Static Analysis
in Medical Device FW/SW Development

Chao Wang, V J Jagannathan, Ken Timmerman, Randy Wells
Medtronic, Cardiac Rhythm Disease Management Division

to ISSRE ‘09

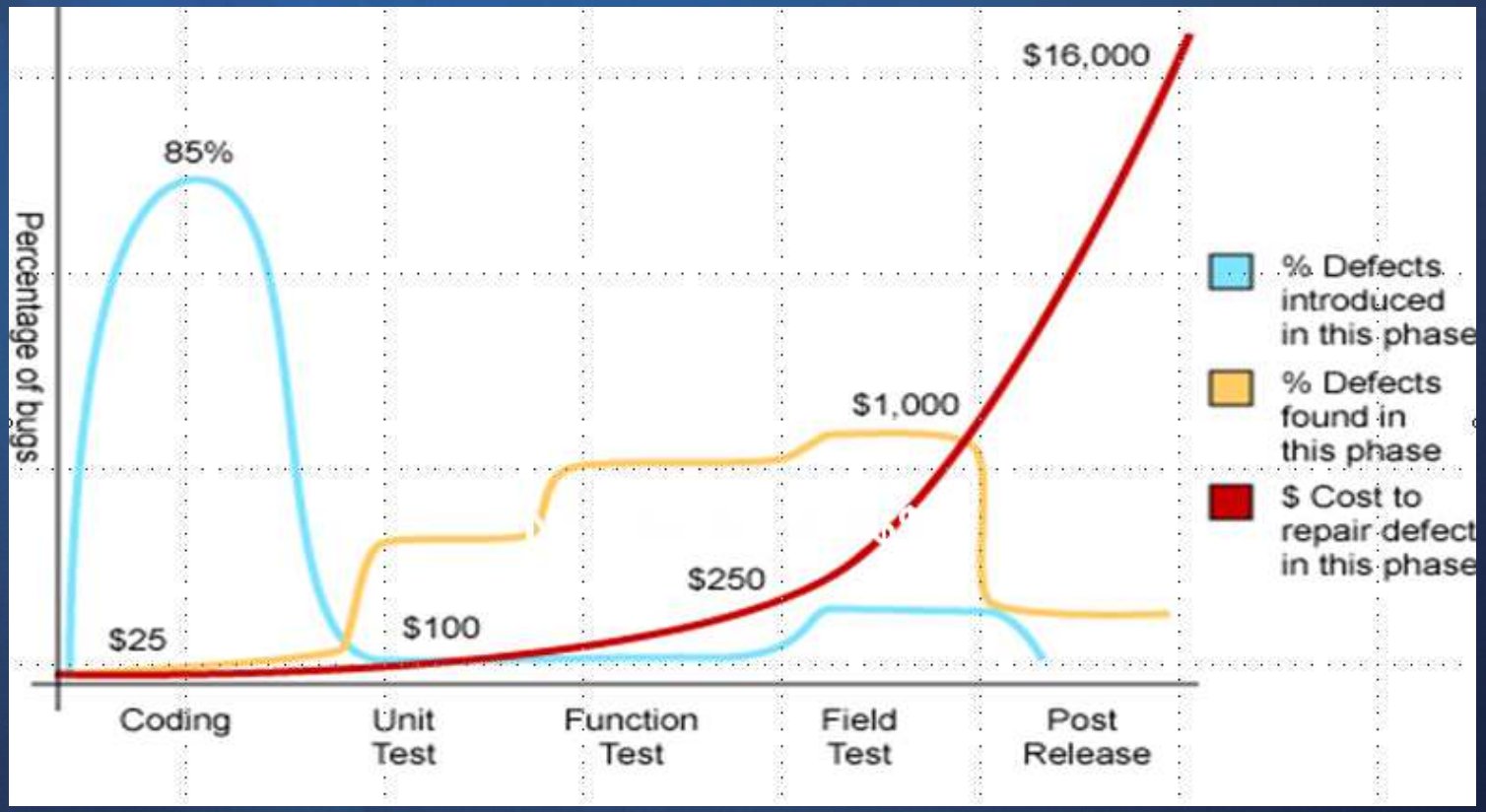
Topics Covered Today

1. Static Analysis' Promise
2. Static Analysis Tools Evaluation
3. Static Analysis Role in FW-SW Development
4. Discussions

To Start With..

