Developing Reliable Products: Static and Dynamic Analysis of the Code

Aleksandr Kuzmin

Java ME Software Quality Architect
Sun Microsystems, Inc.

Member of American Society for Quality

Mikhail Davidov

Java ME Software Quality Manager
Sun Microsystems, Inc.

TS-5711

Analytical Tools Improve Code Quality

Proper application of static and dynamic tools, during development and test execution, significantly improves the reliability of products

Agenda

- Static analysis
  - Theory
  - Benefits
  - Examples
- Dynamic analysis
  - Theory
  - Benefits
  - Examples
- Quality wins !

Why Do We Need Such Tools ?

- Many software defects don't manifest themselves during regular testing; Software with subtle problems may run flawlessly on one platform, but crash on another
- Stability of the code is critical for embedded software and server applications

Static Analysis

- Static analysis helps to detect defects beyond the limits of runtime coverage
- Analytical tools report potential errors by modeling dynamics of software applications relying solely on the source code
- In addition to market tools, Sun develops internal,customized static analysis instruments

Static Analysis: Why It's Important

- Static analysis may detect defects that are not reachable by functional test coverage
- Benefits:
  - All blocks and execution paths can be analyzed
  - All data ranges can be tested
  - No instrumentation of the code
  - No tests to develop

Static Analysis: Call-graph

- Control Flow Graph (CFG) is just a different representation of the program source—it's built on syntax tree of the program and defined constraints over variables assigned to nodes
- A CFG is a directed graph, in which nodes correspond to program points and edges represent possible flow of control

Static Analysis: CFG—Dataflow Analysis

- Dataflow analysis considers CFG with dataflow constraints that relate to the values of the variables of the corresponding CFG nodes
- When we consider the whole chain of function calls, the analysis is called inter-procedural

Static Analysis: Data Ranges

- There are interesting values to inject into given code sections that can trigger bugs and check for boundary values
- Some static analysis tools (e.g. PolySpace) may automatically determine if all callers to a given method only pass safe values to the method

Typical Errors Detected by Static CFG Analysis in Native Code

- Illegal pointer access to variable/structure
- Array index within bounds
- Non-Initialized Variable/Pointer
- User assertion
- Overflows/Underflows, division by zero
- Wrong number, wrong type for arguments
- Non-termination of Call or Loop
- Unreachable code

DEMO

Static Analysis

Static Analysis: Example 1

```
154     if (msgID > 0 && handle != 0) {
155
156         /* Unblock any blocked threads. */
157         cbsUnblockThread((int)handle, WMA_CBS_READ_SIGNAL);
158
159         /* Unregister the CBS port from the CBS pool. */
160         unregisterCBSMidletMsgID((unsigned short)msgID);
161
162         /* If the handle hasn't been created, the connection isn't open. */
163         if (handle == 0) {
164             status = -1;
```

Unreachable code

Static Analysis: Example 2

```
64      getLcConvMethodsID(jchar *uc, int len) {
65          char enc[16];
66          int i;
67
68          for (i = 0; i < len; i++) {
69              enc[i] = (char) uc[i];
70          }
71          enc[i] = 0;
```

Out of bounds if 'len' > 16

Static Analysis: Example 3

If statement condition contains assignment. (Severe Violation: (pbugs-21, IfAssign.rule))

```
720  if ((temp1.ptr[0] == 1) && (temp1.size = 1)) {
```

Assignment operators shall not be used in expressions which return boolean value

(Violation: (misra-035, AvoidAssignmentsInBooleanExpr_MISRA_035.rule))

Static Analysis: Example 4

XDecode.c
186
187 CRC = readHeader(src, chunkLength, &data, CRC);
188

in "XDecode.c" line 187 column 10
the readHeader call never terminates:

```
readHeader():
...
 while (length > 0) {
     int n = (length < (long)sizeof(buf)) > length : n;
     crc->getBytes(src, buf, n);
     CRC = update_crc(CRC, buf, n);
 }
...
```

Static Analysis: Summary

- 75% of bugs are local (intra-procedural)
- Finding real integration bugs might require too heavy inter-procedural analysis
- At least 30% of unreachable code reveals real bugs
- First 20% of review time classifies 60% of false positives
- 80% of warnings can be easy evaluated

