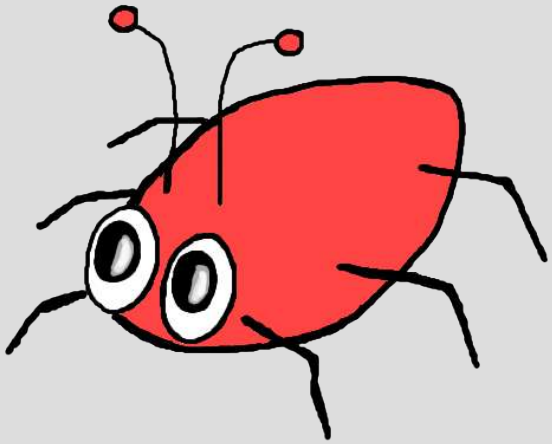
Student Repository Mining with Marmoset

Jaime Spacco
Jaymie Strecker
David Hovemeyer
Bill Pugh

UNIVERSITY OF MARYLAND



Overview

- Student project repository
  - granularity of each save
  - full set of unit tests
- Use repository to validate and tune static analysis for software defects
  - what can we predict using static analysis
  - false positives and negatives

Take away messages

- Student project repositories are uniquely valuable
- We can show that many static detectors are very accurate at predicting faults in student code
  - many faults are not predicted by static analysis
- Want to track occurrences of program features / warnings / faults across versions
  - still working on this

Student project repository

- Get snapshots at each save within Eclipse
- Almost all students participated in study
  - don’t experience course differently
- All sorts of cool stuff about how we collect data
  - no time to talk about it
- 33,015 unique compilable snapshots
  - from 569 student projects
    - 73 students, 8 projects

Analysis and Testing

- Each snapshot is run against all unit tests for that project
  - total of 505,423 test runs
  - exceptions and failures stored in database
  - will be collecting code coverage data from each test
- static analysis tools are run against each snapshot
  - FindBugs, others

Test results

67,650	not implemented
63,488	exception in student code
138,834	assertion failed
235,448	passed

Failure distributions

# snapshots	# projects	Cause of Failure
33,015	569	*max possible*
24,030	528	AssertionFailed
6,132	318	NullPointer
2,176	81	OutOfMemory
2,139	159	ClassCast
2,111	118	StringIndexOOB
1,815	78	IllegalArgument
1,683	61	IOException
1,601	124	IndexOOB
1,419	122	StackOverflow
1,353	54	InvalidRequest

Some FindBugs warnings predict specific Exceptions

- InfiniteRecursiveLoop predicts StackOverflow
- BadCast predicts ClassCast
- Others don't
  - a switch fallthrough doesn't predict any specific runtime exception

- How often does a warning `W` in a snapshot predict the corresponding exception `E` in some test run of that snapshot?

Infinite Recursive Loops

```
// Construct new Web Spider
   public WebSpider() {
    WebSpider w = new WebSpider();
     }
```

- Surprisingly common
- Wrote a detector for this last fall
  - caught a number of cases in student code
  - caught 3 cases in Sun's Java JDK

Statistical Correlation

Expected # of cases where exception and warning both occur: 43

$X^2 = 1748$

by chance: epsilon%

		Stack Overflow Exception	
		yes	no
Recursive Loop Warning	yes	626	187
	no	793	31,409

Class Cast Exception

Expected # of cases where exception and warning both occur: 102

$X^2 = 1{,}518$

by chance: epsilon%

		Class Cast Exception	
		yes	no
Bad Cast Warning	yes	474	1,431
	no	1,296	29,340

Problems Counting

- Guessing about correspondence
  - don’t (yet) compare warning location, exception location and code coverage
- Defect may not be matched by corresponding exception
  - some code defects do not cause immediate exceptions
  - errors can be masked by other errors
- False positives aren’t fixed in successive versions and are thus over counted

Tuning detectors

- By looking at false positives and false negatives, were able to tune detectors
  - an ongoing process
- Runtime exception data allow us to look at false negatives
  - something we hadn’t been able to do before
- A runnable software repository is far more interesting than a dead one

Student faults predict production faults

- Stuff we are learning from student code is helping us find errors in production code

- Tuned infinite loop detectors found defects in lots of production software:
  - 1 more in Sun's JDK
  - 13 in JBoss 4.0.2
  - 14 in Websphere 6.0.3
  - 4 in Eclipse 3.1m6
  - 14 in NetBeans 4.1rc

Instead of counting snapshots, count changes

- Look at changes between successive snapshots
  - If a change introduces/removes a bug warning
    - is it matched by a corresponding introduction/removal of a exception or failure?
  - if a change introduces/removes an exception or failure
    - is it matched by a corresponding introduction/removal of a bug warning?

