Static Analysis Considerations for Medical Device Firmware

Gerald Rigdon
Fellow, Software Engineering
Boston Scientific
gerald.rigdon@bsci.com

Boston, 9-23-2010

ESC-423

Key Points From Abstract

- *Static Analysis essential*
- *COTS adequate for some*
- *Static analysis framework from design*
- *Customized solutions very effective*
- *Learning from customized approaches*

Definitions

Dynamic – executing

Static – not executing

"Static analysis evaluates the software without executing it. Instead, it examines a representation of the software. <u>In some ways, static analysis is more complete than dynamic analysis</u>, since general conclusions can be drawn and not just conclusions limited to the particular test cases that were selected."

-- Nancy Leveson. Safeware. Addison-Wesley, 1995

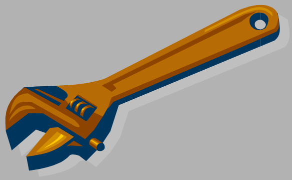
"Static testing plays an important role in establishing the characteristics of the system over its entire operating range – <u>a function that cannot be performed during dynamic methods</u> because of the infinite number of tests that would be required."

-- Neil Storey. Safety-Critical Computer Systems. Addison-Wesley, 1996

Why Static Analysis Tools?

The Problem Space – Candidates for SA Tools

- Well defined set of properties that can be checked by a machine
- High likelihood of missing a defect in dynamic testing and code reviews
- Difficult to perform manually or using simple search techniques
- High Impact (safety risk, efficacy) if a defect escapes and leads to system malfunction



Quantitative Factors

TP = True Positive (defect found was real)

FP = False Positive (defect not real, a false alarm)

FN = False Negative (defect not found)

Recall = TP / (TP + FN)

Precision = TP / (TP + FP)

Static Analysis Tool Challenges

- State space of software is large!
  - Infinite at a practical level

  - Properties are "undecidable"
    - Leads to "approximations"

What is the conclusion?

Answer: You Can't Have it All!

- Over approximate (False Positives)
- Under approximate (False Negatives)

Why Tool Customization?

The Solution Space

Commercial Off the Shelf Software (COTS) vs. Customization

- **Quantitative Factors**
- **Acceptable analysis execution cycles**
- **Enforce Design Constraints**
  - **Select analyses most important to your domain**
    - ***Stay Tuned: Custom Static Analysis Project***

Coding Standards / Rule Sets

- **MISRA** – Motor Industry Software Reliability Association
  - C language rules
  - MISRA-C:2004
    - 121 mandatory rules

- **Should all these rules be used?**

‘it is likely that adherence to the ***MISRA*** *standard as a whole would have made the software less reliable. This observation is consistent with Hatton’s earlier assessment of the* ***MISRA*** *C 2004 standard [10]’.*

-- Cathal Boogerd and Leon Moonen. Assessing the Value of Coding Standards: An Empirical Study. Report TUD-SERG-2008-017.

Coding Standards / Rule Sets

- JPL - Jet Propulsion Laboratories (Power of 10)

Rule1: Restrict all code to very simple control flow constructs
Rule2: All loops must have a fixed upper-bound
Rule3: Do not use dynamic memory allocation after initialization
Rule4: No function should be longer than what can be printed on a single sheet of paper
Rule5: The assertion density of the code should average to a minimum of two assertions
Rule6: Data objects must be declared at the smallest possible level of scope

Rule7: The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.

Rule8: The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions.

Rule9: The use of pointers should be restricted

Rule10: All code must be checked daily with at least one, but preferably more than one, state-of-the-art static source code analyzer and should pass the analyses with zero warnings.

-- ***Gerard J. Holzmann. The Power of Ten – Rules for Developing Safety Critical Code. NASA/JPL Laboratory for Reliable Software.***

Practical Look at JPL Rule 10

Rule 10: Zero Warnings

- This model works well if rule-set is used on “new” code base
- Applying rules on older “mature” code base can be problematic
  - Number of warnings not always indicative of quality
  - Pacifying a tool’s warnings may introduce problems

