Tracking Down Software Bugs Using Automatic Anomaly Detection

Sudheendra Hangal and
Sun Microsystems India Pvt

Monica S. Lam
Computer Systems Laboratory
Stanford University

ICSE 2002

Presented by:
Fahad Arshad

PURDUE UNIVERSITY

Motivation

- Undetected errors which compromise the results of a computation can be dangerous
  - can lead to catastrophes (e.g Therac-25, Ariane 5 Flight 501)
- Integration of many components often lead to rare bugs which can take days or weeks to debug (*rare corner cases*)
- Detecting bugs and hunting for root cause is important

PURDUE UNIVERSITY

Previous Work

- [Ernst et al] detects likely program invariants based on dynamic program behavior .
- It starts with a specific space of program invariants.
- The Daikon tool, is limited by the fixed set of invariants hypothesized and checked for.

DIDUCE (Dynamic Invariant Detection U Checking Engine)

- DIDUCE extracts invariants dynamically from program executions
- DIDUCE continually checks program behavior against the invariants hypothesized up to that point in the program's run and reports all detected violations.
- When a dynamic invariant violation is detected, the invariant is relaxed to allow for the new behavior and program execution is resumed.

Usage Models for DIDUCE

- Debugging programs that fail on some inputs:
  - DIDUCE can pinpoint differences in behavior between the successful and the failing runs
  - Extract invariants from test cases that pass
  - Check invariant violations for cases that fail (reduces debugging time)
- Debugging failures in long-running programs:
  - Some of the hardest bugs to track down are those that occur only after a program has executed for a long time.
  - DIDUCE blindly and continually monitors all the variables in the program

Usage Models for DIDUCE (continued)

- Debugging component-based software:
  - For component-based software, train DIDUCE on codes with same components working correctly, and apply it to check the behavior of a component in the context of the new software
- Testing programs with inputs for which the correct outputs are unknown:
  - Train DIDUCE on known tests cases, and use the invariants gathered to check the runs on inputs with no known outputs
- Assisting in program evolution:
  - Check the invariants collected before and after the update in a program

DIDUCE Invariants

- DIDUCE system instruments Java programs
  - Maintains invariants on the values of a set of *tracked expressions* at various program points
  - An invariant hypothesis on an expression is satisfied by all the values that have occurred in the history of the execution so far.
  - Invariant is relaxed on seeing a new violating value.
- DIDUCE operates in two modes
  - Training Mode:
    - DIDUCE silently learns invariants by relaxing invariant hypotheses as needed
  - Checking Mode:
    - DIDUCE emits messages about invariant relaxations which occur along the way
    - Training continues in checking mode as well

DIDUCE:Instrumented Program Points

- DIDUCE associates invariants with static program points
- DIDUCE allows tracked expressions to be attached to:
  - program points which read from or write to objects
  - program points which read from or write to a static variable
  - procedure call sites
- This design gives visibility to global state of computation
- User provides JAR files, DIDUCE will instrument all the static program points described above.

DIDUCE: Tracked Expressions

- At each instrumented program point
  - a set of expressions is maintained, each of which is a function of the object or variable being accessed.
  - An invariant is maintained for each expression in this set, starting with the strictest invariant assumption at the beginning
  - Gradually relaxes the invariant to encompass the values observed for the expressions.
- Following expressions are tracked by default:
  - the value being read or written
  - parent object, in the case where a field of an object is accessed
  - the difference between the values of the location accessed before and after a write operation.

PURDUE UNIVERSITY

DIDUCE: Tracked Expressions

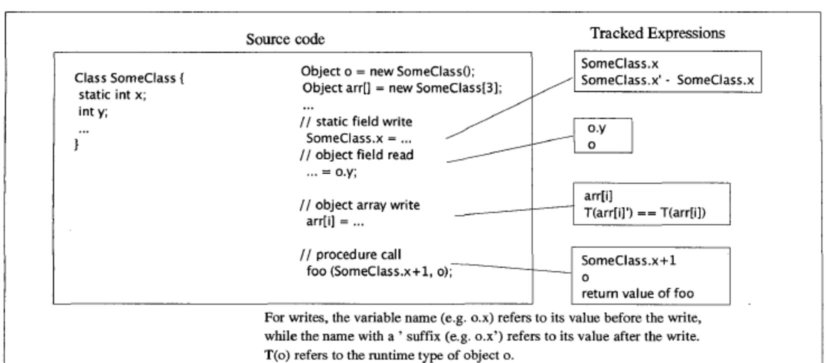

Fig. 1: Some Examples of Tracked Expressions

- For tracked expressions which are of reference type, map objects to their run-time types
- Null values are treated as a special run-time type of their own.

