Warum verwenden Softwareentwickler keine Werkzeuge zur statischen Codeanalyse?

Thies Johannsen
Freie Universität Berlin

9. Juli 2015

Übersicht

Was ist statische Codeanalyse?

Was ist statische Codeanalyse?

- wird auch Quellcodeanalyse genannt
- formale Prüfung des Quellcodes auf Fehler
- wird ausgeführt **bevor** der Code übersetzt wird

Was ist statische Codeanalyse?

Welche Fehler werden erkannt?

- Typechecking (passen die Datentypen)
- Puffer-Überläufe
- Speicherlecks (new ohne `delete`, verlorene Pointer)
- unerreichbarer Code (siehe Beispiel)
- Stylechecking (werden bei `switch...case...` alle Optionen beachet)
- ...

Beispiel: unerreichbarer Code

Ein einfaches Beispiel:
Unerreichbarer Code

Beispiel: unerreichbarer Code

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main() {
5    uint32_t foo;
6
7    if(1)
8      foo = 0x12345678;
9    else
10     foo = 0x89abcdef;
11
12   printf("%x\n", foo);
13
14   return 0;
15 }
```

Beispiel: unerreichbarer Code

```
noerf@noerf-desktop:TestCode$ gcc -ansi -pedantic -Wall -O2 if_true.c -o if_true
noerf@noerf-desktop:TestCode$
```

Beispiel: unerreichbarer Code

```
noerf@noerf-desktop:TestCode$ gcc -ansi -pedantic -Wall -O2 if_true.c -o if_true
noerf@noerf-desktop:TestCode$ ./if_true
12345678
noerf@noerf-desktop:TestCode$
```

Beispiel: unerreichbarer Code

```
 0: 55                          push   %rbp
 1: 48 89 e5                    mov    %rsp,%rbp
 4: 48 83 ec 10                 sub    $0x10,%rsp
 8: 48 bf 00 00 00 00 00        movabs $0x0,%rdi
 f: 00 00 00
12: c7 45 fc 00 00 00 00        movl   $0x0,-0x4(%rbp)
19: c7 45 f8 78 56 34 12        movl   $0x12345678,-0x8(%rbp)
20: 8b 75 f8                    mov    -0x8(%rbp),%esi
23: b0 00                       mov    $0x0,%al
25: e8 00 00 00 00              callq  2a <main+0x2a>
2a: 31 f6                       xor    %esi,%esi
2c: 89 45 f4                    mov    %eax,-0xc(%rbp)
2f: 89 f0                       mov    %esi,%eax
31: 48 83 c4 10                 add    $0x10,%rsp
35: 5d                          pop    %rbp
36: c3                          retq
```

Beispiel: unerreichbarer Code

- Der Code wird ohne Warnungen übersetzt.
- Der tote Code wird vom Compiler wegoptimiert.
- Das Programm arbeitet genau wie erwartet.

Beispiel: unerreichbarer Code

- Der Code wird ohne Warnungen übersetzt.
- Der tote Code wird vom Compiler wegoptimiert.
- Das Programm arbeitet genau wie erwartet.
- Also alles in Ordnung?

Beispiel: unerreichbarer Code

- Der Code wird ohne Warnungen übersetzt.
- Der tote Code wird vom Compiler wegoptimiert.
- Das Programm arbeitet genau wie erwartet.
- Also alles in Ordnung?

- Konnte der Code wirklich wegoptimiert werden oder liegt das Problem eventuell woanders?
- Warum informiert mich der Compiler nicht darüber, dass komplette Codeblöcke entfernt werden?
- Codezeilen werden üblicherweise nicht ohne Grund geschrieben.

Beispiel: unerreichbarer Code

- Der Compiler prüft auf lexikalische/syntaktische Korrektheit.

Beispiel: unerreichbarer Code

- Der Compiler prüft auf lexikalische/syntaktische Korrektheit.
- Ist der Quellcode korrekt, wird er übersetzt und optimiert.

Beispiel: unerreichbarer Code

- Der Compiler prüft auf lexikalische/syntaktische Korrektheit.
- Ist der Quellcode korrekt, wird er übersetzt und optimiert.
- Semantik wird nicht betrachtet.

Beispiel: unerreichbarer Code

- Der Compiler prüft auf lexikalische/syntaktische Korrektheit.
- Ist der Quellcode korrekt, wird er übersetzt und optimiert.
- Semantik wird nicht betrachtet.

Eine mögliche Lösung:

Mit statischer Codeanalyse den Quelltext untersuchen.

Beispiel: unerreichbarer Code

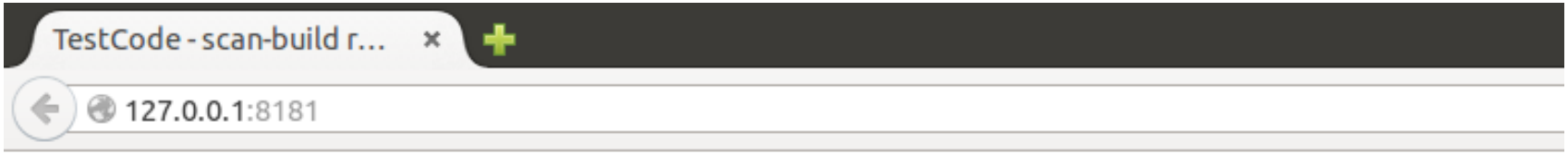
```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main() {
5    uint32_t foo;
6
7    if(1)
8      foo = 0x12345678;
9    else
10     foo = 0x89abcdef;
11
12   printf("%x\n", foo);
13
14   return 0;
15 }
```

Beispiel: unerreichbarer Code

Clang Static Analyzer