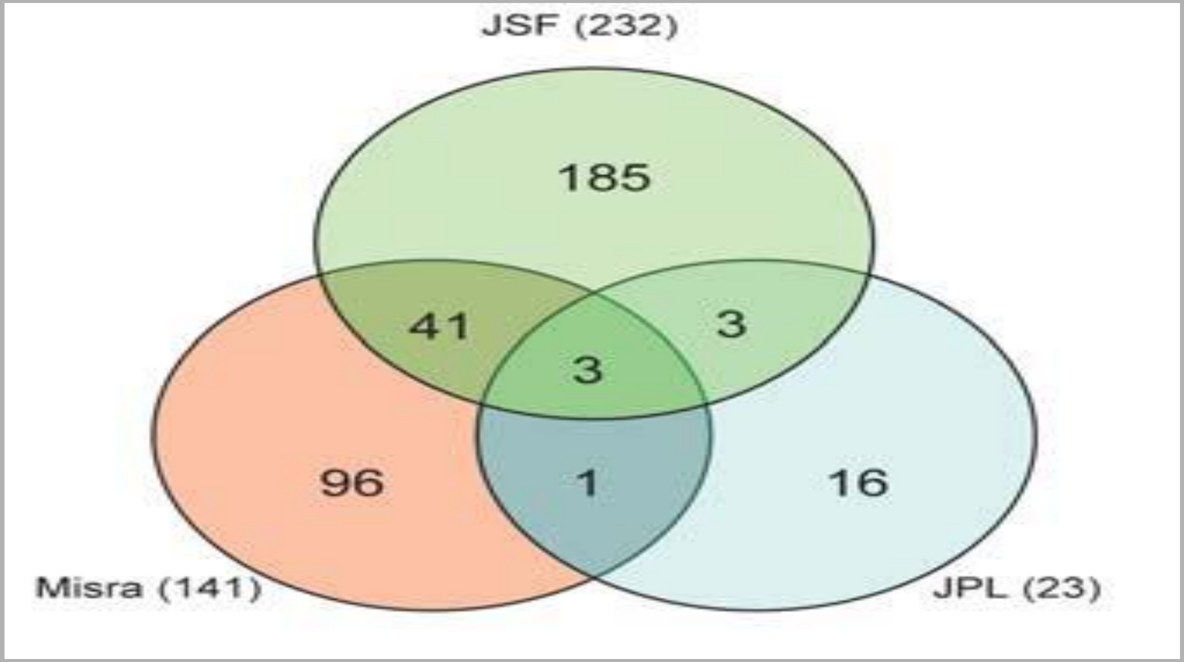
‘it is likely that adherence to the MISRA standard as a whole would have made the software less reliable’

Why?

‘we observed a negative correlation between MISRA rule violations and observed faults. In addition, 25 out of 72 rules had a zero true positive rate…’

-- *Cathal Boogerd and Leon Moonen. Assessing the Value of Coding Standards: An Empirical Study. Report TUD-SERG-2008-017.*

Intersections of Coding Standards



-- Paul Anderson. How to develop a coding standard for an embedded project. Grammatech. June, 2008

Assessing the Coding Standards

- MISRA
  - C Language based, syntactic restrictions, focus on reliability and portability
- JPL
  - Broader than MISRA, but still primarily C language driven rules

Practical Use of Rule Sets

- More effective when subsets are used that make sense for your domain

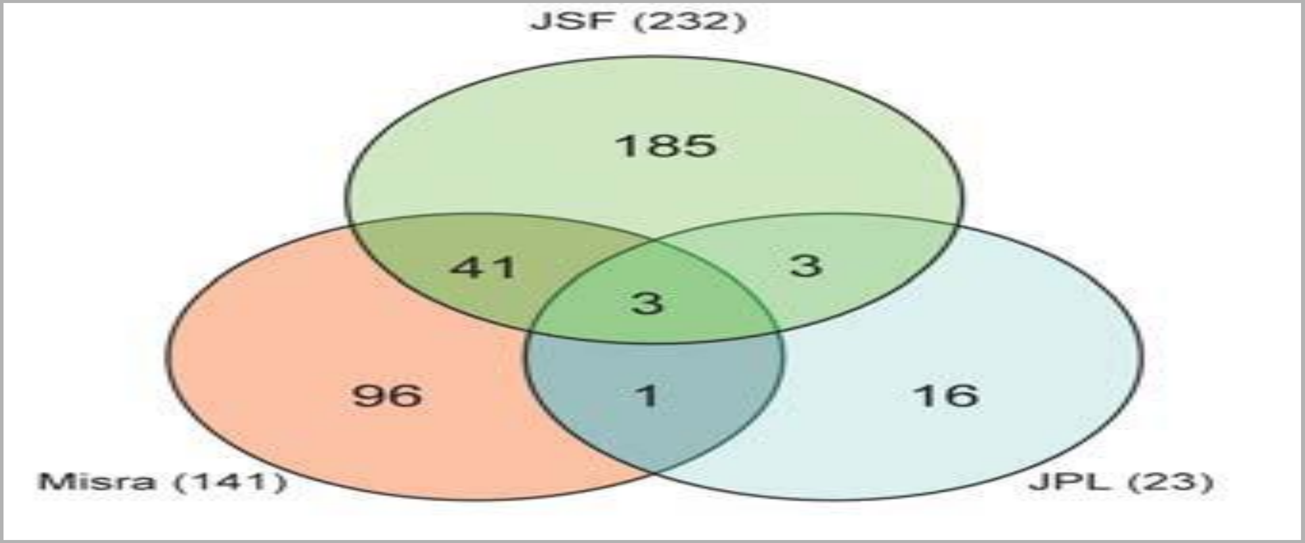
"The subset should be compact and cover most faults in cited surveys…MISRA is welcomed but with caution. When signal to noise ratios are low….it becomes a weapon…only the tool vendors flourish"

-- Les Hatton, A measurement based safer subset of ISO C suitable for embedded system development, Computing Laboratory, University of Kent, 2003

"selection of rules that are most likely to contribute to an increase in reliability maximizes the benefit of adherence while decreasing the necessary effort."