PURDUE UNIVERSITY

DIDUCE: Invariant Representation

- Values of all expressions of all types reduced to integers
- Reference type expressions are mapped to an integer which is the hashcode of the String object for their run-time type
- For each expression's value, the invariant maintains for each bit position two things.
  - 1) the value of that bit the first time the expression was evaluated,
  - 2) whether different values have been observed for that bit position
- A violation is reported if differences between the new value and previous ones are observed in new bit positions

PURDUE UNIVERSITY

DIDUCE: Invariant Representation

- With each expression a tuple of two integers, an initial value V and a mask M is associated
  - The ith bit in M is set to 1 iff the same bit value has always been observed for that position.
- If the first value of an expression is W then,
  - $M := \neg 0$, $V := W$
- Suppose subsequently an expression returns $W`$
  - If $(W` \text{ XOR } V) \wedge M \neq 0$
  - Then a violation is reported and invariant is relaxed by $M := M \wedge \neg (W` \text{ XOR } V)$

PURDUE UNIVERSITY

DIDUCE: Invariant Representation

- DIDUCE keeps track of following properties
  - whether the values were only positive or only negative, only odd or only even
  - an approximate upper bound on the value
  - which of the bits have constant values
- With this representation, the number of violations detected for each expression can be no greater than the number of bits in a word
- The storage required for maintaining invariants is about three words per tracked expression

DIDUCE: Invariant confidence

- *Confidence level* of an invariant is defined as the ratio between the number of times the expression has been evaluated and the number of values the invariant accepts
- Every invariant violation is reported with the change in confidence levels between the old invariant and the newly relaxed invariant.
  - A large drop in confidence signals a noteworthy invariant violation.
- Code executed for the first time is reported with a fixed, user-specifiable invariant confidence change.

DIDUCE Implementation

- ByteCode Engineering Library (BCEL) is used to instrument Java class files and insert calls to the DIDUCE run time system at appropriate program points
- Source code not required but useful in understanding the invariant violations reported
- An instrumented program using the default settings currently runs one to two orders of magnitude slower

DIDUCE EXPERIENCES

Program name	Description	# Lines of Source Code	# Classes (instrumented/total)	# Instrumented program points	Slowdown factor
Simulator	Proprietary performance simulator for multiprocessor memory systems	3300	10/28	3204	8-12X (Using 10 machines)
Mailmanage	Open source mail management utility	1700 (+ ~ 20000 JavaMail library)	214/214 (203 classes in JavaMail library)	13014	6X
JSSE Library	Shipping reference implementation for Java Secure Sockets Layer Library	30000 (+ Obfuscated RSA libraries)	384/384	34844	8X
Joeq	Research project to develop a Java Virtual Machine	31500	18/137	3371	20X

Table 1: Details of programs DIDUCE was tried on

- DIDUCE was especially helpful in pinpointing late-stage bugs that occur after many test cases are run.

DIDUCE EXPERIENCES: Mailmanage

- Mailmanage crashed on a particular mail box throwing a cryptic IO Exception.
- Crash apparently occurred in the JavaMail library while trying to fetch a message from a mailbox
- Both Mailmanage and JavaMail library were instrumented
- DIDUCE was trained on a few mailboxes that worked correctly and then tested with the failing mailbox

PURDUE UNIVERSITY

DIDUCE EXPERIENCES: Mailmanage

```
do {
      switch (buffer[index]) {
           case 'E':
              index += ENVELOPE.name.length;
              // other processing for case E
              break;
           // similar handling of other cases
      }
} while (buffer[index++] != ')' );
```

Fig. 6: Sample code from JavaMail library

- Figure shows the relevant code identified by the invariant violation
- Invariant violation reported that buffer[index] contained a new value at the end of while loop.
- Prior to this violation, invariant accepted both a space character and a ")" character

PURDUE UNIVERSITY

DIDUCE EXPERIENCES: Mailmanage

- The bug in this case was not in either Mailmanage or JavaMail library. It was in IMAP server
- The response by the server contained extra CR-LF characters, which was an inconsistent with its RFC
- This confused the JavaMail parser, which eventually threw an exception.

DIDUCE EXPERIENCES: Mailmanage

- DIDUCE strengths from this case study
  - It detected an anomaly in the input
  - Helped the user debug unfamiliar code, isolating the problem down to the component which actually contained the bug
  - Helped in finding bugs in code that was not even instrumented by finding invariant violations at the interface between instrumented and uninstrumented domains.

