Testing and Analysis of Next Generation Software

Mary Jean Harrold
College of Computing
Georgia Tech

harrold@cc.gatech.edu

Joint work with T. Apiwattanapong, J. Bowring, J. Jones, D. Liang, R. Lipton, A. Orso, J. Rehg, and J. Stasko
Computing (so far)

- Big Iron (‘40s/’50s)
- Mainframe (’60s/’70s)
- Workstations (’70s/’80s)
- Individual PCs (’80s/’90s)
- Internet (’90s)
- Implicit, ubiquitous, everyday computing (21st century)
Some Features/Challenges

Features
- **Scope**
  - embedded in everyday devices
  - many processors/person
- **Connectivity**
  - mobile, interconnected
  - coupled to data sources
  - implicit interactions
- **Computational resources**
  - powerful
  - embedded intelligence

Lucy Dunne
Cornell University
Smart Jacket
Some Features/Challenges

Features
• **Scope**
  • embedded in everyday devices
  • many processors/person
• **Connectivity**
  • mobile, interconnected
  • coupled to data sources
  • implicit interactions
• **Computational resources**
  • powerful
  • embedded intelligence

Challenges
• many environments in which to run
• short development and evolution cycles
• requirement for high quality
• dynamic integration of components
• increased complexity of components, interactions, and computational resources
Before deployment
- test-driven development
- modular testing of components
- formal methods
The Gamma Project

Software
Field data
Software
Field data
Software
Field data
Software
Field data
Software
Field data

Internet

Field-data

Analysis
Outline

- Gamma project
  - Overview, problems
    [Orso, Liang, Harrold, Lipton; ISSTA 2002]
  - Summary of current projects
- Visualization of field data
- Related work
- Summary, Challenges
- Questions
1. Effectively use field data?

2. Efficiently monitor, collect field data?

3. Continuously update deployed software?

Field-data → Analysis
1. Effective use of field data

- Measurement of coverage
  [Bowring, Orso, Harrold, PASTE 02]
- Impact analysis, regression testing
  [Orso, Apiwattanapong, Harrold, FSE 04]
- Classify/recognize software behavior
  [Bowring, Rehg, Harrold, TR 03]
- Visualization of field data
  [Jones, Harrold, Stasko, ICSE 02]
  [Orso, Jones, Harrold, SoftVis 03]
2. Efficient monitoring/collecting of field data

- Software tomography
  [Bowring, Orso, Harrold, PASTE 02]
  [Apiwattanapong, Harrold, PASTE 02]
- Capture/replay of users’ executions
  [Orso, Kennedy, in preparation]

3. Continuous update of deployed software

- Dynamic update of running software
  [Orso, Rao, Harrold, ICSM 02]
1. **Effective use of field data**
   - Measurement of coverage
   - Impact analysis, regression testing
     → Classify/recognize software behavior
   - Visualization of field data

2. **Efficient monitoring/collecting of field data**
   - Software tomography
   - Capture/replay of users’ executions

3. **Continuous update of deployed software**
   - Dynamic update of running software
Classify/Recognize Behavior

Problem

• Behavior classification, recognition difficult, expensive
• Recognize behavior without input/output needed

For classifying and recognizing behavior

• Behaviors are the results of executing program

Approach

- Prepare Training Instances
  - Training set = Branch profiles w/behavior labels

- Train Classifier
  - Markov models
  - active learning
Empirical Studies

• **Research questions**
  1. What is classification rate and classifier precision of trained classifier on different-size subsets of test suite?
  2. How does active learning improve training?

• **Subject program: Space**
  - 8000 lines of executable code
  - Test suite contains 13,500 tests
  - 15 versions

• **Experimental Setup**
  1. For each version (repeated 10 times)
    - trained classifier on (random) subsets 100-350
    - evaluated classifier on rest of test suite
  2. Compared batch, active learning
Results

Classification Rate

<table>
<thead>
<tr>
<th>Training set size</th>
<th># of classifiers</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>150</td>
<td>0.976</td>
</tr>
<tr>
<td>350</td>
<td>150</td>
<td>0.976</td>
</tr>
</tbody>
</table>

Classifier Precision (batch)

Classifier Precision

Training Set Size

100  150  200  250  300  350
Results

Classifier Precision

Training Set Size

Batch learning

Active learning
Outline

• Gamma project
  • Overview, problems
  • Summary of current projects
• Visualization of field data
• Related work
• Summary, Challenges
• Questions
Problem

• Huge amount of execution data difficult to understand, inspect manually
• Developers need help in finding faults

Visualize field data for fault localization

• Visualization for fault localization
  [Jones, Harrold, Stasko; ICSE 02]
• Visualization of field data (Gammatella)
  [Orso, Jones, Harrold; SoftVis 03]
Consider two statements

\[ m = x \]  \[ w = y \]

More suspicious of being faulty
**Visualization for Fault Localization**

- **Uses**
  - Pass/fail results of executing test cases (actual or inferred)
  - Coverage/profiles provided by those test cases (statement, branch, def-use pairs, paths, etc.)
  - Source code of program

