Bug Driven Bug Finding

Chadd C. Williams
Jeffrey K. Hollingsworth

Overview

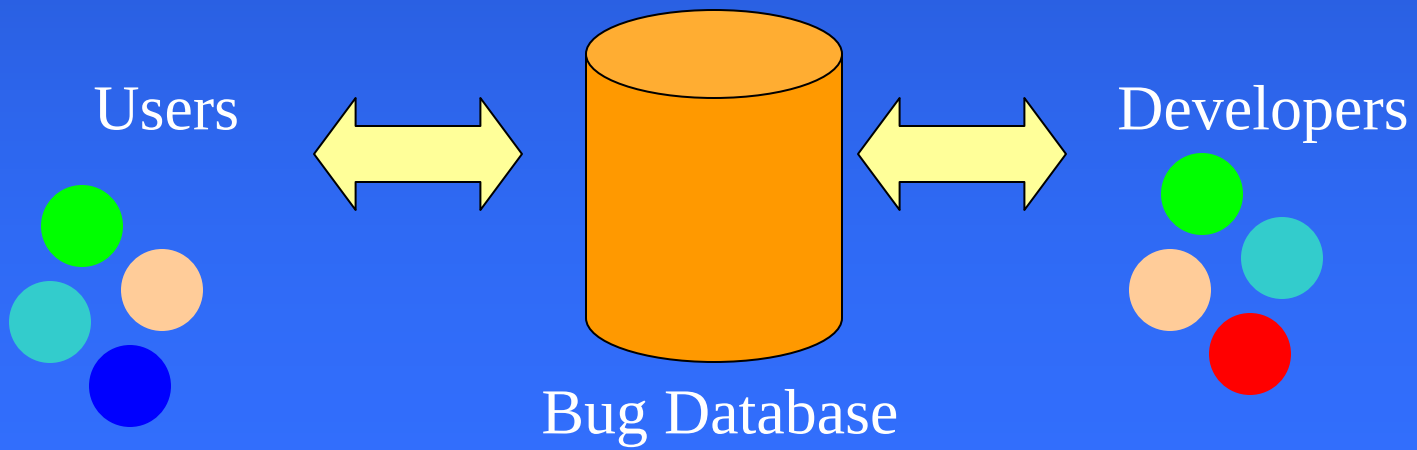
- Review historical data to choose a static bug checker to apply
  - manual inspection to gather background data
  - identify common types of bugs being fixed
  - Bug report database
  - CVS commit messages
- Mine historical data to refine the results of a static bug checker
  - automatic data gathering
  - reduce false positive warnings
  - provide ranking of warnings
  - CVS commits

Background

- Statically checking source code has been very effective
  - finds complex errors
  - test suite is not needed
- Which checkers to run?
  - there are lots of possible static checkers
  - what kinds of bugs are really out there?
- False positive warnings are a challenge for static checkers
  - can we rank the warnings better?
  - too many make a tool unusable