1. **Static Analysis' Promise**
2. Static Analysis Tools Evaluation
3. Static Analysis Role in FW-SW Development
4. Discussions

Facts of Defects



(Applied Software Measurement by Capers Jones)

Medtronic

Way to High Confidence Coding

- Manual review of the entire code base is an impractical task.
- Nearly impossible for VT to test every eventuality that the software could lead to.
- Static Analysis provides a mechanism to automatically accomplish the daunting task of covering every scenario.
- Static Analysis is an effective approach to evaluate the code without executing the program and testing the intrinsic limits of the program, as opposed to dynamic analysis where the software is executed and is bounded by user test scenarios.

Static Analysis Approach

- Thoroughly examining the code (100% path coverage) –white box approach. *Catches problems that test suites may miss*
- Revealing errors that do not manifest themselves in testing process, but until an unusual set of conditions met (often after release). *Catches bugs early, when they are less expensive to fix*
- Automation of code review, coding standard (Power of 10, MISRA, etc.) enforcement and reliability metrics. Pinpoints defects automatically, *improving productivity*

Methodology I - Data flow (symbolic execution)

- Defect example 1: Memory leaks
- Defect example 2: Buffer overflow
- Defect example 3: Infinite Loops
- Defect example 4: Division by zero

Methodology II - Pattern match

- Defect example 5: Cast alters value
- Defect example 6: Uninitialized variables
- Defect example 7: Global valuables
- Defect example 8: multiple threads conflict

Methodology III - Customized Rules

- Rule example 1: MISRA
- Rule example 2: The Power of 10 (NASA-JPL)
- Rule example 3: Our coding standards
- Rule example 4: Metrics
  - Code complexity
  - Code coupling visualization and statistics
  - Code stability by # of warnings
  - Code / specification match

Example 1

- Wrong Operator

```
// clear bit 7
#define BIT_MASK 0x80
unsigned int my_val = 0xffff;

// incorrect code using logical negation operator
my_val &= !BIT_MASK;

// correct code using bit-wise complement operator
my_val &= ~BIT_MASK;
```

Example 2

- Inappropriate Cast

```
int char_io() {
        char c;
        c = getchar();
        return c;          // which is a char
}
```

Example 3

- Memory Leak

```
int leak_example(int c) {
        void *p = malloc(10);
        if(c)
                return -1; // DEFECT: “p” is leaked
        /* ... */
        free(p);
        return 0;
}
```