-- Cathal Boogerd and Leon Moonen. Assessing the Value of Coding Standards: An Empirical Study. Report TUD-SERG-2008-017.

Coding Standards & Customization



-- Adapted from Anderson. How to develop a coding standard for an embedded project. Grammatech. June, 2008

Why Focus on Design For Static Analysis?

- Are fielded products issues traceable to MISRA and JPL violations?
  - Probably some

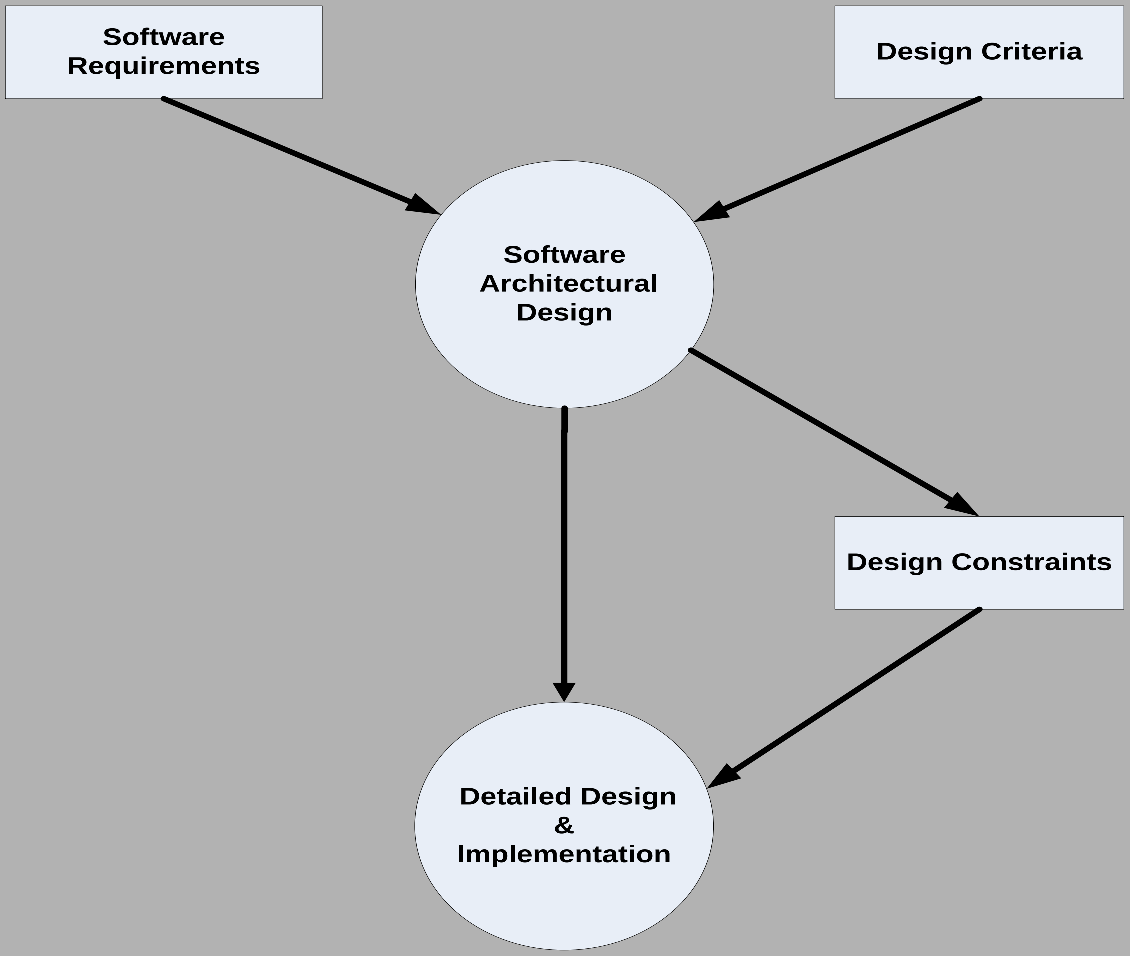
- But, what are the real issues…***really?***
  - Missing / incomplete requirements
  - Requirements not understood during *design*
  - Complexity…interactions between components in a *design*
  - Unconstrained, *brittle design*, maintenance issues

- Static analysis should target the areas of most concern (your domain!)
  - Implementation driven by design constraints - not just language driven

Use of Design Constraints

- Design constraints are mid-level design rules based on design choices made in the software architecture and platform
- Design constraints are an integral part of software risk management:
  - Define the implementation specifics for risk control measures
  - Provide a systematic way to capture the design rules that uphold the assumptions and limitations of the underlying architecture
  - Can dictate how something may NOT be done
  - Provide a means to force an analysis
    - Use of static analysis for compliance, analysis traceable to constraint

Design Constraints Visual Context


Software Requirements
Design Criteria
Software Architectural Design
Design Constraints
Detailed Design & Implementation

Criteria Examples:
- Maintainability
- Robustness
- Reliability
- Testability

Examples of Design Constraints

<u>Design Constraint</u>:
"All data instances that are accessed from multiple priority levels of threads must be protected with adequate guard constructs."