Bug Database

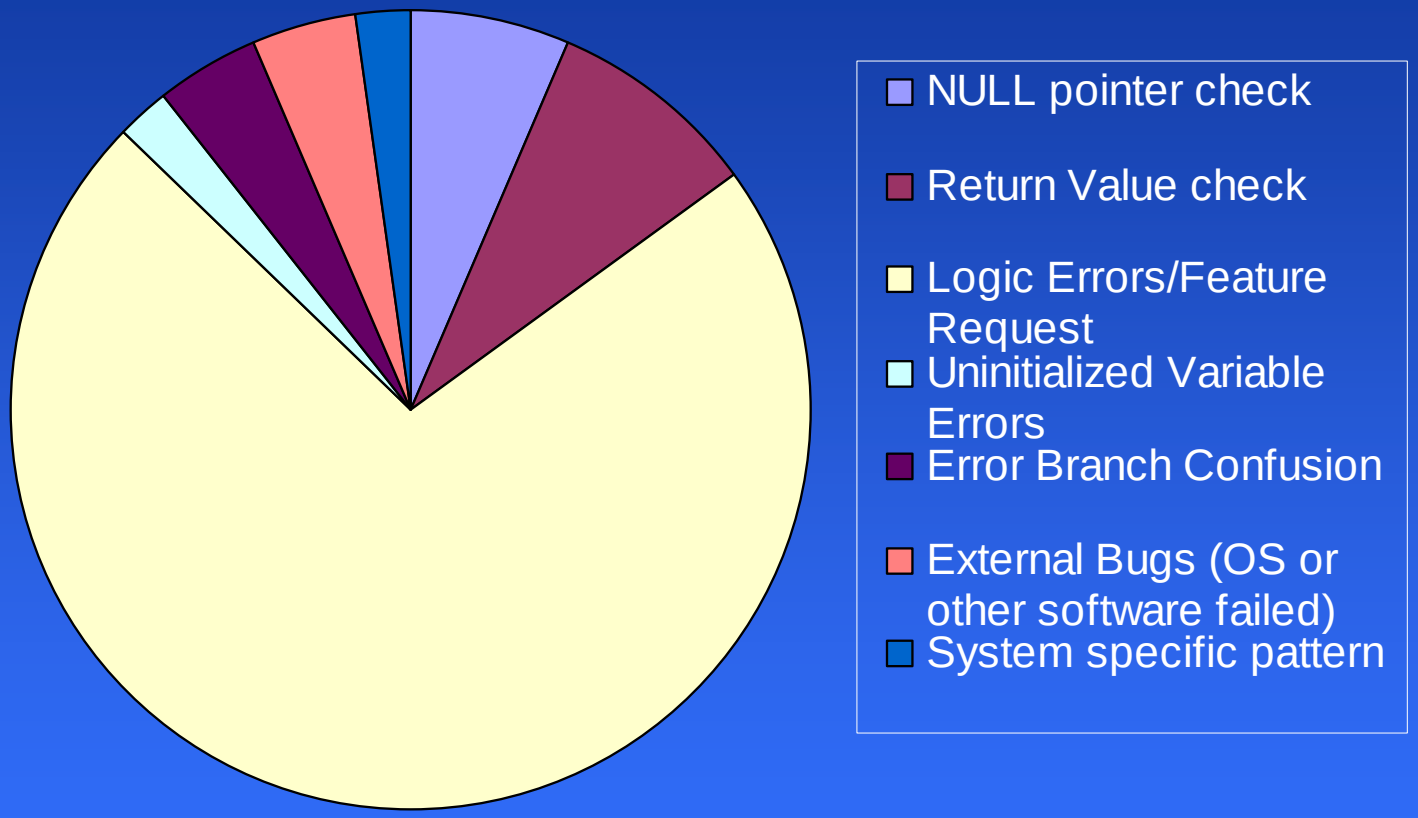
- Inspect fixed bugs
  - manual inspection to gain understanding
  - review bug discussions
  - tie fixed bug to source code change
  - classify the type of the bug
  - look for bugs that can be found statically



Bug Database: Practical Experience

- We inspected the Apache httpd bug database
  - inspected 200 bug reports marked as fixed
  - bug reports include a discussion of the problem
  - not as helpful as we expected
- Only 24% easily tied directly to a software change
  - rarely is a diff or a CVS revision noted
- Most bugs were logic errors
  - verification errors
  - system specific errors
  - also includes feature requests