We Are Getting Into…

1. Static Analysis' Promise

2. Static Analysis Tools Evaluation

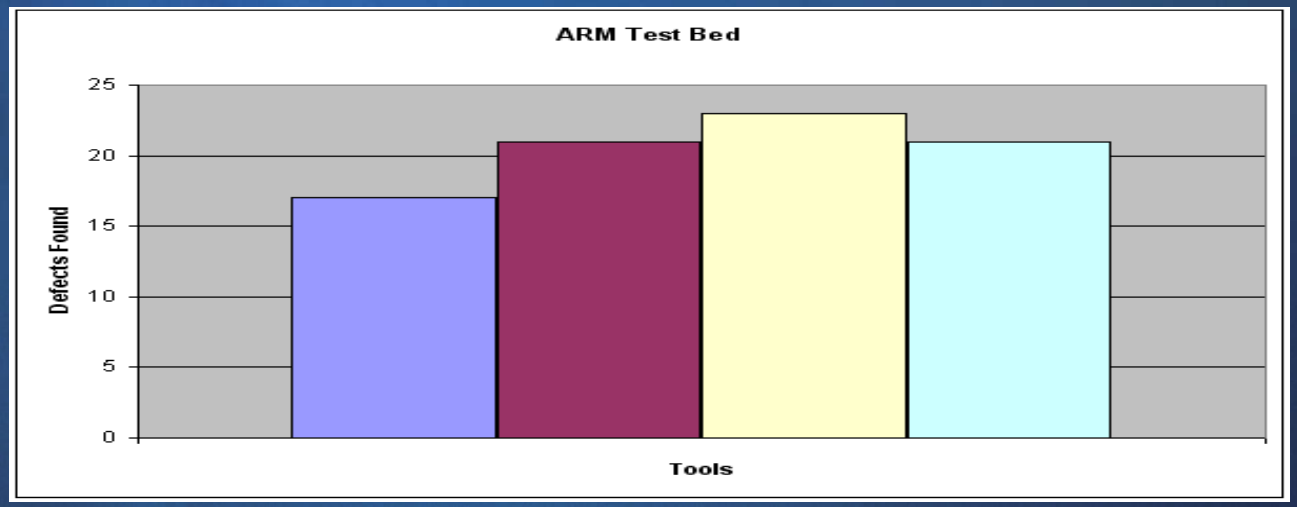
3. Static Analysis Role in FW-SW Development

4. Discussions

Static Analysis Tool Effectiveness

- False Negative (Soundness of Tool)

  Out of 29 pre-seeded bugs



- Overall robustness: 90~95%
- "No one tool meets our needs but a suite can, when used together, combine the high points of each to provide us a workable approach to investigation." (FDA)

Static Analysis Tool Effectiveness (cont.)

- Worst cases:
  - TestSpec - 01 - Expression always evaluates to False
  - TestSpec - 12 - macro not parenthesized
  - TestSpec - 15 - suspicious constant
  - TestSpec - 42 - fail to leave Critical Section
  - TestSpec - 43 - fail to enter Critical Section

Tool Efficiency

- False positive (Measure of Completeness)
  - Ability to strike the right “balance” between false positive and false negative is important
- IDE integration (work flow)
- How to deal with inline assembly code

Usability

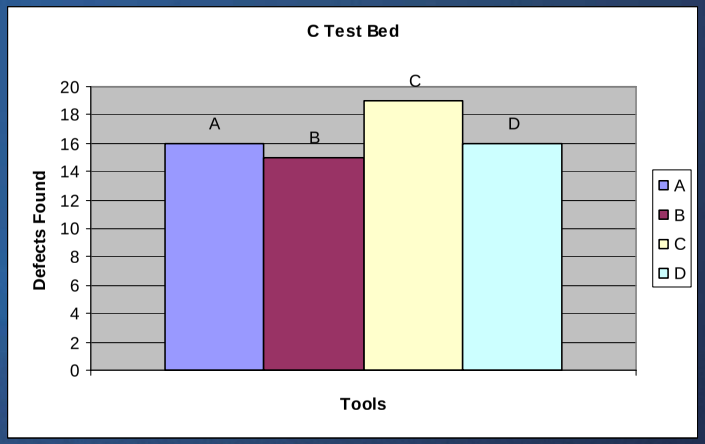
- Reporting
- Tracking
  - Code relationship
  - Coding history
- Administration
- Customizing
  - Enable/disabling checks – group or individual
  - Additional checks
- Other factors