```
scan-build: Using '/usr/lib/llvm-3.6/bin/clang'
    for static analysis
if_true.c:10:11: warning: This statement is never executed
    foo = 0x89abcdef;
          ^~~~~~~~~~
1 warning generated.
scan-build: 1 bug found.
scan-build: Run
    'scan-view /tmp/scan-build-2015-07-01-111953-6304-1'
    to examine bug reports.
```

Beispiel: unerreichbarer Code



TestCode - scan-build results

User:	noerf@noerf-desktop
Working Directory:	/home/noerf/Uni/SS15/Seminar Software Engineering/TestCode
Command Line:	gcc -ansi -pedantic -Wall -O2 if_true.c -o if_true
Clang Version:	Ubuntu clang version 3.6.0-2ubuntu1 (tags/RELEASE_360/final) (based on LLVM 3.6.0)
Date:	Wed Jul 1 11:19:53 2015

Bug Summary

Bug Type	Quantity	Display?
All Bugs	**1**	☑
Dead code		
Unreachable code	1	☑

Reports

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length			
Dead code	Unreachable code	if_true.c	main	10	1	View Report	Report Bug	Open File

Beispiel: unerreichbarer Code

Annotated Source Code

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main() {
5    uint32_t foo;
6
7    if(1)
8      foo = 0x12345678;
9    else
10     foo = 0x89abcdef;
                 This statement is never executed
11
12   printf("%x\n", foo);
13
14   return 0;
15 }
16
```

Tabelle: Gängige Analysewerkzeuge

Tool	Programmiersprache
FxCop	.NET (prüft CIL)
Clang	C, C++, Objective-C, Objective-C++
Cppcheck	C, C++
Lint/Splint	C, C++
Checkstyle	Java
FindBugs	Java

Plugins sind für diverse IDEs verfügbar

In die meisten IDEs ist bereits eine einfache Analysefunktion integriert.

