Static Analysis

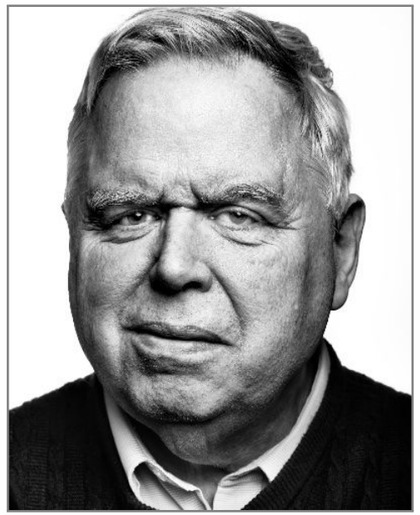
Yegor Bugayenko

Lecture #23 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.

1. Analyzing a program before it starts running may help detect its defects earlier.



Steven Johnson

"**Lint** is a command which examines C source programs, detecting a number of bugs and obscurities. It enforces the type rules of C more strictly than the C compilers. It may also be used to enforce a number of portability restrictions involved in moving programs between different machines and/or operating systems. Another option detects a number of wasteful, or error prone, constructions which nevertheless are, strictly speaking, legal."

— Stephen C. Johnson. Lint, a C Program Checker, 1977

"This is dryer lint (BOPC), which is scraped out of a clothes dryer filter after it has dried a few loads. The idea of the Lint tool is to get this sort of stuff out of your code by being very pedantic about warnings and advice on possible bad code constructions." — Quora

Some Types of Bugs to Be Found by Static Analysis

Unreachable Code:

```
int a = 10;
if (a > 20) {
  a = a + 1;
}
```

Uninitialized Variable:

```
int x;
int y = x + 42;
print(y);
```

Division by Zero:

```
int f(int x) {
  return 42 / x;
}
```

Integer Overflow:

```
var x: u8 = 142;
x = x * 2;
```

Endless Loop:

```
int x = 5;
int y = 0;
while (x > 0) {
  y = y + x; }
```

Buffer Overflow:

```
#include <stdio.h>
char buf[16];
fgets(buf, 1024, stdin);
```

Inter-procedural Analysis

Unused Global Var:

```
1 int x;
2 int foo() {
3   return 42;
4 }
5
6 int bar(int x) {
7   return x + 1;
8 }
```

Endless Recursion:

```
1 int foo(int n) {
2   return bar(n - 1);
3 }
4
5 int bar(int n) {
6   return foo(n + 1);
7 }
```

Pointer Dereferencing:

```
1 int foo() {
2   return *bar();
3 }
4
5 int* bar() {
6   return 0;
7 }
```

Violations, Smells, Bugs

Style Violation:

```
1 int f
2   (int x)
3 {
4     return 42/x;
5 }
```

Line 2: Indentation
Line 3: Curled bracket
Line 4: Indentation

Code Smell:

```
1 int f(int x) {
2   return 42.0 / x;
3 }
```

Line 2: Implicit type cast from float to int

Bug:

```
1 int f(int x) {
2   return 42 / x;
3 }
```

Line 2: Division by zero

2. Static analyzers can fail in two ways: by falsely reporting bugs or by missing them.

Brian Chess

> “Beware of any tool that says something like, ‘zero defects found, your program is, rather, now secure.’ The appropriate output is, ‘sorry, couldn’t find any more bugs.’”

— Brian Chess and Gary McGraw. Static Analysis for Security. *IEEE Security & Privacy*, 2(6):76–79, 2004. doi:10.1109/msp.2004.111

False Negative vs. False Positive

```
1 int f(int x) {
2   return 42 / x;
3 }
```

True Positive (TP):
“Division by zero”

False Positive (FP):
“Integer overflow”

True Negative (TN):
“No buffer overflow”

False Negative (FN):
“No errors at all”

Precision & Recall

Precision is the fraction of relevant instances among the retrieved instances (100% precision means *soundness*). *Recall* is the fraction of relevant instances that were retrieved (100% recall means *completeness*).

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{F1} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Sunghun Kim

"About 90% of warnings remain in the program or are removed during non-fix changes — likely false positive warnings."

— Sunghun Kim and Michael D. Ernst. Which Warnings Should I Fix First? In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference*, pages 45–54, 2007. doi:10.1145/1287624.1287633

Brittany Johnson

"Our results confirmed that false positives and developer overload play a part in developers' dissatisfaction with current static analysis tools."

— Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE, 2013. doi:10.1109/ICSE.2013.6606613

Benjamin Livshits

"We are not aware of a single realistic whole-program analysis tool that does not purposely make unsound choices... Soundness is not even necessary for most modern analysis applications, however, as many clients can tolerate unsoundness."

— Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. In Defense of Soundiness: A Manifesto. *Communications of the ACM*, 58(2):44–46, 2015. doi:10.1145/2644805

Steven Arzt

"In our experiments on DroidBench examples, TASMAN reduces the number of false positives by about 80% without pruning any true positives."

— Steven Arzt, Siegfried Rasthofer, Robert Hahn, and Eric Bodden. Using Targeted Symbolic Execution for Reducing False-Positives in Dataflow Analysis. In *Proceedings of the 4th International Workshop on State of the Art in Program Analysis*, pages 1–6, 2015. doi:10.1145/2771284.2771285

Nachiappan Nagappan

"Our results show that the static analysis defect density is correlated at statistically significant levels to the pre-release defect density determined by various testing activities. Further, the static analysis defect density can be used to predict the pre-release defect density with a high degree of sensitivity."

— Nachiappan Nagappan and Thomas Ball. Static Analysis Tools as Early Indicators of Pre-Release Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 580–586, 2005. doi:10.1145/1062455.1062558

3. Existing open-source static analyzers are less powerful than commercial ones.

My Favorite Static Analyzers

- Java: SpotBugs and PMD
- C++: Clang-Tidy
- Rust: clippy

There are many more of them:
`https://github.com/analysis-tools-dev/static-analysis`

Some Static Analysis Mechanisms

- Data Flow Analysis
- Symbolic Execution
- Model Checking
- Taint Analysis

You may want to watch my “Practical Program Analysis” course.

For some tools, you have to pay:

- Coverity by Synopsys (US)
- Klockwork by Perforce (US)
- Fortify by Micro Focus (UK)
- Checkmarx (US)
- Veracode (US)
- Snyk (US)
- PVS-Studio (Russia)

Usually, up to $3,000 per developer per year.

SARIF

```
{
  "results": [
    {
      "ruleId": "CA2101",
      "message": {
        "text": "Variable '{0}' is uninitialized.",
        "arguments": [ "pBuffer" ]
      }
    }
  ]
}
```

"This document defines a standard format for the output of static analysis tools."

Source: OASIS. Static Analysis Results Interchange Format (SARIF) Version 2.1.0 Plus Errata 01. https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html, 2023. [Online; accessed 08-03-2024]

4. Some bugs are significant enough to be assigned unique identifiers, such as CVEs (Common Vulnerabilities and Exposures).

CVE Databases:

- CVE.org
- National Vulnerability Database (NVD)
- OSV.dev
- CVEDetails.com

5. How about asking static analyzers to detect good code instead of bad code?

Florian Obermüller

"We introduce the concept of code perfumes as the counterpart to code smells, indicating the correct application of programming practices considered to be good. Using a catalogue of 25 code perfumes for, we empirically demonstrate that these represent frequent practices in, and we find that better programs indeed contain more code perfumes."

— Florian Obermüller, Lena Bloch, Luisa Greifenstein, Ute Heuer, and Gordon Fraser. Code Perfumes: Reporting Good Code to Encourage Learners. In *Proceedings of the 16th Workshop in Primary and Secondary Computing Education*, pages 1–10, 2021. doi:10.1145/3481312.3481346

Bibliography

Steven Arzt, Siegfried Rasthofer, Robert Hahn, and Eric Bodden. Using Targeted Symbolic Execution for Reducing False-Positives in Dataflow Analysis. In *Proceedings of the 4th International Workshop on State of the Art in Program Analysis*, pages 1–6, 2015. doi:10.1145/2771284.2771285.

Brian Chess and Gary McGraw. Static Analysis for Security. *IEEE Security & Privacy*, 2(6):76–79, 2004. doi:10.1109/msp.2004.111.

Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE, 2013. doi:10.1109/ICSE.2013.6606613.

Stephen C. Johnson. Lint, a C Program Checker, 1977.

Sunghun Kim and Michael D. Ernst. Which Warnings Should I Fix First? In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference*, pages 45–54, 2007. doi:10.1145/1287624.1287633.

Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. In Defense of Soundiness: A Manifesto. *Communications of the ACM*, 58(2):44–46, 2015. doi:10.1145/2644805.

Nachiappan Nagappan and Thomas Ball. Static Analysis Tools as Early Indicators of Pre-Release Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 580–586, 2005. doi:10.1145/1062455.1062558.

OASIS. Static Analysis Results Interchange Format (SARIF) Version 2.1.0 Plus Errata 01. `https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html`, 2023. [Online; accessed 08-03-2024].

Florian Obermüller, Lena Bloch, Luisa Greifenstein, Ute Heuer, and Gordon Fraser. Code Perfumes: Reporting Good Code to Encourage Learners. In *Proceedings of the 16th Workshop in Primary and Secondary Computing Education*, pages 1–10, 2021. doi:10.1145/3481312.3481346.