Bug Database Bug Types



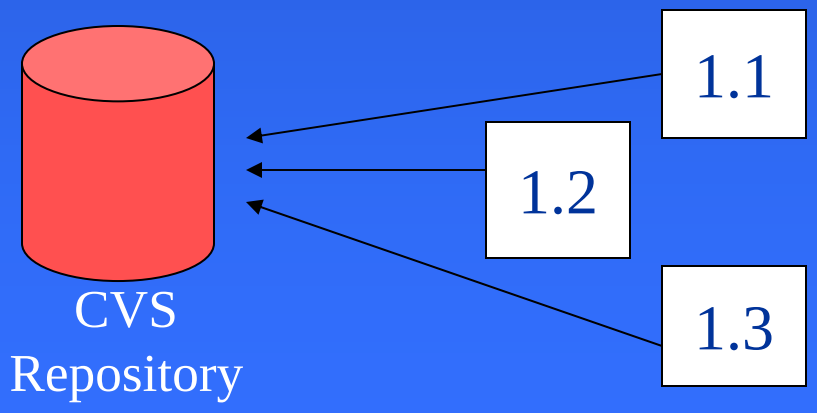
- Most classified bugs are logic errors

Bug Database: Practical Experience

- Most bug reports came from outside the project
  - 197 out of 200
  - does not capture bugs found by developers
- Most bug reports came against a release of the software, not a CVS-HEAD
  - 198 out of 200
  - does not capture bugs between releases
- What about the bugs that don't make it into the release?
  - they may be in the CVS repository…

CVS Repository

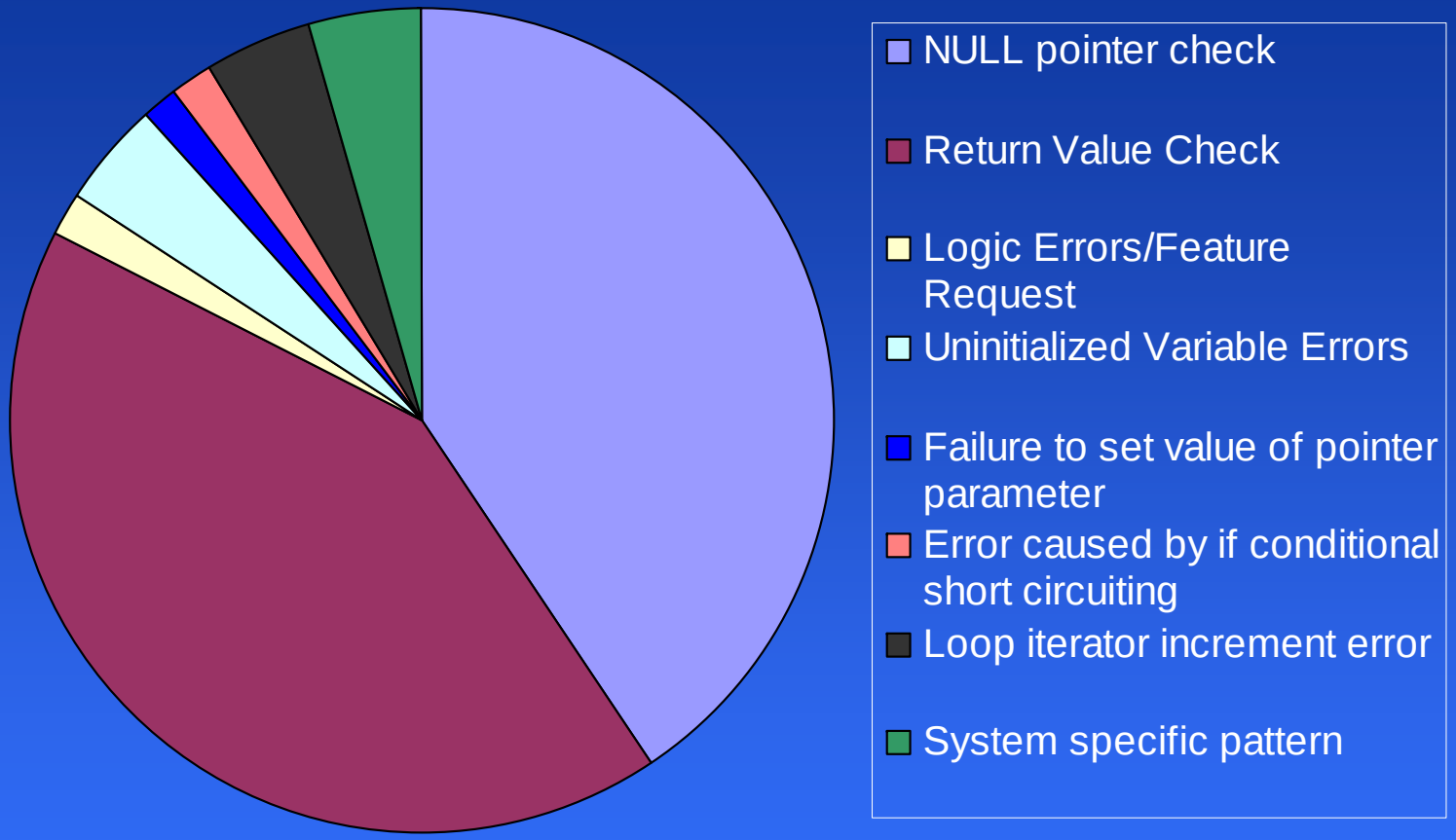
- Commits may contain useful data
  - any bug fix must show up in a commit
  - will commit messages lead us to bug fixes?
- Shows bugs fixed between releases
- Bugs caught by developers