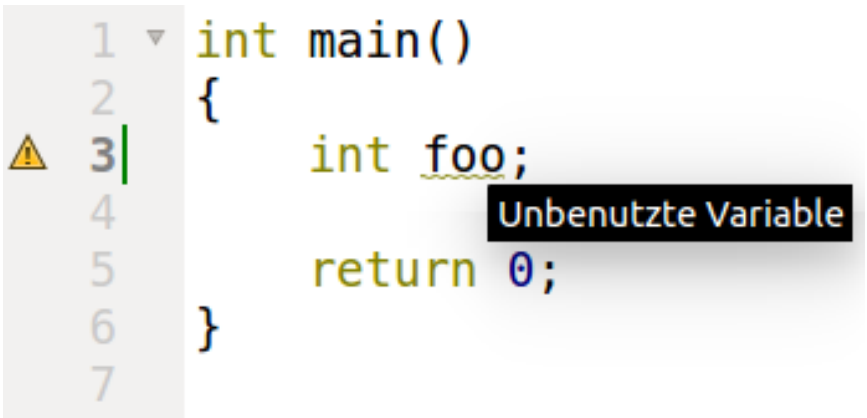
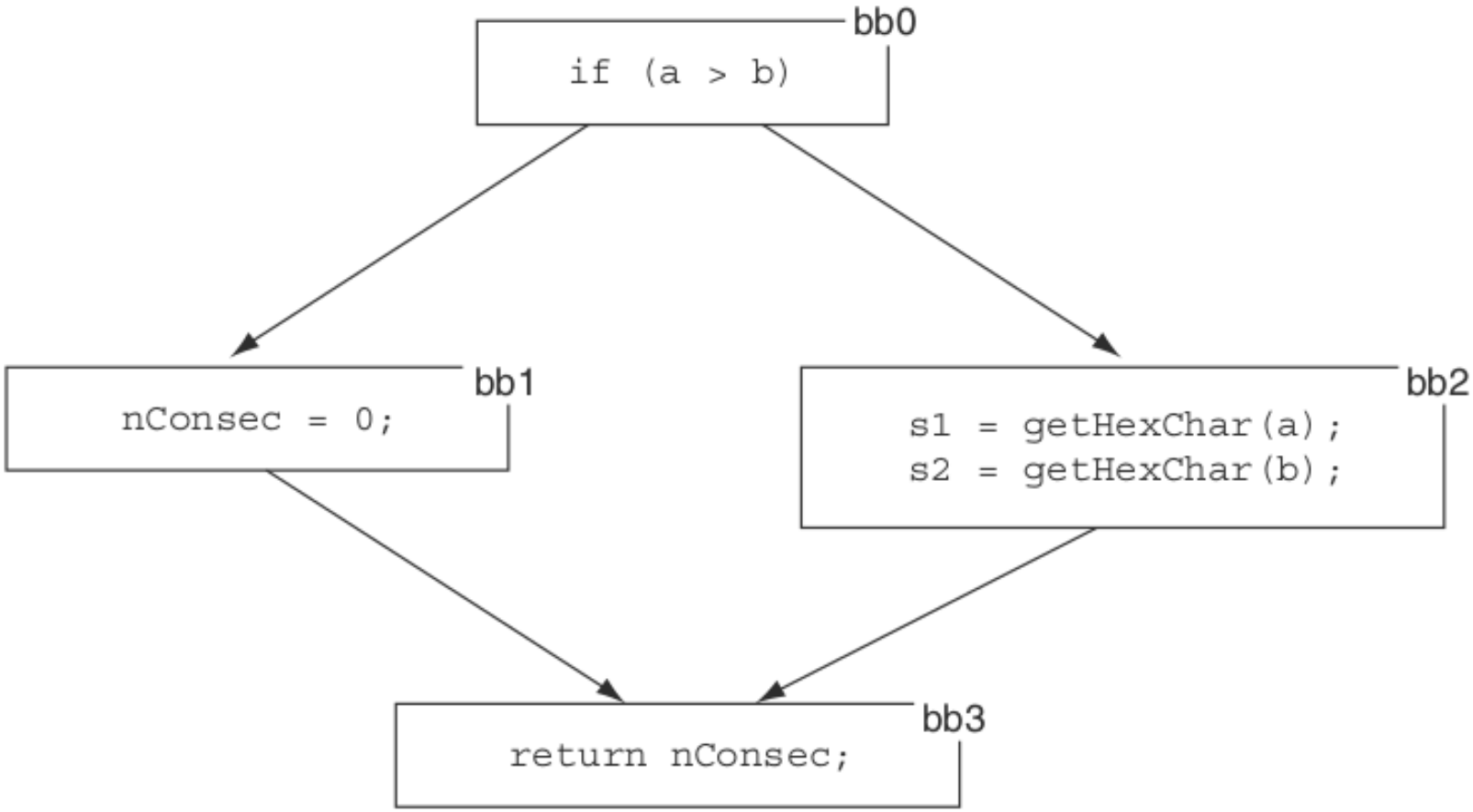


Abbildung: Hinweis auf unbenutzte Variable in QtCreator

Funktionsweise

```
1 if (a > b) {
2   nConsec = 0;
3 } else {
4   s1 = getHexChar(1);
5   s2 = getHexChar(2);
6 }
7 return nConsec;
```

