Finding Latent Code Errors via Machine Learning over Program Executions

Yuriy Brun California

Michael D. Ernst

University of Southern Massachusetts Institute of Technology

Bubble Sort

```
// Return a sorted copy of the argument
double[] bubble_sort(double[] in) {
   double[] out = array_copy(in);
   for (int x = out.length - 1; x >= 1; x--)
     for (int y = x - 1; y >= 1; y--)
     if (out[y] > out[y+1])
        swap (out[y], out[y+1]);
   return out;
}
```

Bubble Sort

Faulty (?) Code:

// Return a sorted copy of the argument double[] bubble_sort(double[] in) { double[] out = array_copy(in); for (int x = out.length - 1; x >= 1; x--) for (int y = x - 1; y >= 1; y--) if (out[y] > out[y+1]) swap (out[y], out[y+1]); return out; } Fault-revealing properties

out[0] = in[0]

```
out[1] \le in[1]
```

Bubble Sort

Faulty Code:

// Return a sorted copy of the argument double[] bubble_sort(double[] in) { double[] out = array_copy(in); for (int x = out.length - 1; x >= 1; x--) // lower bound should be 0, not 1 for (int y = x - 1; y >= 1; y--) if (out[y] > out[y+1]) swap (out[y], out[y+1]); return out; Fault-revealing properties

out[0] = in[0]

 $out[1] \leq in[1]$

Outline

- Intuition for Fault Detection
- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Outline

Intuition for Fault Detection

- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Targeted Errors

- Latent Errors
 - unknown errors
 - may be discovered later
 - no manifestation
 - not discovered by test suite

Targeted Programs

- Programs that contain latent errors
- Test inputs are easy to generate, but test outputs can be hard to compute, e.g.:
 - Complex computation programs
 - GUI programs
 - Programs without formal specification

Learning from Fixes

. . .

Program A:

```
Fixed Program A:
```

```
print (a[a.size] + "elements");
```

```
•••
```

```
print (a[a.size - 1] + "elements");
```

```
Program B:
```

```
if (store[store.length] > 0);
```

. . .

Outline

- Intuition for Fault Detection
 Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Program Description Mapping



Machine Learning Approach

- Extracts knowledge from a training set
- Creates a model that classifies new objects
- Requires a numerical description of the samples

Training a Model program with program with **Examples**: errors removed known errors program program analysis analysis $out[1] \leq in[1]$ properties properties characteristic characteristic extractor extractor (1,0,0,2)features features machine learner

model



Classifying Properties



Related Work

- Redundancy in source code
 [Xie et al. 2002]
 - find an error
 - 1.5-2 times
 improvement over
 random sampling

- Relevance:
- same goal
- we have 50 times improvement over random sampling (for C programs)

Related Work

- [Xie et al. 2002]
- Partial invariant violation [Hangal et al. 2002]
 - is there an error?

- Relevance:
- similar program analysis
- similar goal

Related Work

- [Xie et al. 2002]
- [Hangal et al. 2002]
- Clustering of function call profiles [Dickinson et al. 2001, Podgurski et al. 2003]
 - find relevant tests
 - select faulty executions

- Relevance:
- uses machine learning

Latent Error-Finding Technique

- Abstract properties
- Abstract features
- Generalizes to new properties and programs



Model

- A function:
 - {set of features} \rightarrow {fault-revealing, non-fault-revealing}
- Examples:
 - Linear combination functions
 - If-Then rules

Outline

- Intuition for Fault Detection
- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Tools Required for Fault Invariant Classifier

- Program Property Extractor
 Daikon: Dynamic analysis tool
- Property to Characteristic Vector Converter
- Machine Learning
 - Support Vector Machines (SVMfu)
- technique is equally applicable to static and dynamic analysis



Daikon: Program Property Extractor

- Daikon
 - Dynamic analysis tool
 - Reports properties that are true over program executions



- Examples:
 - myPositiveInt > 0
 - length = data.size

Characteristic Vector Extractor

- Daikon uses Java objects to represent properties
- Converter extracts all possible numeric information from those objects
 - # of variables e.g. $x > 5 \rightarrow 1 x \in array \rightarrow 2$
 - is inequality? e.g. $x > 5 \rightarrow 1 x \in array \rightarrow 0$
 - -involves an array? e.g. $x > 5 \rightarrow 0$ $x \in array \rightarrow 1$
- Total: 388 features

Support Vector Machine Model

- Predictive power
- But not explicative power
- Consists of thousands of support vectors that define a separating area of the search space

Outline

- Intuition for Fault Detection
- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Subject Programs

- 12 Programs
 - C and Java programs
 - Largest: 9500 lines
 - 373 errors (132 seeded, 241 real)
 - with corrected versions
 - Authors (at least 132):
 - Students
 - Industry
 - Researchers

Accuracy Experiment

- Goal:
 - Test if machine learning can extrapolate knowledge from some programs to others
- Train on errors from all but one program
- Classify properties for each version of that one program
- Compare to expected results

Measurements and Definitions

- Fault-revealing property:
 - property of an erroneous program but not of that program with the error corrected
 - indicative of an error
- Brevity:
 - average number of properties one must examine to find a fault-revealing property
 - best possible brevity is 1

Accuracy Experiment Results

- C programs (single-error)
 - brevity = 2.2
 - improvement = 49.6 times
- Java programs (mostly multiple-error)
 - brevity = 1.7
 - improvement = 4.8 times

Outline

- Intuition for Fault Detection
- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment
- Conclusion

Fault Invariant Classifier Usability Study

- Would properties identified by the fault invariant classifier lead a programmer to errors in code?
- Preliminary experimentation:
 - 1 programmer's evaluation
 - 2 programs (41 errors, 410 properties)

Usability Study Results

- Replace (32 errors)
 - 68% of properties reported fault-revealing would lead a programmer to the error
- Schedule (9 errors)
 - 58% of properties reported fault-revealing would lead a programmer to the error

The majority of the reported properties were effective in indicating errors

Outline

- Intuition for Fault Detection
- Latent Error Finding Technique
- Fault Invariant Classifier Implementation
- Accuracy Experiment
- Usability Experiment

➢ Conclusion

Conclusion

- Designed a technique for finding latent errors
- Implemented a fully automated Fault Invariant Classifier
- Fault Invariant Classifier revealed fault-revealing properties with brevity around 2
- Most of the fault-revealing properties are expected to lead a programmer to the error
- Overall, examining 3 properties is expected to lead a programmer to the error in our tests

Backup Slides

- Works Cited
- Explicative Machine Learning Model

Works Cited

[Dickinson et al. 2001] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by clus execution profiles. In ICSE, pages 339–348, May 2001.

- [Hangal at al. 2002] S. Hangal and M. S. Lam. Tracking down software bugs using autor detection. In ICSE, pages 291–301, May 2002.
- [Podgurski at al. 2003] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, ar Automated support for classifying software failure reports. In ICSE, pages 465–475, May 2003.
- [Xie et al. 2002] Y. Xie and D. Engler. Using redundancies to find errors. In FSE, pages Nov. 2002.

Explicative Machine Learning Model

- C5.0 decision tree machine learner
- Examples:
- Based on large number of samples and neither an equality nor a linear relationship of three variables → likely fault-revealing
- Sequences contains no duplicates or always contains an element → likely fault-revealing

- No field accesses \rightarrow even more likely fault-revealing