- **Computes**
  - Likelihood that a statement is faulty
  - Summarizes pass/fail status of test cases that covered the statements

- **Maps to visualization (Tarantula)**
  - Using two variables
For statement $s$:

**Hue** summarizes pass/fail results of test cases that executed $s$

**Brightness** presents the “confidence” of the hue assigned to $s$
mid() {
    int x, y, z, m;
1:  read("Enter 3 numbers:", x, y, z);
2:  m = z;
3:  if (y<z)
4:      if (x<y)
5:          m = y;
6:      else if (x<z)
7:          m = y;
8:  else
9:    if (x>y)
10:       m = y;
11:   else if (x>z)
12:       m = x;
13:  print("Middle number is:", m);
}

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>3,3,5</th>
<th>1,2,3</th>
<th>3,2,1</th>
<th>5,5,5</th>
<th>5,3,4</th>
<th>2,1,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass Status</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>F</td>
</tr>
</tbody>
</table>
```c
mid() {
    int x, y, z, m;
    read("Enter 3 numbers:", x, y, z);
    m = z;
    if (y < z)
        if (x < y)
            m = y;
        else if (x < z)
            m = y;
        else
            if (x > y)
                m = y;
            else if (x > z)
                m = x;
    print("Middle number is:", m);
}
```
SeeSoft view

- each pixel represents a character in the source

```c
mid() {
    int x,y,z,m;
    read(“Enter 3 numbers:”,x,y,z);
    m = z;
    if (y<z)
        if (x<y)
            m = y;
        else if (x<z)
            m = y;
        else
            if (x>y)
                m = y;
            else if (x>z)
                m = x;
    print(“Middle number is:”, m);
}
```
SeeSoft view

- each pixel represents a character in the source
System-level View

TreeMap view

- each node
  - represents a file
  - is divided into blocks representing color of statements
• **Research questions**
  1. How red are the faulty statements?
  2. How red are the non-faulty statements?

• **Subject program: Space**
  • 8000 lines of executable code
  • 1000 coverage-based test suites of size 156-4700 test cases
  • 20 faulty versions (10 shown here)

• **Experimental Setup**
  • Computed the color for each statement, each test suite, each version
  • For each version, computed the color distribution of faulty, non-faulty statements
Results

Redness of Faulty Statement

Color distribution of faulty statements

Faulty Versions

Redness of Non-faulty Statement

Color distribution of non-faulty statements

Faulty Versions

Redness of Faulty Statement

Color distribution of faulty statements

Faulty Versions

Redness of Non-faulty Statement

Color distribution of non-faulty statements

Faulty Versions
Gammatella

Software Developer

Tarantula

queries

Database

data

InsECT Instrumenter

instrumented program

Data Collection Daemon

execution data

In the Field

At Developers’ Site

User 1

User 2

User N

program

visualization/interaction

program
• **Subject program: JABA**
  • Java Architecture for Bytecode Analysis
  • 60,000 LOC, 550 classes, 2,800 Methods

• **Data**
  • field data: > 2000 executions (15 users, 12 weeks)
Results

- **Use of software**
  - identified *unused features* of JABA
  - redesigned into a separate plug-in module

- **Error**
  - identified *specific combination* of platform and JDK predictably causes problems
Results

Public display monitors deployed software
Outline

• Gamma project
  • Overview
  • Summary of current projects
• Visualization of field data
• Related work
• Summary, Challenges
• Questions
Related Work

**Gamma Project**
- Perpetual/Residual testing (Clarke, Osterweil, Richardson, Young)
- Expectation-Driven Event Monitoring (EDEM) (Hilbert, Redmiles, Taylor)
- Remote Monitoring/Measurement of Deployed Software (Notkin, Porter, Schmidt)
- Bug Isolation (Liblit, Aiken, et al.)

**Visualization**
- Seesoft, SeeSys (Eick, Sumner, Baker)
- Treemap (Schneiderman)
- Bloom, ALMOST, … (Reiss, Renieris)
- Jinsight (DePauw et al.)

**Behavior Modeling, Instrumentation, Profiling**
- Too numerous to list
Outline

• Gamma project
  • Overview
  • Summary of current projects
• Visualization of field data
• Related work
  • Summary, Challenges
• Questions
Summary

- Motivated need for new kind of testing for next generation software
- Described new kind of testing---Gamma testing
  - addresses challenges of testing next generation software: many environments, short development cycles, high-quality requirements, dynamic integration, and complexity
  - a collaborative effort between developer and users
- Presented problems that must be solved
- Described several Gamma projects
(Some) Challenges

• **Effective use of field data**
  • very preliminary results so far
  • effective techniques will be mix of
    • in-house analysis (static and dynamic) and
    • analysis of field data (dynamic, aggregate)

• **User participation in analysis of field data**
  • filtering before sending to developer
  • initiating new analyses in response to events at their sites or due to interactions with other users
  • creating their own test suites to be run locally

• **Privacy of users**
  • techniques that protect users data
  • user-specific analysis/testing for privacy