CVS Repository: Practical Experience

- Inspected commit messages
  - manual inspection
  - looked for 'fix', 'bug' or 'crash'
  - ignored those with bug number listed
- Commit messages more useful
  - trivially tied to source code change
  - fewer logic errors
- Common errors found
  - NULL pointer check
  - failing to check the return value of a function before use

CVS Repository Bug Types



- NULL pointer bugs and return value bugs are both statically checkable

Function Return Value Check Bug

- Returning error code and valid data from a function is a common C idiom
  - the return value should be checked before being used
  - control flow decision must depend on it
  - lint checks for this error
- Error types
  - completely ignored
    - foo();
  - return value used directly as an argument
    - bar(foo());
  - others …

```
int foo(){
    …
    if( error ){
        return error_code;
    }
    ….
    return data;
}

…
value = foo();
newPosition + = value; // ???
```

Our Tool

- Static checker that looks for return value check bugs
  - based on ROSE by Dan Quinlan, et al., at Lawrence Livermore National Laboratory
- Classify each warning by category
  - ignored return value
  - return value used as argument, etc.
- Produce a ranking of the warnings
  - group warnings by called function
  - rank functions that most need their return value checked higher

Return Value Checker: Problems

- Some functions don't need their return value checked
  - no error value returned
  - could lead to many false positives
  - int printf()
- Naively flagging all unchecked return values leads to many false positives
  - over 7,000 warnings reported for the Apache httpd-2.0 source
- Must find which are most likely true bugs
  - use historical data

Which return values need checked?

- Infer from historical data
  - look for an add of a check of a return value in a CVS commit
  - implies the programmer thinks its important

```
…
value = foo();
newPosition + = value; // ???
…
```

Commit
Bug Fix

```
…
value = foo();
if( value != Error) { // Check
    newPosition + = value;
}
…
```

- Infer from current usage
  - does the return value of a function get checked in the current version of the software?
  - how often?