Funktionsweise



(B. Chess, J. West, “Secure Programming with Static Analysis: Getting Software Security Right with Static Analysis“, Addison-Wesley Professional, 2007)

Funktionsweise

- Aufrufgraph für Funktionen wird erzeugt
- Datenfluss wird kontrolliert
  - Wird eine Variable mehrfach beschrieben ohne gelesen zu werden?
  - Wird aus uninitialisierten Variablen gelesen?
- Mit Hilfe eines Zustandsautomaten wird geprüft, ob auf jedes `new` ein `delete` folgt.
- weitere Sprachenabhängige Regeln

Freie Universität Berlin

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

Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, Robert Bowdidge

ICSE 2013

- Analysetools machen das finden von Bugs schneller und günstiger.

- Analysetools machen das finden von Bugs schneller und günstiger.
- Alle Teilnehmer der Studie sind der Meinung, dass die Verwendung von Analysetools Vorteile bringt.

- Analysetools machen das finden von Bugs schneller und günstiger.
- Alle Teilnehmer der Studie sind der Meinung, dass die Verwendung von Analysetools Vorteile bringt.

- Warum benutzt es trotzdem kaum jemand?

Forschungsfragen:

- What reasons do developers have for using or not using static analysis tools to find bugs?

Forschungsfragen:

- What reasons do developers have for using or not using static analysis tools to find bugs?
- How well do current static analysis tools fit into the workflows of developers? We define a workflow as the steps a developer takes when writing, inspecting and modifying their code.

Forschungsfragen:

- What reasons do developers have for using or not using static analysis tools to find bugs?
- How well do current static analysis tools fit into the workflows of developers? We define a workflow as the steps a developer takes when writing, inspecting and modifying their code.
- What improvements do developers want to see being made to static analysis tools?

Teilnehmer an der Studie:

- 20 Teilnehmer insgesamt
  - 16 professionelle Entwickler
  - 4 Studenten (graduate) mit Berufserfahrung
- 2 Teilnehmer hatten Erfahrung im Erstellen von Statischen Analysetools.
- 2 Teilnehmer wurden per Telefon, bzw. Videochat befragt.

TABLE I
DESCRIPTIVE STATISTICS REPORTED BY PARTICIPANTS.

Participant	Open-source Tools	Closed-source Tools	Local
Abby	FindBugs	IntelliJ	Yes
Adam	CheckStyle, FindBugs, PMD	IntelliJ	Yes
Andy	FindBugs, Lint	Jtest [20]	Yes
Chris	CheckStyle, FindBugs, Lint	Coverity	Yes
Cody	Dehydra	-	Yes
Frank	-	-	Yes
Gordon	Lint, CheckStyle, FindBugs	-	Yes
Jake	FindBugs, Lint	FlexLint, Klocwork Insight [21], Visual Studio [22]	Yes
James	Lint, CheckStyle, FindBugs	Visual Studio	Yes
Jason	Lint, FindBugs	-	Yes
John	CheckStyle, Copy/Paste Detector(CPD), FindBugs, Lint, PMD	CodePro [23]	Yes
Jordan	CheckStyle, FindBugs, PMD	Jtest	Yes
Josh	FindBugs, Lint	Coverity	No
Lee	CheckStyle, FindBugs, Lint	Visual Studio	Yes
Matt	Lint	FlexLint, PyCharm	Yes
Mike	cpplint, Lint	-	Yes
Phil	-	-	Yes
Ray	CheckStyle, FindBugs	-	Yes
Ryan	FindBugs, Lint	Coverity	Yes
Steve	CheckStyle, CPD, FindBugs, Lint	IntelliJ	Yes
Tony	CPD, FindBugs, Lint, Splint,cpplint, PMD, Checkstyle	Coverity	No

Methodik:

- „halb-strukturiertes“ Interview, 40 - 60 Minuten
  - Es wurde ein Frageboden vorbereitet, aber abhängig von den Antworten der Teilnehmer wurden spontan weitere Fragen gestellt oder ausgelassen.