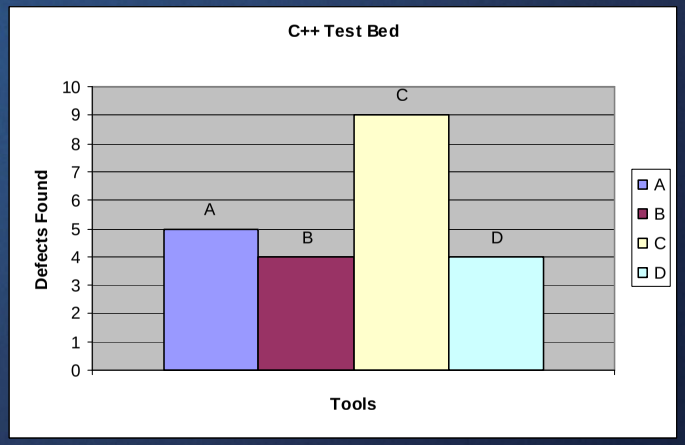
Evaluation Process

- **6 Selection Criteria**
  - Robustness
  - Integration/configuration to current development environments
  - ROI
  - Platform coverage
  - Usability
  - Customization
- **3 Bug-seeded Test Beds**
  - Archimedes 68HC11
  - ARM RVDS compiler for Cortex M3
  - Windows-based
- **43 Types of Errors to check**
  - 24 in 68HC11
  - 29 in ARM Cortex M3
  - 13 in Windows-based
  - Errors were based on inputs from developers, industry and FDA field reports.

Results of the evaluation

- **Vendors on radar screen**
  - Coverity, CodeSonar, IBM, Klocwork, Parasoft, Polyspace, Lint etc.
- C and D (see chart) are the top two vendors.
- Use of objective score card to drive consensus.
- Validated our hypothesis: Static Analysis is a valuable tool to improve SW reliability.





Medtronic

Top Two Tool Comparisons

Comparison	C	D
Effectiveness - Data Flow		x
Effectiveness - Patton Match	x	
Usability - Performance (speed)		x
Usability - reporting	x	
Usability - Integration	X (w/ higher cost)	x
Side Feature - Unit Test	x	
Side Feature - Architecture Analysis		x
Maintenance Cost		x
Fixed License Policy	x day incremental	y day idle
Overall Pricing		x

We Are Getting Into…

1. Static Analysis' Promise

2. Static Analysis Tools Evaluation

3. Static Analysis Roles in FW/SW Development

4. Discussions

SA Roles Beyond Reliability

- In design review
- In verification test
- In increasing productivities
- In helping code development

Roles in Design Review

- No intent to use “pure” static analysis in the software design review.
- While static analysis is good at finding logical errors, it is not “sensitive” to design mistakes.
- Extended capabilities may help in the design review

Roles in Verification / Validation

- Supplement not replacement of verification test
- Fits well in Test-Driven Development (TDD)
- Not used in validation

Roles in Increasing Dev Productivity

- Code inspection
  - Take the burden away from developers of tedious and repentant code checking.
- Code Review
  - Reduce/eliminating coding error so code review can focus on correctness of implementing specifications.
- Regression test in incremental releases
  - Speed up the regression test
- Design for Reliability

Roles in helping development

- Reliability metrics
  - Code complexity
  - “Dead” code
  - Root cause analysis (RCA)
  - Etc.
- Unit test
- Architecture analysis

Roles in helping development (cont.)

– IEC 62304 Compliance (Medical Device Software – Software Life-cycle Process)

Description	Specific SA notes
6. Verify that all counters maintained within the unit are properly incremented or decremented with successive calls to the unit (no infinite looping and proper termination).	Checks for intfinite loops.
7. Verify that the unit properly handles exceptions/faults: a. Arithmetic faults i. Divide by zero ii. Numeric overflow/underflow b. Array size limits c. Freeze/Wrap counters d. Data Corruption i. Negative numbers ii. Invalid input values (confirm that e_faults are working correctly).	SA tool can be used as a suppliment to Unit Test to mitigate the risk.
9. Verify locking and unlocking of memory/registers is correct. (Memory/register locked/unlocked status is correct when the unit completes).	Some tools can probably be customized to flag unmached lock/unlock pairs.
10. Verify units that write to buffers (e.g. Dx block moves) do not exceed the buffer size.	
Error Guessing - Test areas where things may go wrong, from a design perspective.Brainstorm possible test cases that stress the design. Abort/resume, nested aborts, queue/stack overflow, etc.	Can the tool pick up potential stack uoverflow, infinite loops, recursion, etc
Verify correctness of stack utilization	
d) For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.	Some tools can check correct usage of loop variable used as array index
Although it's typically expected that most of the coverage objectives identified in this section to have been previously verified by other FW test activities (Unit & Integration testing), it's good to reconfirm at this point if any of these items warrant additional focus due to any remaining known risks.	