Ranking Warnings

- Rank warnings in two ways
- Split functions into two groups
  - functions flagged with a CVS commit
    - at least one CVS commit adds a check of the function's return value
  - functions not flagged with a CVS commit
- Within each group, rank by how often the function's return value is checked in the current software distribution

Case Study

- Apache httpd-2.0 on Linux
- Checked all CVS commits for bug fix
  - 6100 commits checked
  - 2600 commits failed to go through our tool
    - required older version of autoconf
    - compile bugs in the CVS commits
- Difficult to check an old version!
  - may rely on specific compiler release
  - may rely on specific configuration tools
  - may rely on specific libraries/header files

Case Study

- Our checker marked over 7,000 warnings
  - individual call site for non-void function where the return is not checked
- Too many too look at!
  - expect many are false positives
- Rank warnings
  - 115 functions called in the current software distribution were involved in a bug fix
    - 62 *always* had their return value checked
  - inspect CVS flagged functions
  - inspect functions with return value checked >50% of the time in the current source tree

```
value = foo(); // WARNING
newPosition + = value;
…
result = foo(); // WARNING
zoo(result);
```

Case Study: Warning Breakdown

- Inspected 490 warning (of 7000)
  - found 129 potential bugs!
- 235 warnings associated with a CVS flagged function
  - 84 of the bugs found here
  - false positive rate of 64%
- 255 warnings associated with a function that has its return value checked > 50% of the time
  - 45 of the bugs found here
  - false positive rate of 82%

Case Study: A Bug

- We investigated a warning and found it did crash httpd
  - warning near the top of the ranking
- The function builds a string from arguments that represent a filename and a pathname
  - a **`char`** array is returned and directly used as an argument to **`strcmp()`**
  - NULL return value will cause a seg fault
  - return value is NULL if the path is too long!

Analysis

- False positive rate too high!
  - overall false positive rate: 74% (1-(129/490))
  - for CVS flagged functions it's only 64%
- A false positive rate closer to 50% would be acceptable
  - the user is as likely as not to find a true error
  - cluster true errors near the top of the ranking
    - CVS flagged functions, fewer false positives
- We did cull 7,000 warnings down to 490
  - lint would have flagged only the 'ignored' warnings and not ranked them

Conclusion

- Bug databases are not useful in understanding much about low-level bugs
  - deeper analysis could match bug report to CVS commit but this can be expensive
- CVS commit messages give a better picture of low-level bugs
  - CVS commits can give useful data to help classify error reports
- Mining data from CVS commits is useful to determine properties of the code
  - which functions need return value checked

Future Work

- Run on different software project
- What other checkers can benefit from CVS data?
- Can we dynamically gather data on functions called via function pointers?
  - many of the warnings involved calls through function pointers
- What other uses for CVS data can we find for refining bug finding techniques?
  - can we use data on a semantic level about how the code changes?

University of Maryland

Bug Driven Bug Finding

- We have done a study of a bug report database and a CVS repository
  - found types of bugs that are being fixed
- Used CVS data to refine the output of a bug finder
  - static source code checker
  - reduced false positive error reports to a usable number
- Suggests looking at CVS commits is a good way to determine important properties of the code

Bug Driven Bug Finding

- Common bugs in C code are often context-less
  - either a bug or not
  - NULL pointer check
  - array bounds checking
- May need to look at properties different than common bugs
  - system specific properties
  - slowly built up over time
  - undocumented

Reducing False Positives

- Return value checkers inherently have large numbers of false positives
- We automatically ignored some common types of false positives
  - return value is tested based on other data
  - return value is directly returned by the calling function
  - return value is returned via pointer argument to function

```
int x = foo();
if( y != NULL )
    if( x = = 42 )
        bar(x);
```

```
return foo();
```

```
void foo(int *arg){
  …
  *arg = foo();
  …
}
```

Further Reducing False Positives

- Interprocedural analysis
- Many functions check their arguments before using them
  - check pointers for NULL before use
- Passing a return value to a checked argument should not be an error
  - simple matter of tracking which arguments are checked before use

```
int bar(struct data* ptr){
   if( ptr != NULL ){
         ptr–>theData = … ;
   }
   …
}
```

Return Value Checker: Error Types

- Completely ignored
  - foo();
- Return Value Used as an argument
  - bar(foo());
- Return Value dereferenced without NULL check
  - foo()->data;
- Return Value Stored but:
  - unused
  - untested
    - assignment statement