Methodik:

- „halb-strukturiertes“ Interview, 40 - 60 Minuten
  - Es wurde ein Frageboden vorbereitet, aber abhängig von den Antworten der Teilnehmer wurden spontan weitere Fragen gestellt oder ausgelassen.
- Interaktives Interview
  - Die Teilnehmer wurden bei ihrer Arbeit beobachtet und gebeten ihren Arbeitsablauf zu erklären.
  - Währenddessen wurden Fragen gestellt:
    - Now that you have run your tool and gotten your feedback, what is your next move(s)?
    - Do you configure the settings of your tool from default? If so, how?
    - Does this static analysis tool aid in assessing what to do about a warning?
    - Do you feel that ”quick fixes” or code suggestions would be helpful if they were available?
  - 6 Teilnehmer konnten aus Vertraulichkeitsgründen nicht am interaktiven Interview teilnehmen, diese bekamen Aufgaben gestellt.
  - Den 2 Teilnehmern, die per Telefon, bzw. Videochat befragt wurden, wurde ein Szenario vorgelegt und es wurde gefragt, wie sie vorgehen würden.

Methodik:

- „halb-strukturiertes“ Interview, 40 - 60 Minuten
  - Es wurde ein Frageboden vorbereitet, aber abhängig von den Antworten der Teilnehmer wurden spontan weitere Fragen gestellt oder ausgelassen.
- Interaktives Interview
  - Die Teilnehmer wurden bei ihrer Arbeit beobachtet und gebeten ihren Arbeitsablauf zu erklären.
  - Währenddessen wurden Fragen gestellt:
    - Now that you have run your tool and gotten your feedback, what is your next move(s)?
    - Do you configure the settings of your tool from default? If so, how?
    - Does this static analysis tool aid in assessing what to do about a warning?
    - Do you feel that ”quick fixes” or code suggestions would be helpful if they were available?
  - 6 Teilnehmer konnten aus Vertraulichkeitsgründen nicht am interaktiven Interview teilnehmen, diese bekamen Aufgaben gestellt.
  - Den 2 Teilnehmern, die per Telefon, bzw. Videochat befragt wurden, wurde ein Szenario vorgelegt und es wurde gefragt, wie sie vorgehen würden.
- Participatory Design
  - Den Teilnehmern wurde ein leeres Blatt Papier gegeben und sie durften nach belieben schreiben/zeichnen, welche Verbesserungen an den Analysewerkzeugen sie gerne hätten.

Auswerten der Antworten:

- abschreiben aller gegebenen Antworten
- Kategorisieren nach R. Gordon „Coding interview responses“

Auswerten der Antworten:

- abschreiben aller gegebenen Antworten
- Kategorisieren nach R. Gordon „Coding interview responses“
  - Tool Output
  - Supporting Teamwork
  - User Input and Customizability
  - Result Understandability
  - Developer Workflows

Auswerten der Antworten:

- abschreiben aller gegebenen Antworten
- Kategorisieren nach R. Gordon „Coding interview responses“
  - Tool Output
  - Supporting Teamwork
  - User Input and Customizability
  - Result Understandability
  - Developer Workflows
- Jede Antwort wurde markiert, in welche Kategorie sie eingeordnet werden kann und ob die Antwort positiv oder negativ ist.