We Are Wrapping Up…

1. Static Analysis' Promise
2. Static Analysis Tools Evaluation
3. Static Analysis Roles in FW/SW Development
4. **Discussions**

Discussion I – False Positive

- False negative is a more critical concern
- Over suppression of false positives is NOT a good practice
- Philosophical understanding of what FPs are
- Server level tuning
- Promote best practices in software development
- Raise the bar of acceptable coding habits

Discussion II – Integration

- Bug tracking system
- Code repository system
- Unit test

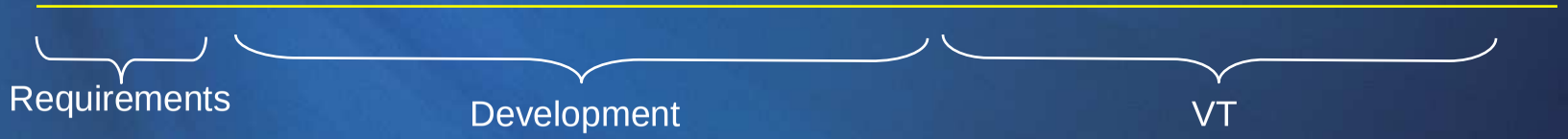
Discussion III – Platforms

- Use model: Two Tier Model as a minimum
- Sensitivity of Methodology to platform
- FW on-target testing
- Security
- Limitation of Static Analysis in assembly
- Compiler independent

Design for Reliability and Manufacturability (DRM) in SW

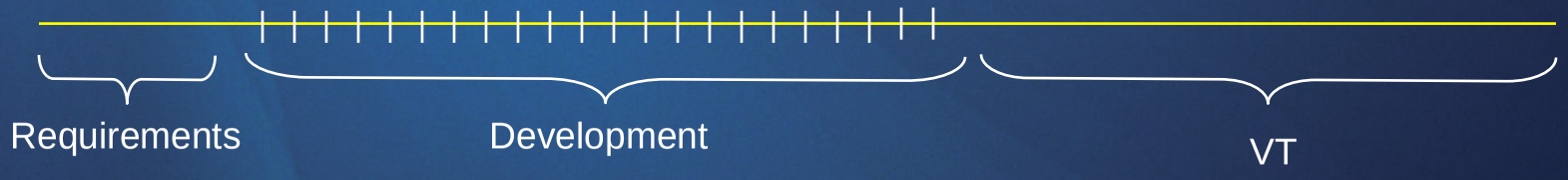
Past

VT is separated as ‘end of cycle’ activity



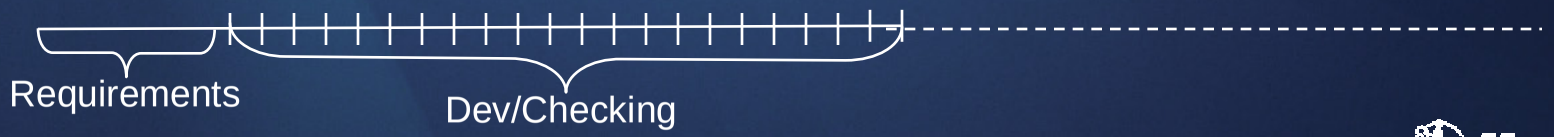
Current

Coding and VT processes integration



Future

Minimize/eliminate code defects before VT



Medtronic