Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?

A survey of static analysis tool usability

Ágoston Sipos
2018. 12. 05.

Outline

- What is static analysis?
- A case study about why developers use or don‘t use them
- Some own experience with tools

Introduction

- Static analysis: automated examination of source code *without running it*
- Purpose: finding potential errors
- Simplest: basic compiler warnings

```
int i, j;
j = i+1;
```

```
warning: ‘i’ is used uninitialized in this function [-Wuninitialized]
```

- More advanced:
  - Model checking
  - Data-flow analysis
  - Symbolic execution

Problems

- False positive
  - Potential error found where it is not
  - For example, too long string would cause problem, but length of string is limited on web form
- False negative
  - Not finding a real error
  - Path leading to error is too complex to be analyzed
- It depends which is more problematic
  - Saying is the industry hates false positives, the academia hates false negatives

Example – symbolic execution

```
int a, b;
scanf("%d", &a);
if(a == 2)
    b = 6 / (a - 2);
```

Potential error

```
int a, b;
scanf("%d", &a);
if(a == 3)
    b = 6 / (a - 2);
```

No potential error

- Unknown (input) values are considered symbolic variables
- At the division, divisor is a function of unknown value
- Can be computed, what input will cause division-by-zero
  - Loops, function calls make it hard to compute

Case study

- By Johnson et al., NC State
- Based on interviewing developers about their perceptions of tools
- Research questions:
  - Why do or don‘t developers use static analyzis tools
  - How do current tools fit into a developer‘s workflow
  - What improvements do they want to be made to them

Case study

- Interviews consisted of 3 parts:
  - Short questions about static analysis tools (Have you ever used…?, What is your opinion...?)
  - Interactive usage of tools: Participants worked on code while explaining what they are doing out loud
  - Asking for design suggestions for static analysis tools

Case study

- Answers where grouped by the authors
  - Tool output (e.g. false positives, readability)
  - Supporting teamwork
  - User input & customizability
  - Result understandability (why something is an error)
  - Workflow (e.g. tool integration)
  - Tool design
- Examples in the article
  - *„(…) so now I wanna know why raising a string exception is bad. Like what should I be doing instead? Since it thinks it's a problem. And so none of these really help me.“*

Case study - answers

RQ1: Reasons for Use and Underuse

- Automation is well-liked
- Poorly presented output
  - More user-friendly output could counterweight false positivies, according to answerers
- Lack of support for collaboration
  - E.g. sharing configuration, discussing bugs
- Hard configurability
  - Suppressing certain types of warnings, turning on and off checkers like in a menu

Case study - answers

- Non-useful messages
  - When not understanding an error they tend to ignore it
  - Lack of „quick fixes“

Case study - answers

RQ2: Workflow integration

- Stop-and-run tools are inconvenient
- Running in background is preferred
  - IDE integration
    - Some are weel integrated
    - Others say they are inconvenient
  - Compiler integration
- Time taken away from development
- Fixing bugs immediately

Case study - answers

RQ3: Tool design

- Quick fixes
  - Preview, then apply/reject
- Fast feedback (without disruption)
- Making judgments about each warning
- Miscellaneous ideas
  - Diagrams, „heat map“

Case study - answers

RQ3: Tool design

- Quick fixes
  - Preview, then apply/reject
- Fast feedback (without disruption)
- Making judgments about each warning
- Miscellaneous ideas
  - Diagrams, „heat map“

Case study - implications

- Developers would like to use static analysis tools
- But they lose their motivation on technical issues
- Tools could be improved with:
  - Teamwork support
  - Workflow integration
  - Better explanation of errors + automatic fixes
  - Easier configuration

Clang static analyzer usage

- Tried to run analyzer on own code
- Usage:

```
clang++ -Xclang -analyze -Xclang -analyzer-
checker=core,cplusplus,deadcode,unix -Xclang -analyzer-
output=text -c main.cpp
```

  - Not very intuitive (easy to miss something, causing no output or errors)
- Mostly no warnings :) or just trivial ones
- Some example warnings on following slides

Static analyzer warnings (trivial)

- warning: Value stored to 't' during its initialization is never read [clang-analyzer-deadcode.DeadStores]

```
        unsigned t = SDL_GetTicks();
```

- warning: illegal character encoding in string literal

```
      [-Winvalid-source-encoding]

  ...std::cout << "Hiba a shader program
l<E9>trehoz<E1>sakor" << std::endl;
```

- warning: '/*' within block comment

```
      [-Wcomment]

    /*double   xDl = -1000.;
```

Static analyzer warnings

- warning: Memory allocated by 'new[]' should be deallocated by 'delete[]', not 'delete' [clang-analyzer-unix.MismatchedDeallocator]

```
                delete aSzoveg;
                ^
```

- note: Taking true branch

```
          if (GL_FALSE == result)
          ^
```

- note: Memory is allocated

```
                char* aSzoveg = new char[ProgramErrorMessage.size()];
                                        ^
```

- note: Memory allocated by 'new[]' should be deallocated by 'delete[]', not 'delete'

```
                delete aSzoveg;
                ^
```

Static analyzer warnings

- error: non-constant-expression cannot be narrowed from type 'int' to 'bool' in initializer list [clang-diagnostic-c++11-narrowing]

```
        Border w = { a.x >= b.x ? a.x : b.x, a.y >= b.y ? a.y : b.y, a.x == b.x ? 0 : 1 };
                     ^
                     static_cast<bool>()
```

- note: insert an explicit cast to silence this issue

```
        Border w = { a.x >= b.x ? a.x : b.x, a.y >= b.y ? a.y : b.y, a.x == b.x ? 0 : 1 };
                     ^
```

Static analyzer warnings

- warning: Call to 'malloc' has an allocation size of 0 bytes [clang-analyzer-unix.API]

```
    input.pointattributelist = (REAL *) malloc(input.numberofpoints *
                                               ^
```

- note: Call to 'malloc' has an allocation size of 0 bytes

```
    input.pointattributelist = (REAL *) malloc(input.numberofpoints *
                                               ^
```

- warning: the computation of the size of the memory allocation may overflow [clang-analyzer-alpha.security.MallocOverflow]

```
    input.pointmarkerlist = (int *) malloc(input.numberofpoints *
sizeof(int));
                                                                 ^
```

- note: the computation of the size of the memory allocation may overflow

```
    input.pointmarkerlist = (int *) malloc(input.numberofpoints *
sizeof(int));
                                                                 ^
```