<u>Design Constraint</u>:
"Memory resources must include safety margin, overflow detection, and monitoring."

How are these constraints enforced?

- Stay tuned…………..

CSAP

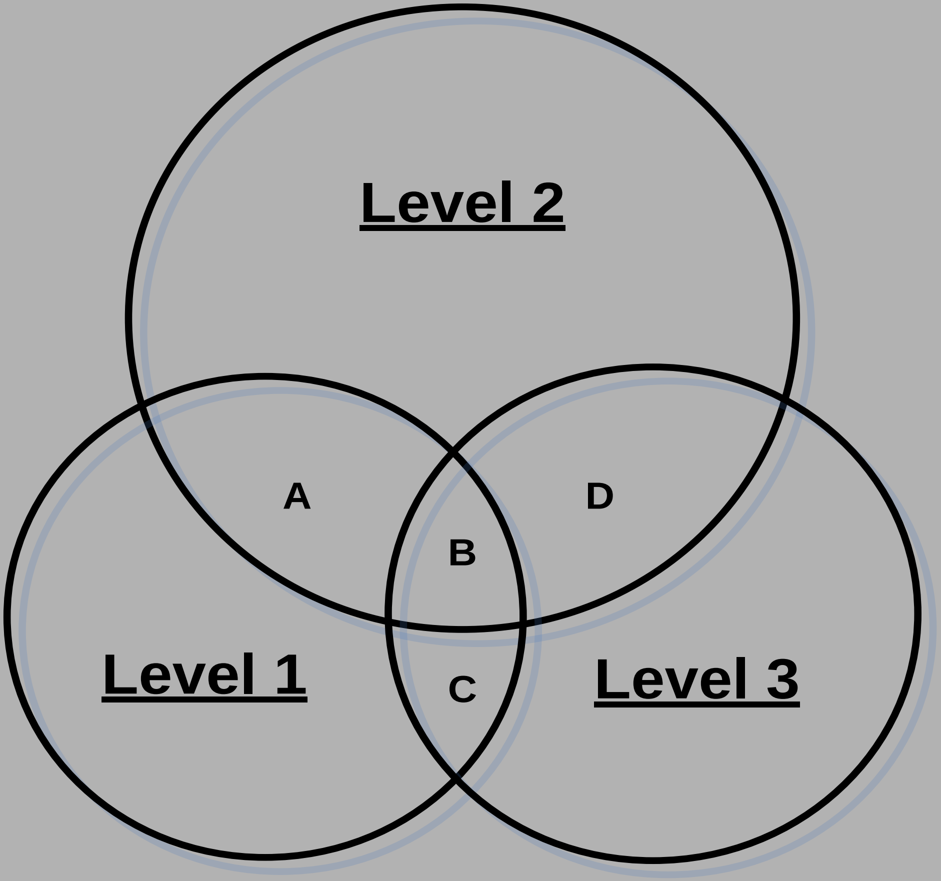
Custom Static Analysis Project

- **Participants:**
  - Boston Scientific (Implantable Firmware)
  - Grammatech (CodeSonar Static Analyzer Tool Vendor)
- **Objective:**
  - Customize a commercial static analysis tool to automate high-risk analyses that are manually intensive
- **Scope:**
  - 5 analyses, 2 of which shall be discussed here:

    1. Shared Data Access and Race Conditions
    2. Stack Usage

Shared Data Analysis

- Design Constraint: "All data instances that are accessed from multiple priority levels of threads must be protected with adequate guard constructs."
- Rationale: Race conditions result in making assumptions contingent on the sequence or timing of other processes, which leads to unpredictable behavior.