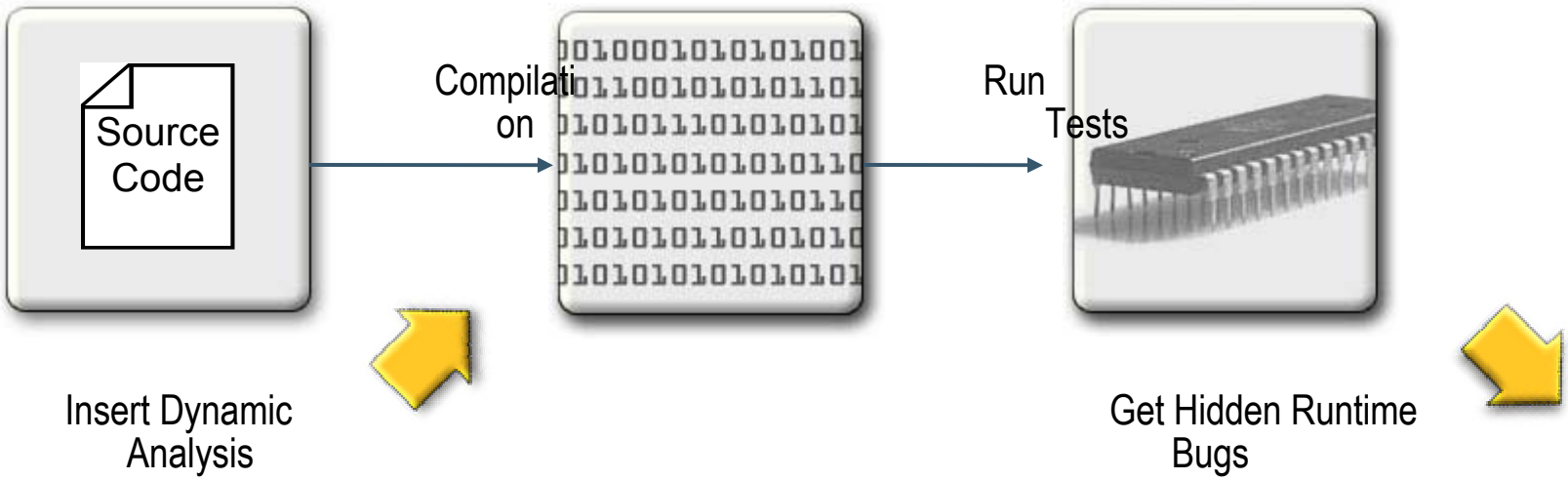
Dynamic Analysis

- Dynamic analysis helps to identify the source of the problem much faster than intensive stress testing
- Dynamic analysis discovers real problems with high precision (vs. potential defects) using its instrumentation of the code and analysis of all memory operations

Dynamic Analysis: Why It's Important

- Memory leaks or illegal pointer operations may not be noticed during functional nor stress testing, but may cause problems in production deployment
- Benefits:
  - Low rate of false positives
  - Easy to automate

Dynamic Analysis: How It Works



The tools insert some analysis code at every line of source code; They build a database of all program elements, and then at runtime, the tools check each data value and memory reference against its database to verify consistency and correctness

Typical Errors Detected by Dynamic Analysis

- Memory Leaks
- Invalid Pointers
- Memory Corruption
- Memory Overflow
- Reading/Writing Uninitialized Memory
- Unused Variables/Arguments
- Data Formatting Problems
- Unexpected Errors
- Invalid Arguments
- Invalid System Calls

DEMO

Dynamic analysis

Dynamic Analysis: Example 1

```
keyPtr = "com.sun.midp.io.j2me.sms.DatagramPortOut";
valuePtr = strdup(serverTrafficPort);
setInternalProp(keyPtr, valuePtr);
setSystemProp(keyPtr, valuePtr);
```

Memory leak: *valuePtr* was not free()-ed

Dynamic Analysis: Example 2

```
else {
    carry = (t1 < t2);
    t1=(t1 - t2);
}
```

t1 was not initialized

Dynamic Analysis: Statistics

- Low rate (10%) of false positives
- Up to 1 defect per 3-10 Klines_of_code

Dynamic Analysis: Summary

- As usual, a regular memory debugger is helpful when the program already has reproducible crash; instead, dynamic analysis helps to find hidden defects automatically during regular functional testing
- The precision and breadth of dynamic analysis is limited only by runtime test coverage of the product

False Positives

- Analytical tools may report “false positives”
- This is OK, because:
  - General-purpose tools may not know about specific assumptions in your code
  - Their rules might be too general
  - The warnings might be over-prioritized