Bad Cast Warning changes

- 233 changes in presence of a Bad Cast warning
- 79 of which correctly predicted change in existence of ClassCast exception
  - 66% false positive
    - better, but more tuning may be possible
- 435 changes in presence of a ClassCast exception
  - 82% false negative
    - use of generics may help students eliminate a class of errors we currently don't detect

Overall predictions

- Can we match changes in set of warnings to changes in number of passing test cases?
  - harder: change may fix one bug only to reveal another
- Rank individual detectors based on how well they do
  - use first k detectors

Cumulative prediction of all faults and errors

Detector	Total hits	expected hits	total predictions	Hit rate
SF_SWITCH_FALLTHROUGH-M	4	0	4	100%
DM_EXIT-L	19	3	26	73%
IL_CONTAINER_ADDED_TO_ITSELF-M	21	3	29	72%
RE_BAD_SYNTAX_FOR_REGEXP-M	35	6	52	67%
BC_IMPOSSIBLE_CAST-H	39	7	59	66%
ICAST_IDIV_CAST_TO_DOUBLE-M	88	17	152	58%
NO_NOTIFY_NOT_NOTIFYALL-L	89	17	154	58%
IL_INFINITE_RECURSIVE_LOOP-H	160	34	298	54%
DM_EXIT-M	167	35	313	53%
NP_IMMEDIATE_DEREFERENCE_READLN-	179	38	341	52%
BC_UNCONFIRMED_CAST-H	201	44	394	51%
NM_METHOD_NAMING_CONVENTION-L	210	47	416	50%
NP_ALWAYS_NULL-H	238	55	487	49%
EC_BAD_ARRAY_COMPARE-M	240	55	492	49%
UWF_UNWRITTEN_FIELD-H	307	74	656	47%
QF_QUESTIONABLE_FOR_LOOP-M	311	75	667	47%

Appraisal of results

- Some detectors very accurate at predicting faults in test cases
  - we believe other detectors are useful in finding faults, just harder to automatically show correlation
- Most faults are *not* predicted by FindBugs warnings
  - didn't expect them to be
  - many ways to get program logic wrong
    - not detectable by general purpose defect detector
    - could do better with detectors targeted to project

Track exception or warning between versions of file

- If we generate a NullPointer dereference warning in 2 successive versions
  - is it the same line of code?
  - What if a comment on the line has been changed?
- Also track correspondance of exceptions, code coverage, etc

Tracking Lines Across Versions

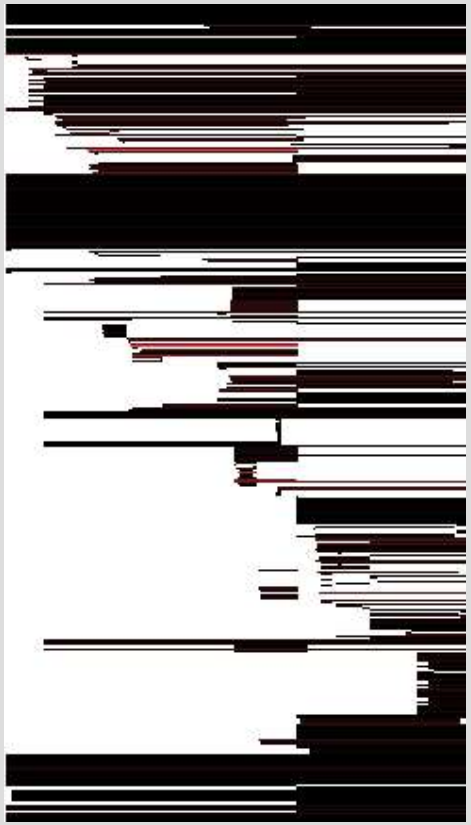
- Want to track lines of code across all the versions they occur in
  - recognize lines that are slight modifications of lines in previous versions
  - Accommodate
    - reformatting,
    - small edits,
    - rewriting comments,
    - commenting a line in/out

Results

- Small studies
  - not yet run on entire repository
- 32% of 2,023 changes were recognized as a change of an existing line
  - of those, 40% were changes to a line already changed
- Some lines were modified 5, 8 or even 12 times

Visualization

- Each line corresponds to a unique line
  - red lines were modified several times



Wrap up

- Unique and valuable repository
  - fine grained
  - executable with unit tests
    - ideally, full test coverage
- Expanding the repository
  - more courses added Spring 05 (C in one course)
  - even more in fall 05
    - student effect of generic classes from Java 5.0
  - might expand to other universities
  - data from one project being made available now

Thank you!

Questions?