A, B, C, D represent "Shared Data" between different priority levels

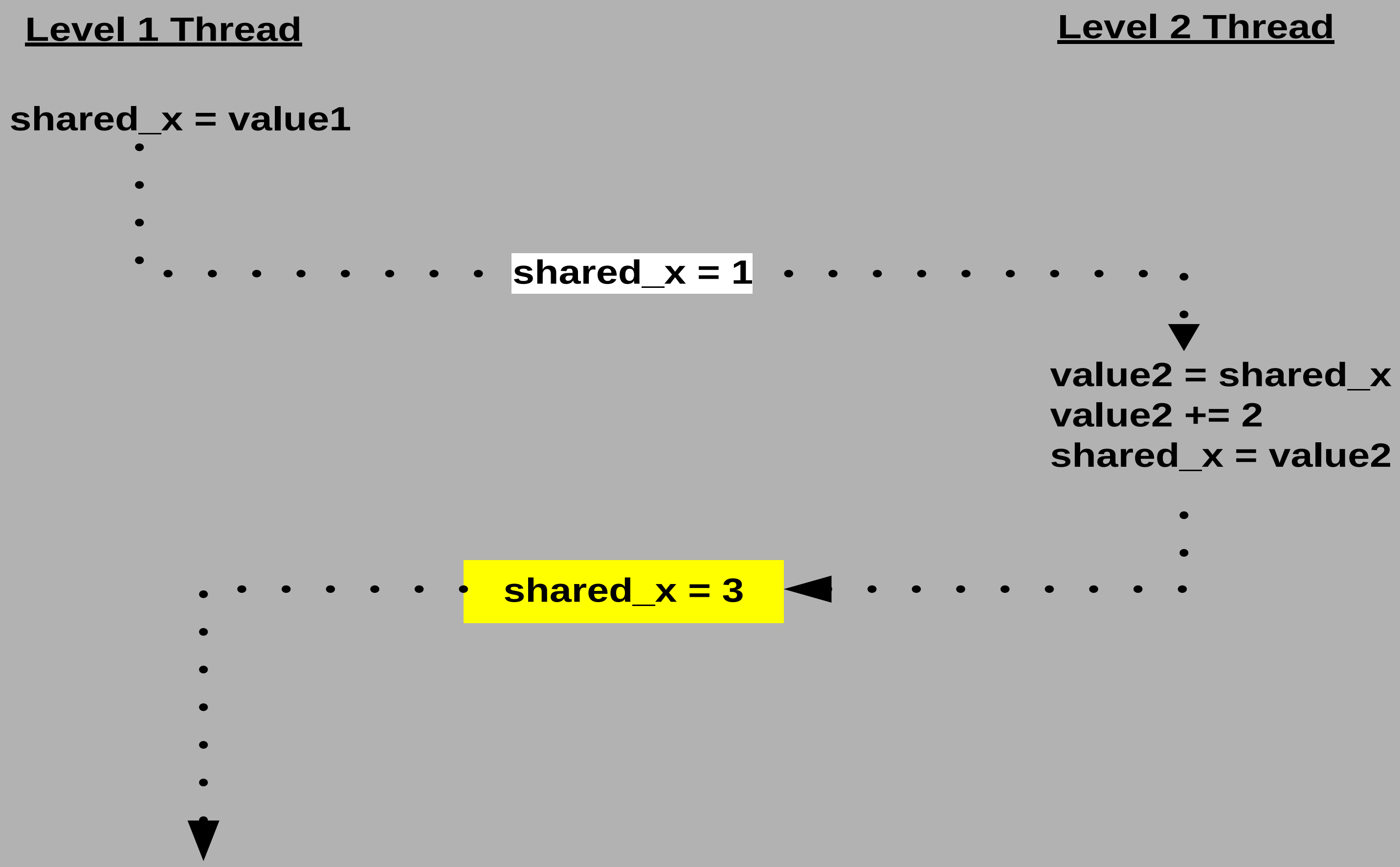
Shared Data Analysis

- A good example where static analysis is better than dynamic testing
  - Static analysis is more complete than dynamic analysis, since:
    - *'general conclusions can be drawn and not just conclusions limited to the particular dynamic test cases that are selected '*

      -- *Nancy Leveson. Safeware. Addison-Wesley, 1995*
  - Characterizing one's software system by static analysis is:
    - *'a function that cannot be performed during dynamic methods because of the infinite number of tests that would be required'.*

