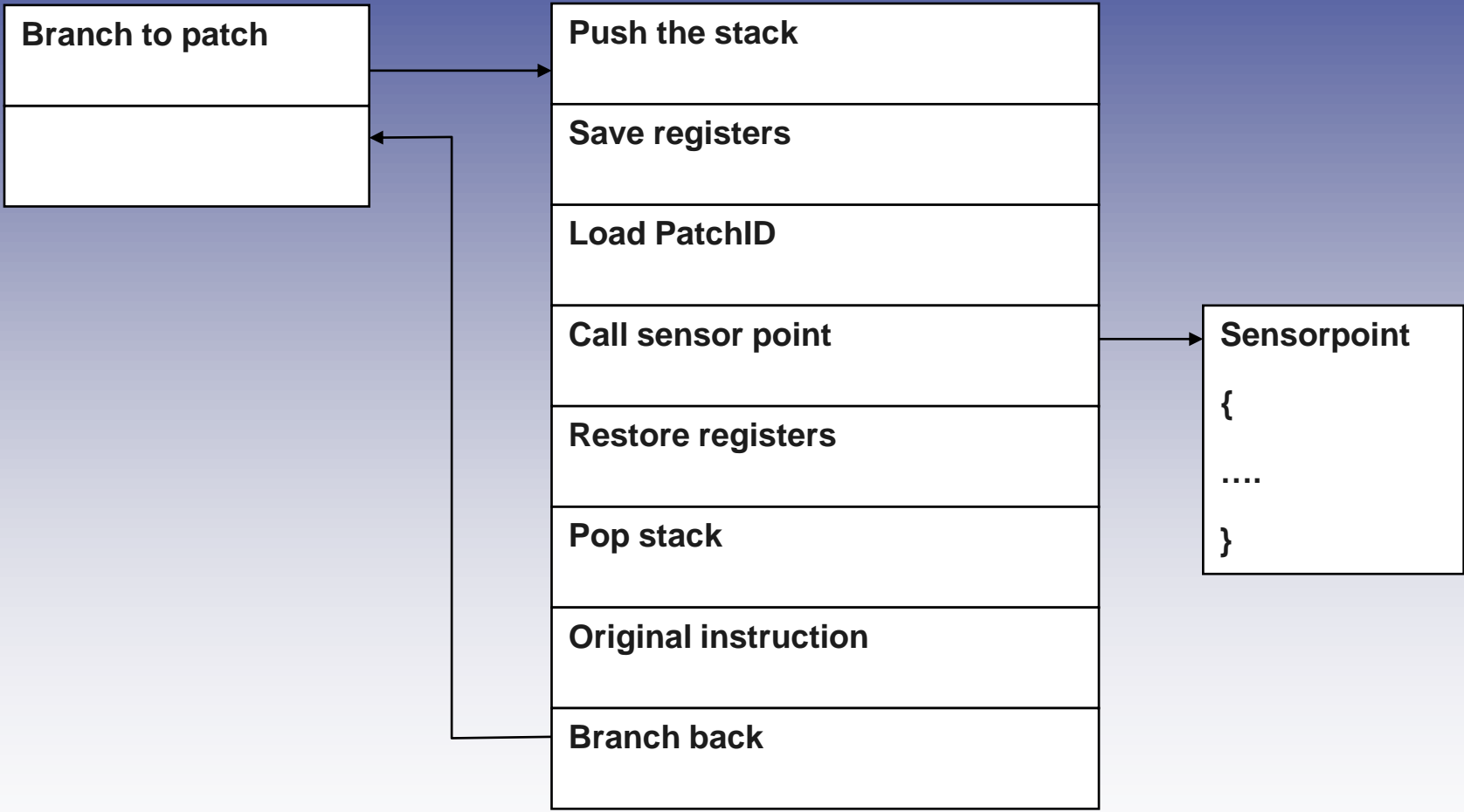
- Three fundamental capabilities:
 - *Replace* an existing function with a new one (patchpoint).
 - *Add* new code to an existing function
 - Entry, exit, or anywhere in between.
 - *Inject* new code to a process/kernel and run it.
- All of these capabilities are managed dynamically
 - Dynamically enable/disable the new code remotely from a client application.
 - No need to pre-instrument or “prime” the application.
 - On some platforms, application is pre-linked with a small-library however.
 - An agent in the device enables remote management and deployment of sensorpoints and patchpoints.