DIDUCE EXPERIENCES: Java SSE Library

- On adding a proxy server to the library, an unseen failure in unrelated parts of the code was observed
- Programmer tried to debug by working backwards from the point of failure through the rest of the library
- After 2 days of manual debugging, she isolated the problem to a particular function
- When asked to use DIDUCE, she trained it with correct runs and ran it on the failing runs in checking mode.

DIDUCE EXPERIENCES: Java SSE Library

- DIDUCE reported a high confidence invariant on the return value of a call to SocketInputStream.read() method
- This method does not guaranteed to fill the entire array and can return after it has filled 1 or more bytes.
- The programmer had fundamental misunderstanding of the java.io library

```
InputStream  s = x; // x is instance of SocketInputStream

// .. various SSL protocol processing

if (...) {
    int len = ... // expression for  length of header,
                  // always 74 at this program point
    byte[] hdr = new  byte[len];
    s.read (hdr);
}
```

Fig. 7: Excerpt from the SSL library

DIDUCE EXPERIENCES: Java SSE Library

- DIDUCE was modified to include a simple static check for immediately discarding the return values from calls to various flavors of InputStream.read() with a byte array argument
- Over 80 such examples in the Java 2 Standard Edition and Enterprise Edition v.1.3 libraries were found, most of which were likely to be errors.
- This shows the importance of automatic invariant discovery

DIDUCE EXPERIENCES: Joeq

- DIDUCE was run in checking mode without training. Initial invariant violations were ignored
- Joeq failed an assertion while compiling a particular version of the Java Runtime Library
- Joeq read each entry in the library JAR file, processed it, and entered the name of the entry into its own hash table.

```
JarInputStream in = ...;
Hashtable names = new Hashtable()...;
for (<each entry in the jar file> ) {
  JarEntry je = jin.getNextJarEntry();
  // process entry ...
  names.put (je.getName());
}
assert (names.size() == jfile.size());
```

Fig. 8: Excerpt from joeq

DIDUCE EXPERIENCES: Joeq

- DIDUCE precisely pointed to the source of the problem
  - the return value of Hashtable.put() method indicates whether the object being inserted is already present in hash table
  - It returns the existing object if the key matches an element in the hash table, and NULL otherwise
  - The programmer implicitly assumed that the entries in a JAR file were unique, and ignored the return values
  - DEDUCE reported a warning on finding a duplicate entry thus finding the root cause of the problem

DIDUCE EXPERIENCES: MAJC Memory Simulator

- Ten classes were instrumented in the program separately, and these ten versions were run in parallel
- Invariant violations in the initial part were ignored as it was considered as the training phase
- DIDUCE discovered two bugs in the simulator that would otherwise be undetected and found the root causes of 3 other bugs
- All bugs were serious algorithmic errors

DIDUCE EXPERIENCES: MAJC Memory Simulator

- "New code" category
  - tracks when execution reaches a program point for the first time
- High confidence invariant violations refer only to the violations above the confidence change level of 100
  - Last stub in the figure is the only bug with a confidence change of over one million

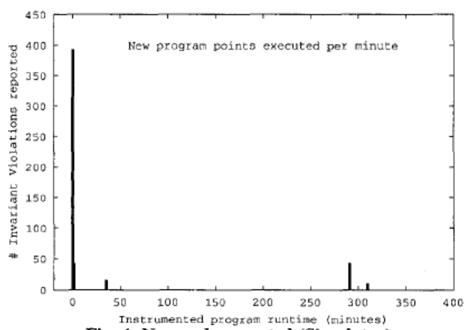

Fig. 4: New code executed (Simulator)

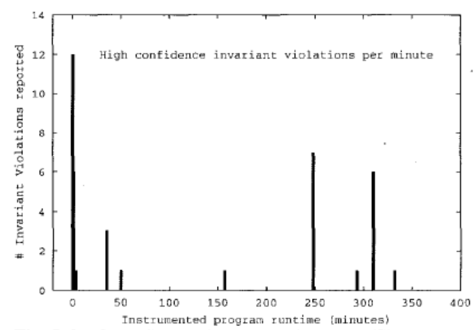

Fig. 5: High confidence invariant violations (Simulator)

PURDUE UNIVERSITY

Conclusions

- Finding program anomalies through online dynamic program invariant detection and checking engine
- DIDUCE is effective in detecting hidden errors and finding the root causes of complex programming errors.
- Finding bugs that result from algorithmic errors, errors in inputs, and developers' misconceptions of the APIs.
- Helps programmers locate bugs in unfamiliar code and, sometimes even in codes that has not been instrumented.

PURDUE UNIVERSITY

QUESTIONS ????