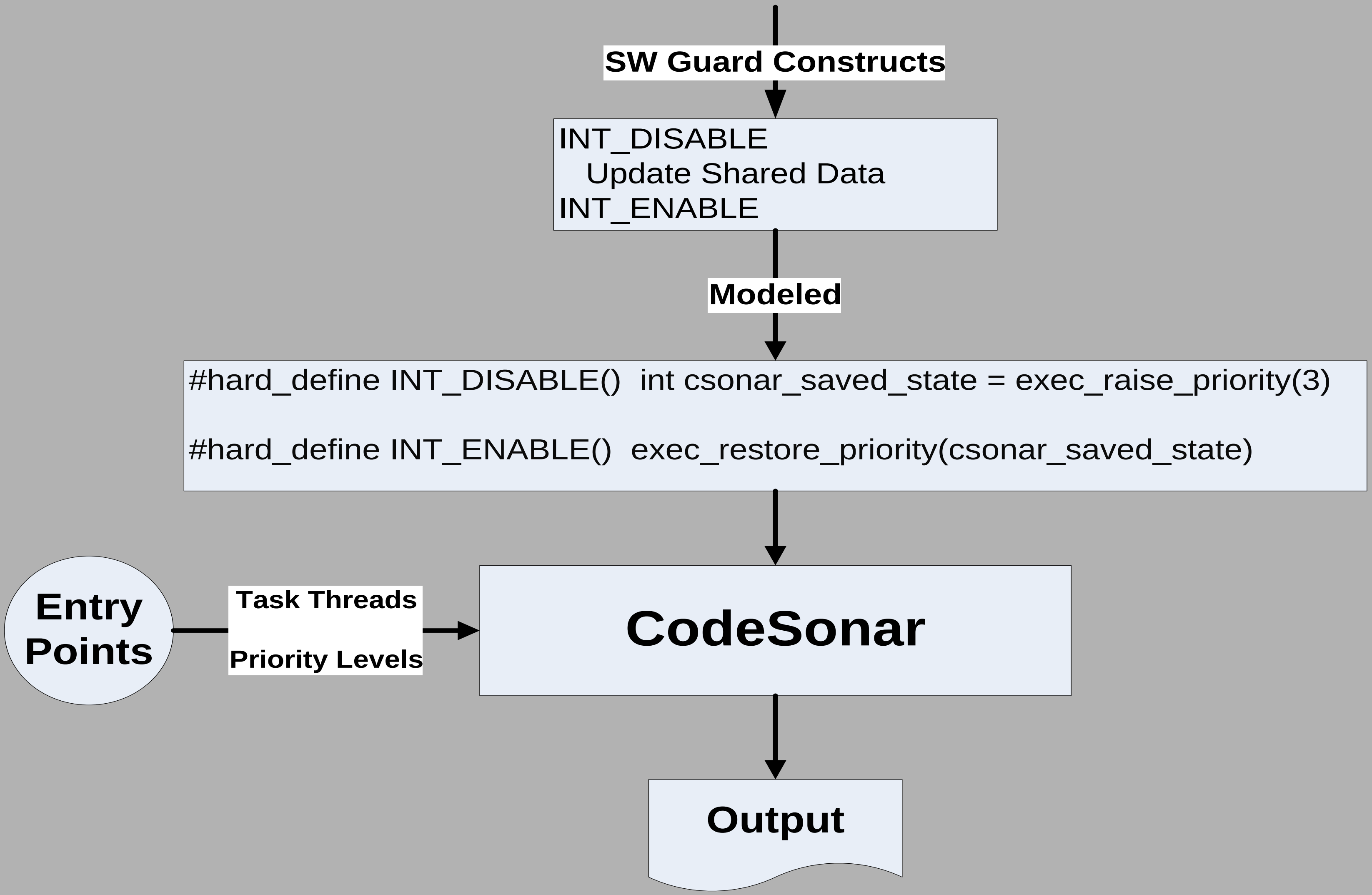
      -- *Neil Storey. Safety-Critical Computer Systems. Addison-Wesley, 1996*

Shared Data Example – Static Analysis Is Best Choice



What are assumptions about use of “shared_x” in Level 1 Thread?

Shared Data Analysis Technical approach


SW Guard Constructs
INT_DISABLE
Update Shared Data
INT_ENABLE
Modeled
#hard_define INT_DISABLE() int csonar_saved_state = exec_raise_priority(3)
#hard_define INT_ENABLE() exec_restore_priority(csonar_saved_state)
Entry Points
Task Threads
Priority Levels
CodeSonar
Output

Stack Usage Analysis

Design Constraint:
"Scalable resources must include safety margin, overflow detection, and monitoring."

- For embedded developers, a trial and error process

"With experience, one learns the standard, scientific way to compute the proper size for a stack: Pick a size at random and hope."
-- Jack Ganssle, "The Art of Designing Embedded Systems," Elsevier, 1999.

- State of the art / standard practices for detecting **stack overflow**
  - Watermark / Empirical

Watermark Example: Constraint…… “Margin, Overflow….”

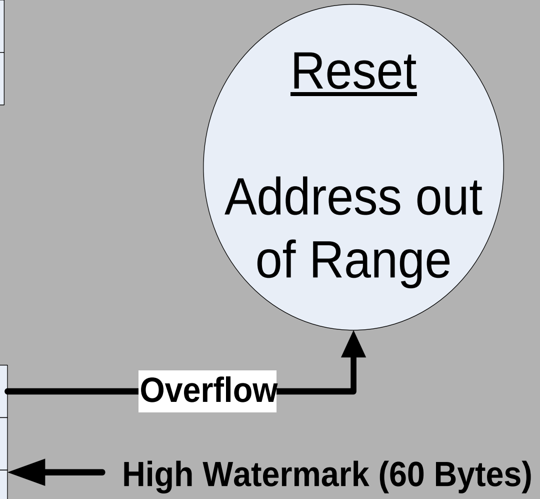
- Uses “Dynamic Testing” Input

Stack Initialization Pattern (100 bytes)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Stack Dynamic Testing Profile