How does it work?

- Sensorpoints and patchpoints are compiled native binaries.
 - This makes it less safe, compared to an “interpreted” model, but also much less intrusive in terms of performance and memory.
- Sensorpoints are fragments of code that are to become a part of an existing function, at the specified location.
 - They are written in C, with some proprietary directives to specify code location.
- Sensorpoints can also be just loaded into the application process space and run.
- Patchpoint is a full replacement function for an existing one.
 - It is written in the native language of the application.
 - It has a simpler/less overhead execution path (next slide) compared to Sensorpoint.

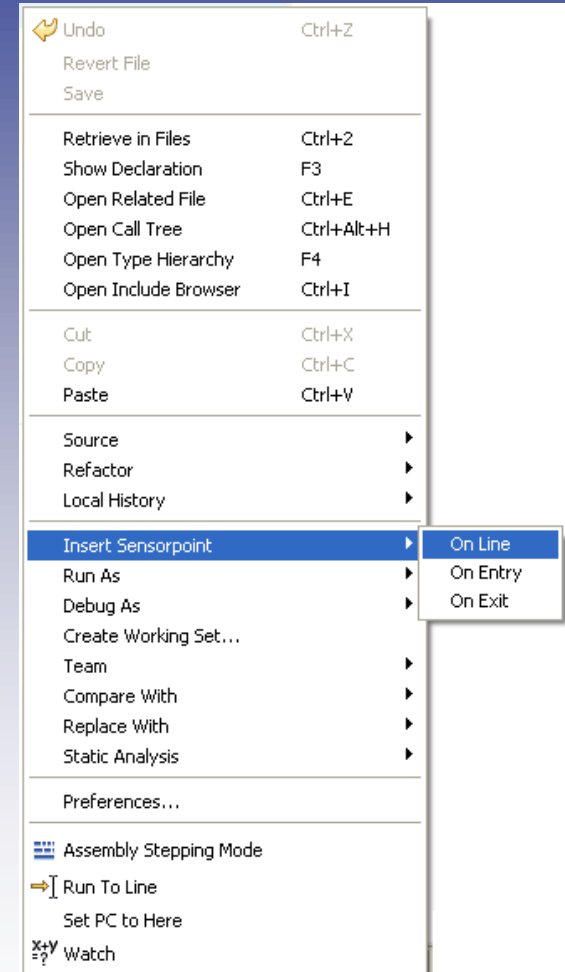
Sensorpoint Execution Path

Foo()



Sensorpoint and Patchpoint Development

- Eclipse based IDE to automatically generate Sensorpoint and patchpoint from source code.
- Features:
 - Pull-down menu with sensor point actions
 - **Line select with right-click instrumentation insertion**
 - Auto-template generation for on-entry, on-line, and on-exit instrumentation types
 - Patchpoint generation
 - **Copies the original function into patchpoint source for modifications**



How do we use this technology?

- Debugging applications by easily injecting `log()`, `printf()` into a running application for debug purposes.
 - Used by developers to shorten the debug time, especially for complex environments.
 - Can use sensorpoints and patchpoints.
- Monitor system values, execute test code during test.
 - Monitor semaphore states, CPU, memory utilization, global variables, linked lists.
- Change application behavior.
 - Change global variables, function arguments, simulate events.
- Try out fixes by using patchpoints.

How do we use it in testing?

- **Fault injection**
 - Inject faults by altering function arguments, alter return values, skip execution of functions.
 - Using sensorpoint technology, function return values can be altered based on the calling function.
- **Test coverage**
 - Dynamically add instrumentation to thousands of functions during test and measure coverage.
- **Measure function performance**
 - Measure execution time of critical functions.
- **Because instrumentation is dynamic, special, instrumented builds are not required.**

Wind River is the global leader in
Device Software Optimization (DSO).

Wind River enables companies to develop,
run,
and manage device software better, faster,
at lower cost, and more reliably.

Thank you.