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

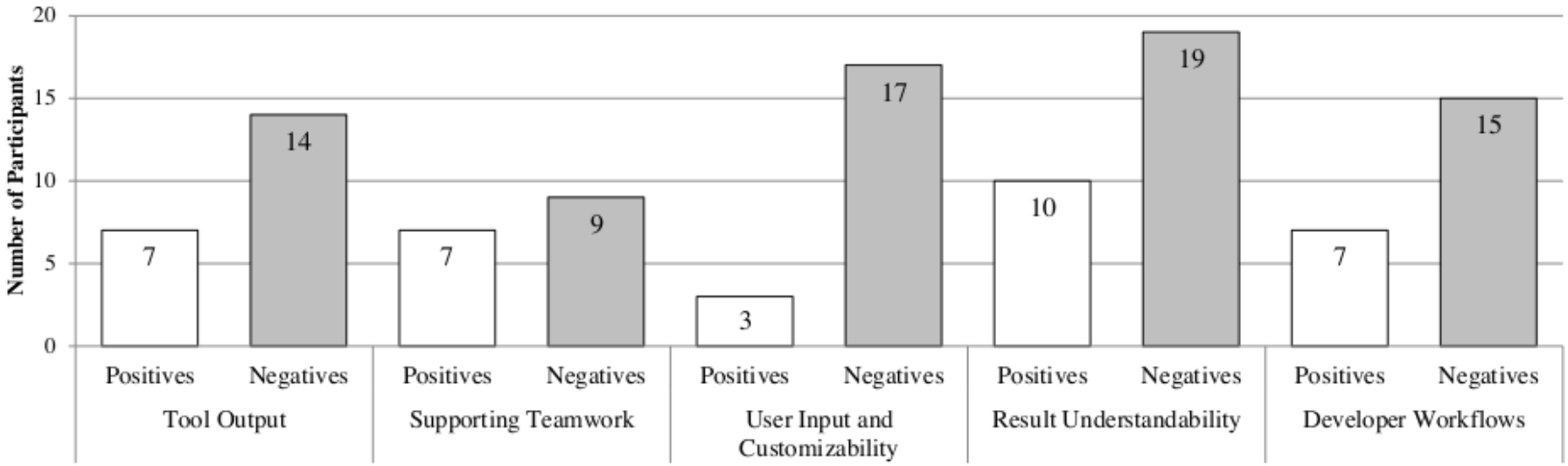


Fig. 1. The number of participants in each category expressing the good and the bad about static analysis tools they have used.

Einige beispielhafte Antworten:

Einige beispielhafte Antworten:

- ”...like I mentioned with FlexLint it gives you so many warnings and sifting through them is so, arduous that whenever I just look at it I’m like ehhh forget this.”

Einige beispielhafte Antworten:

- ”...like I mentioned with FlexLint it gives you so many warnings and sifting through them is so, arduous that whenever I just look at it I’m like ehhh forget this.”
- ”so now I wanna know why raising a string exception is bad. Like what should I be doing instead?”

Einige beispielhafte Antworten:

- ”...like I mentioned with FlexLint it gives you so many warnings and sifting through them is so, arduous that whenever I just look at it I’m like ehhh forget this.”
- ”so now I wanna know why raising a string exception is bad. Like what should I be doing instead?”
- ”I find that the information they provide is not very useful, so I tend to ignore them.”

Einige beispielhafte Antworten:

- ”...like I mentioned with FlexLint it gives you so many warnings and sifting through them is so, arduous that whenever I just look at it I’m like ehhh forget this.”
- ”so now I wanna know why raising a string exception is bad. Like what should I be doing instead?”
- ”I find that the information they provide is not very useful, so I tend to ignore them.”
- ”Clang is my favorite. Its built into the compiler. You don’t have to invoke anything special.”

What reasons do developers have for using or not using static analysis tools to find bugs?

+ Bugs können automatisch gefunden werden.
  - ”anything that will automate a mundane task is great.”
+ Das Tool ist bereits in der IDE integriert.
+ Es motiviert in Teams auf „dumme Fehler“ zu achten.

What reasons do developers have for using or not using static analysis tools to find bugs?

+ Bugs können automatisch gefunden werden.
  - ”anything that will automate a mundane task is great.”
+ Das Tool ist bereits in der IDE integriert.
+ Es motiviert in Teams auf „dumme Fehler“ zu achten.

- false positives
- schlechte/unverständliche Fehlermeldungen
  - ”it’s one thing to give an error message, it’s another thing to give a useful error message.”
- umständliches Arbeiten in Teams
  - Einstellungen lassen sich nicht kopieren
- schwer zu konfigurieren
- ”quick fixes” fehlen oder sind schlecht implementiert

How well do current static analysis tools fit into the workflows of developers?

- 19 der 20 Teilnehmer ist eine gute Integration in den Arbeitsablauf wichtig.
  - gute Integration in die IDE
  - gute Integration in den Compiler

How well do current static analysis tools fit into the workflows of developers?

- 19 der 20 Teilnehmer ist eine gute Integration in den Arbeitsablauf wichtig.
  - gute Integration in die IDE
  - gute Integration in den Compiler
- Arbeitsumgebung sollte nicht gewechselt werden müssen
  - ”if it disrupts your flow, you’re not gonne use it.”

What improvements do developers want to see being made to static analysis tools?

- bessere Darstellung von ”quick fixes”
  - es sollte eine Vorschau geben, wie der Code geändert wird, z.B. geteilte Ansichten
  - ”three option dialog” (anwenden, abbrechen, Schritt für Schritt)

What improvements do developers want to see being made to static analysis tools?