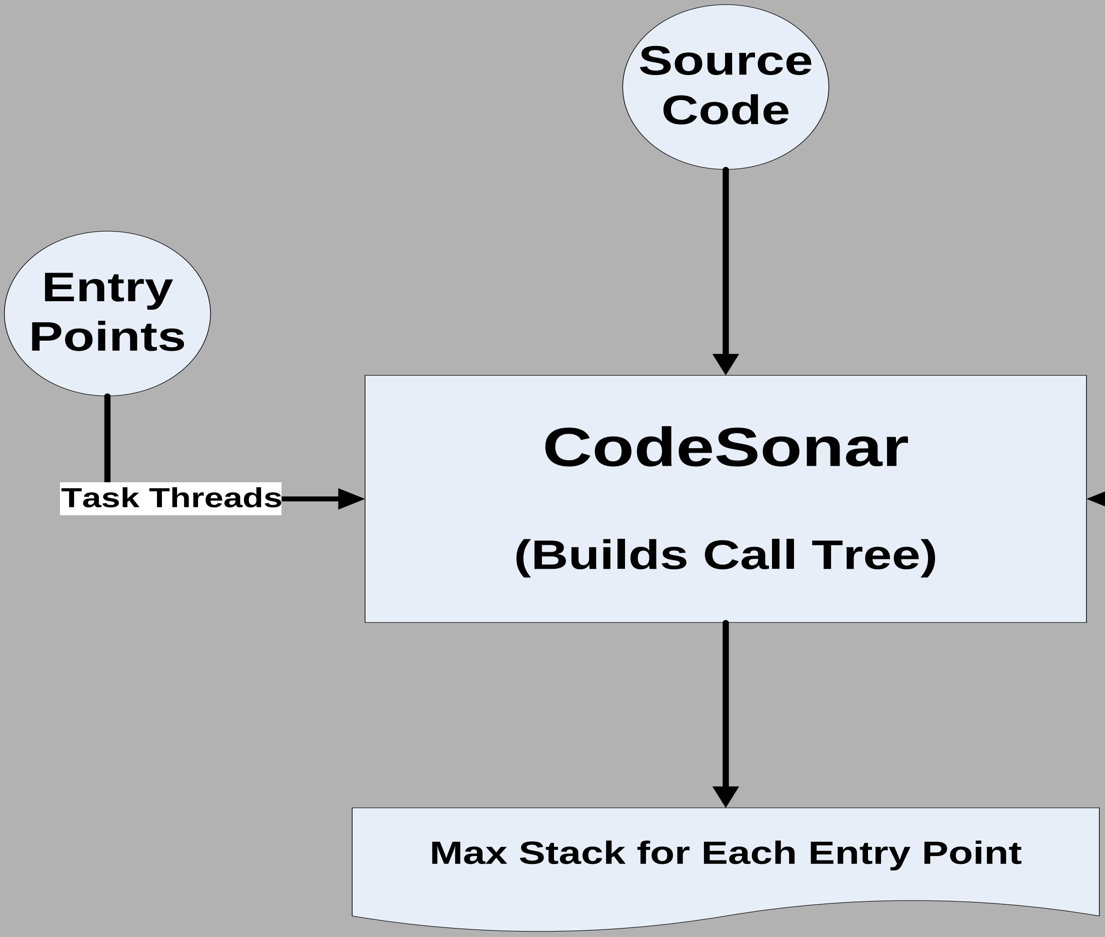
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	4	7	8	9	6	7	0	1	0	4	6	7	9	1	2	2	9
6	0	4	3	6	2	0	1	4	1	0	3	2	9	9	8	1	6	3	2
1	9	3	6	0	8	9	0	2	4	4	2	1	8	6	3	7	9	0	1



Stack Usage Analysis

- Industry tools are sparse
- A new product claims:
  “ESC-Silicon Valley, San Jose, CA, March 31, 2009 - Express Logic, Inc., the worldwide leader in royalty-free real-time operating systems (RTOS) today, introduced a new development tool that helps developers avoid stack overflow problems that traditionally have plagued embedded systems. The new tool, StackX™, performs a comprehensive analysis…“
- This tool does not support all Compiler/Linker output files

Stack Usage Analysis



Stack Usage Analysis

Post Processing CodeSonar Output

- Max Stack estimated based on our pre-emption model
  - Max(Level 1) + Max(Level 2) +….. Max(Level N) = Worst Case Usage
- Allows us to bound the problem
  - watermark stack usage <= actual stack usage <= worst case stack usage

Static vs. Dynamic Review

- Shared Data Analysis
  - Best choice was static analysis

- Stack Usage Analysis
  - Use of static and dynamic techniques

Making the Business Case

Investment considerations

Questions to Ponder:

- How much does your company presently spend on dynamic testing versus static testing?
- If your company invests in static analysis tooling, is there an inclination to purchase these tools off-the-shelf instead of investing in the development of custom tools driven by design and domain specific needs?

Making the Business Case

Investment Assumptions

Test cases are custom – Requirements driven

Is there perhaps a widely accepted false assumption that dynamic testing necessitates custom development while static testing does not?

Building Bridges

- Not all static analysis rule checkers are language driven

- Other classic checks for SA tools
  - Divide by zero
  - Buffer overruns
  - Uninitialized variables

- Opportunities to expand COTS
  - Embracing customized techniques from industry
    - Example: Custom static analysis project we discussed today

Bridge Breaker?

