SmartUnit: Empirical Evaluations for Automated Unit Testing of Embedded Software in Industry

Chengyu Zhang, Yichen Yan, Hanru Zhou, Yinbo Yao, Ke Wu, Ting Su, Weikai Miao, Geguang Pu

East China Normal University, China
Nanyang Technological University, Singapore
National Trusted Embedded Software Engineering Technology Research Center, China

2018.5.30
Outline

- Background
- Approach
- Implementation
- Evaluation
- Conclusion
Motivation

RTCA DO-178B/C

IEC 61508

ISO26262
Unit Testing
Code Coverage Criterion

RTCA DO-178B/C

Level A
Level B
Level C

MC/DC
Decision Coverage
Statement Coverage
Condition Coverage

If ( A || B ) && C ) {
    /*Instructions*/
}
else{
    /*Instructions*/
}

A = True  B = True  C = True
A = False  B = False  C = False
If ( A || B ) && C ) {
    /*Instructions*/
}
else{
    /*Instructions*/
}

A = True B = True C = True
=> True

A = False B = False C = False
=> False
Modified Condition/Decision Coverage (MC/DC)

If (A || B) && C) {
    /*Instructions*/
}
else{
    /*Instructions*/
}

A = False B = True C = True
A = False B = True C = False
A = False B = False C = True
A = True B = False C = True
A = True B = False C = True
Modified Condition/Decision Coverage (MC/DC)

If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}

A = False B = True C = True
A = False B = True C = False
A = False B = False C = True => False
A = True B = False C = True => True
Modified Condition/Decision Coverage (MC/DC)

If ( A || B ) && C ) {
    /*Instructions*/
}
else{
    /*Instructions*/
}

A = False B = True C = True => True
A = False B = True C = False => False
A = False B = False C = True
A = True B = False C = True
Modified Condition/Decision Coverage (MC/DC)

If (A || B) && C) {
    /*Instructions*/
}
else{
    /*Instructions*/
}

A = False B = True C = True => True
A = False B = True C = False => False
A = True B = False C = True
Dynamic Symbolic Execution (Concolic Testing)

```c
int checkSign (int x){
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}
```
Approach

Dynamic Symbolic Execution
(Concolic Testing)

```c
int checkSign (int x){
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}
```

1. Random generated input : 11
2. $x > 0$ : T
3. return 1

Conjunction constraint : $x \leq 0$
New input : -7
Dynamic Symbolic Execution (Concolic Testing)

```c
int checkSign (int x){
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}
```

1. input `x := -7`

2. `x > 0` (T)

3. `return 1`

4. `x == 0` (T)

5. `return 0`

6. `else`

7. `return -1`

Conjunction constraint : `x <= 0 ^ x== 0`

New input : -7

New input : 0

7. `return -1`
Dynamic Symbolic Execution (Concolic Testing)

```c
int checkSign (int x){
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}
```

Conjunction constraint : none

New input : 0

1. input x := 0
2. x > 0
   T
   3. return 1
   F
5. return 0
7. return -1

4. x == 0
   T
6. else
   return -1;
   F
```
Dynamic Symbolic Execution (Concolic Testing)