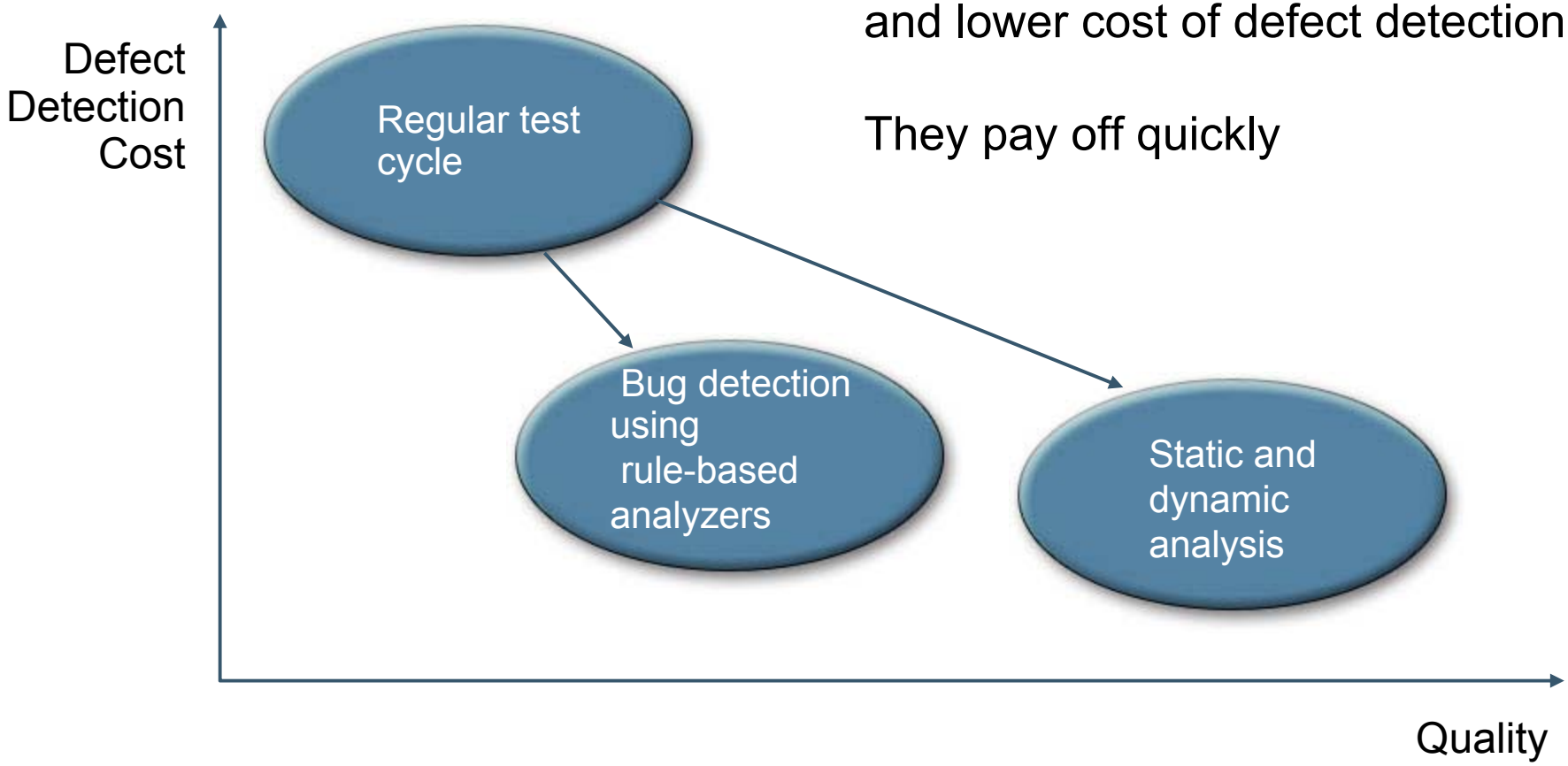
How to Manage False Positives

- False alarms can be suppressed by:
  - Customized filters
  - Proper prioritization of the rules
  - Better stub-functions
  - Analyzing the whole system instead of a single module
  - Modify data ranges

Summary

Static and dynamic analyses help to significantly improve quality and lower cost of defect detection

They pay off quickly



Our Message to JavaOne^{SM} Conference

Sun Java™ technology product testing teams apply various static and dynamic analysis in software development process to find bugs earlier and to build quality into the software that we deliver

Tools Available for Java Code and Native Code

www.sun.com

www.parasoft.com


http://findbugs.sourceforge.net

www.enerjy.com

www.coverity.com

www.polyspace.com

www.klocwork.com


http://valgrind.org/

www.ibm.com

See Also

- ***TS-9667: Testing Java Code: Beyond the IDE, Wednesday, 2:50pm***
- ***BOF-9587: Pimp My Java Application: Applying Static Analysis Tools to Boost Java Code Quality, Wednesday, 7:55pm***
- ***BOF-9231: FindBugs BOF, Wednesday, 8:55pm***
- ***TS-2007: Improving Software Quality with Static Analysis, Wednesday, 1:30pm***

Q&A

Developing Reliable Products: Static and Dynamic Analysis of the Code

Aleksandr Kuzmin

Java ME Software Quality Architect
Sun Microsystems, Inc.

Member of American Society for Quality

Mikhail Davidov

Java ME Software Quality Manager
Sun Microsystems, Inc.

TS-5711