Using line tracking

- Match static and dynamic information across versions
  - static warnings, code coverage, runtime exceptions
- Trace history of student decisions
- Understand programming activity

Pinpoint tracking

- We can get stack trace information that will pinpoint the exact bytecode that causes an exception
- For example, determine what, exactly, caused a null pointer exception
  - null value returned by method
    - which method
  - null value loaded from field
  - null parameter

Null Pointer detector

Expected # of cases where exception and warning both occur: 414

		Null Pointer Exception	
		yes	no
NP Warning	yes	1,168	2,033
	no	3,100	26,714

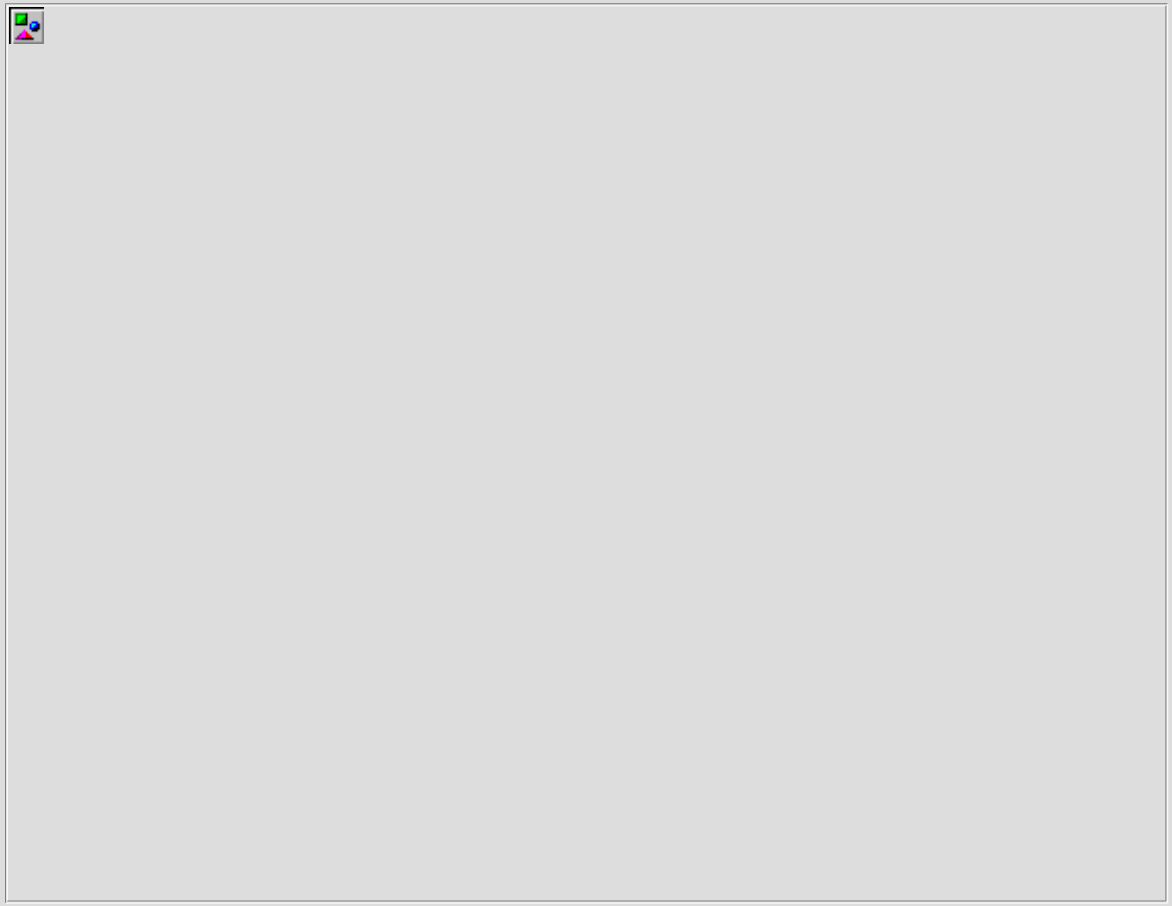
Marmoset

- Course Project Manager Eclipse Plugin
  - Takes a “snapshot” of student's code at each save to a central CVS repository
- SubmitServer
  - Students can upload projects for testing against a suite of unit tests
  - We do automated testing in a really cool way!
    - We unfortunately don't have time to cover it
  - Basically:
    - JUnit tests
    - Static checker: FindBugs

FindBugs

- Open source static checker
  - findbugs.sourceforge.net
- Bug-driven bug finder
  - Start with a bug, build analysis from there
- We've found hundreds of bugs in production software
- Everyone makes stupid mistakes
  - and we can find them with simple analyses

Overall numbers, Fall 2004
2nd semester OOP



Magnitude of Commits

# Lines Added or Changed	Number of Commits	Total %
1	12,873	39%
2	5,484	56%
3-4	4,726	70%
5-8	3,608	81%
9-16	2,503	88%
17-32	1,229	92%
33-64	612	94%
65+	352	95%

Lots of data!

- What to do with it?
  - Validate tools for software checking
  - Understand student coding practices
  - Understand software changes
  - Track changes/errors/warnings across versions
  - Plus more things we haven't though of yet

Types of failure

- Failed:
  - `AssertionFailedError`
    - how JUnit signals a test case failed
  - Exception that originates in the test driver code
    - For example, student function inappropriately returns null the test driver dereferences a null pointer returned by student code
- Error:
  - Exception in the student's code, i.e.
    - `NullPointerException`
    - `ClassCastException`
    - `etc`