```c
int checkSign(int x)
{
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}
```

Test suite:
{11, -7, 0}
Cloud-based Platform
Cloud-based Platform Workflow

Implementation

Source Code

Source Code with Stub

SmartUnit Core

Test Suite

Report

SmartUnit Features

Start

Homepage | Projects | Help

Web UI

Server

Database

File System

Worker
SmartUnit Execution Core

C Source code → Paser → CFG

Execution Module:
- Memory Model
- Executor
- Searcher

Constraint Solver

Test Suite
Features

Automatically

- generate test suite.
  (For LDRA Testbed, Tessy, etc.)

- generate test report.
  (Statement, branch, MC/DC coverage)

- insert stubs for function calls & variables
  (Global variables, function inputs, etc.)
Research Questions

RQ1: How about the performance of SmartUnit on both commercial embedded software and open-source database software?

RQ2: What factors make dynamic symbolic execution get low coverage?

RQ3: Can SmartUnit find the potential runtime exceptions in real-world software?

RQ4: What are the differences in terms of time, cost and quality between automatically generated test cases and manually written test cases?
### RQ1:

**How about the performance of SmartUnit?**

<table>
<thead>
<tr>
<th>Subjects</th>
<th># Files</th>
<th># Functions</th>
<th># LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Software</td>
<td>8</td>
<td>54</td>
<td>3,769</td>
</tr>
<tr>
<td>Automotive Software</td>
<td>4</td>
<td>330</td>
<td>31,760</td>
</tr>
<tr>
<td>Subway Signal Software</td>
<td>108</td>
<td>874</td>
<td>37,506</td>
</tr>
<tr>
<td>SQLite</td>
<td>2</td>
<td>2,046</td>
<td>126,691</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>906</td>
<td>6,105</td>
<td>279,809</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,028</strong></td>
<td><strong>9,409</strong></td>
<td><strong>479,535</strong></td>
</tr>
</tbody>
</table>
RQ1: How about the performance of SmartUnit?

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Statement Coverage*</th>
<th>Branch Coverage*</th>
<th>MC/DC Coverage*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A  0-50%  50-99% 100%</td>
<td>N/A  0-50%  50-99% 100%</td>
<td>N/A  0-50%  50-99% 100%</td>
</tr>
<tr>
<td>Aerospace Software</td>
<td>1  3  10  41</td>
<td>1  5  8  41</td>
<td>45  2  -  8</td>
</tr>
<tr>
<td>Automotive Software</td>
<td>1  3  11  315</td>
<td>1  6  8  315</td>
<td>274  5  1  50</td>
</tr>
<tr>
<td>Subway Signal Software</td>
<td>6  1  50  817</td>
<td>6  2  55  811</td>
<td>558  11  11  294</td>
</tr>
<tr>
<td>SQLite</td>
<td>86  86  206  1668</td>
<td>86  119  205  1636</td>
<td>1426  118  149  351</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>687  732  1044  3642</td>
<td>687  1102  804  3512</td>
<td>4083  1308  249  465</td>
</tr>
</tbody>
</table>

* : Statistic with the number of functions for corresponding coverage range
RQ2: What factors make dynamic symbolic execution get low coverage?

- Environment variables and Environment functions
- Complex operations
- Limitation of SMT solver
RQ3: Can SmartUnit find the potential runtime exceptions in real-world software?

- Array index out of bounds
- Fixed memory address
- Divided by zero
RQ3:
Can SmartUnit find the potential runtime exceptions in real-world software?

• Array index out of bounds

```c
static char *cmdline_option_value(int argc, char **argv, int i) {
    if (i == argc) {
        utf8_printf(stderr, "Error: missing argument to %s\n", argv[0], argv[argc - 1]);
        exit(1);
    }
    return argv[i];
}
```
RQ3:
Can SmartUnit find the potential runtime exceptions in real-world software?

• Fixed memory address

(*0X00000052) or (*(symbolic_variable+12)
RQ3:
Can SmartUnit find the potential runtime exceptions in real-world software?

- Divided by zero

```c
static void getLocalPayload(int nUsable, u8 flags, int nTotal, int *pnLocal){
    int nLocal, nMinLocal, nMaxLocal;
    if( flags==0x0D ){
        nMinLocal = (nUsable - 12) * 32 / 255 - 23;
        nMaxLocal = nUsable - 35;
    }else{
        nMinLocal = (nUsable - 12) * 64 / 255 - 23;
        nMaxLocal = (nUsable - 12) * 64 / 255 - 23;
    }
    nLocal = nMinLocal + (nTotal - nMinLocal) % (nUsable - 4);
}
```
**RQ4:**

What are the differences between automatically generated and manually written test cases?

<table>
<thead>
<tr>
<th>Subjects</th>
<th># Functions</th>
<th>Time (s)</th>
<th>Average (s/func)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Software</td>
<td>54</td>
<td>318</td>
<td>6</td>
</tr>
<tr>
<td>Automotive Software</td>
<td>330</td>
<td>329</td>
<td>1</td>
</tr>
<tr>
<td>Subway Signal Software</td>
<td>874</td>
<td>2,476</td>
<td>3</td>
</tr>
<tr>
<td>SQLite</td>
<td>2,046</td>
<td>13,482</td>
<td>6</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>6,105</td>
<td>18,857</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9,409</strong></td>
<td><strong>35,462</strong></td>
<td><strong>3.77</strong></td>
</tr>
</tbody>
</table>

A trained test engineer can only produce test case for 5-8 functions per day.
Conclusion

SmartUnit

Dynamic symbolic execution
(High coverage unit testing)

Potential runtime exceptions
(Out of bounds, divided by zero, etc.)

Industry application
(Insert stubs, test report, test suite)

China Academy of Space Technology