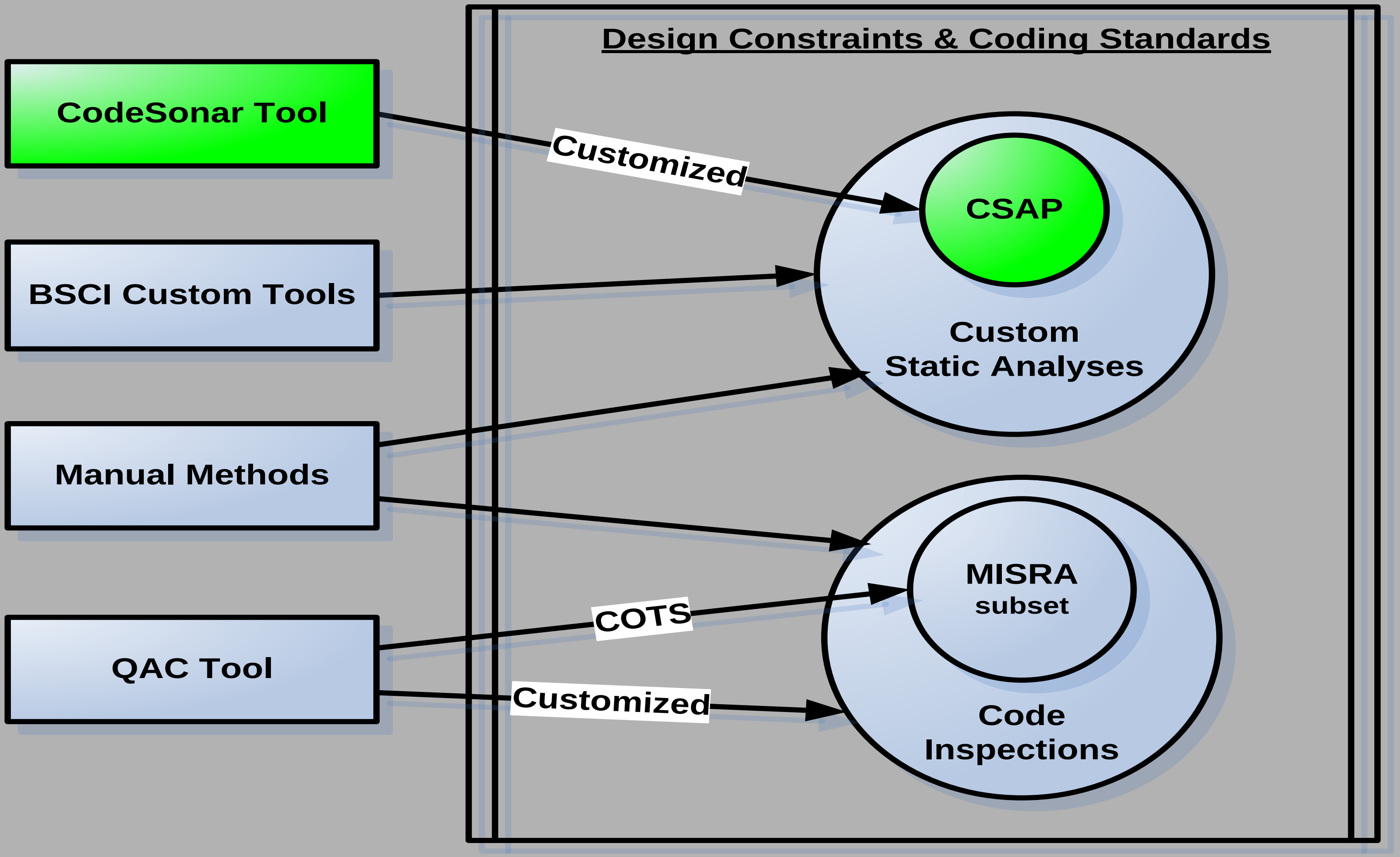
- Are static analysis companies willing to sell highly customizable tools?
  - Does this strategy break their business model by drying up their enhancement pipeline?

Summary

Static Analysis for Medical Devices

- Static Analysis and Dynamic Testing are complementary
  - Static analysis more suitable for certain problems
- COTS static analysis tools are often language driven and implementation focused
  - Effective analysis driven by design requires customization…at present
  - Build the bridges to the future…customization techniques into configurable analysis features built into COTS

Our Static Analysis Framework



Discussion Slide – Where Are We Headed?

Brian Fitzgerald:

U.S. Food and Drug Administration
Protecting and Promoting *Your* Health

- *"We're hoping that by quietly talking about static analysis tools, by encouraging static tool vendors to contact medical device manufacturers, and by medical device manufacturers staying on top of their technology, that we can introduce this up-to-date vision that we have."*

  -- *Chloe Taft. CDRH Software Forensics Lab: Applying Rocket Science To Device Analysis. Published in Medical Devices Today. October 15, 2007.*

Our position:

- Our model of static analysis focuses on design, and our conclusion is that static analysis tools are most effective when they are customized to complement an existing framework based on domain-specific needs.