- bessere Darstellung von ”quick fixes”
  - es sollte eine Vorschau geben, wie der Code geändert wird, z.B. geteilte Ansichten
  - ”three option dialog” (anwenden, abbrechen, Schritt für Schritt)
- Feedback sollte schneller kommen
  - Analyse sollte immer im Hintergrund laufen
  - Analyse sollte bei jedem Compile-Vorgang durchgeführt werden

What improvements do developers want to see being made to static analysis tools?

- bessere Darstellung von ”quick fixes”
  - es sollte eine Vorschau geben, wie der Code geändert wird, z.B. geteilte Ansichten
  - ”three option dialog” (anwenden, abbrechen, Schritt für Schritt)
- Feedback sollte schneller kommen
  - Analyse sollte immer im Hintergrund laufen
  - Analyse sollte bei jedem Compile-Vorgang durchgeführt werden
- gefundene Fehler sollten einfach temporär zurückgestellt und mit anderen geteilt werden können

What improvements do developers want to see being made to static analysis tools?

- bessere Darstellung von ”quick fixes”
  - es sollte eine Vorschau geben, wie der Code geändert wird, z.B. geteilte Ansichten
  - ”three option dialog” (anwenden, abbrechen, Schritt für Schritt)
- Feedback sollte schneller kommen
  - Analyse sollte immer im Hintergrund laufen
  - Analyse sollte bei jedem Compile-Vorgang durchgeführt werden
- gefundene Fehler sollten einfach temporär zurückgestellt und mit anderen geteilt werden können
- eine Übersicht, wo im Code sich viele Fehler befinden (ähnlich CodeCity)

What improvements do developers want to see being made to static analysis tools?

- bessere Darstellung von ”quick fixes”
  - es sollte eine Vorschau geben, wie der Code geändert wird, z.B. geteilte Ansichten
  - ”three option dialog” (anwenden, abbrechen, Schritt für Schritt)
- Feedback sollte schneller kommen
  - Analyse sollte immer im Hintergrund laufen
  - Analyse sollte bei jedem Compile-Vorgang durchgeführt werden
- gefundene Fehler sollten einfach temporär zurückgestellt und mit anderen geteilt werden können
- eine Übersicht, wo im Code sich viele Fehler befinden (ähnlich CodeCity)
- statt verschiedener Farben für unterschiedlich schwere Fehler lieber eine Farbe mit unterschiedlicher Intensität

Externe Gültigkeit:

- nur 20 Teilnehmer
  - Mehr Teilnehmer hätten andererseits den Aufwand der Auswertung enorm erhöht und dadurch hätte eventuell die Sorgfalt gelitten.

Externe Gültigkeit:

- nur 20 Teilnehmer
  - Mehr Teilnehmer hätten andererseits den Aufwand der Auswertung enorm erhöht und dadurch hätte eventuell die Sorgfalt gelitten.
- Alle Teilnehmer arbeiten auch außerhalb der Studie mit Werkzeugen zur statischen Analyse.

Externe Gültigkeit:

- nur 20 Teilnehmer
  - Mehr Teilnehmer hätten andererseits den Aufwand der Auswertung enorm erhöht und dadurch hätte eventuell die Sorgfalt gelitten.
- Alle Teilnehmer arbeiten auch außerhalb der Studie mit Werkzeugen zur statischen Analyse.
- 2 Teilnehmer haben bereits selbst Werkzeuge zur statischen Analyse entwickelt und haben so eventuell eine nicht repräsentative Sicht.

Interne Gültigkeit:

- 2 Teilnehmer waren nicht vor Ort und mussten über Telefon/Videochat befragt werden.

Interne Gültigkeit:

- 2 Teilnehmer waren nicht vor Ort und mussten über Telefon/Videochat befragt werden.
- 6 Teilnehmer durften nicht an ihrem normalen Arbeitsplatz arbeiten und bekamen daher andere Aufgaben.

Interne Gültigkeit:

- 2 Teilnehmer waren nicht vor Ort und mussten über Telefon/Videochat befragt werden.
- 6 Teilnehmer durften nicht an ihrem normalen Arbeitsplatz arbeiten und bekamen daher andere Aufgaben.
- Zu Beginn wurden den Teilnehmern die Ziele der Studie genannt.

False Positive

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5   printf("%s\n", argv[0]);
6   return 0;
7 }
```

False Positive

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5   printf("%s\n", argv[0]);
6   return 0;
7 }
```

```
unused.c: (in function main)
unused.c:3:14: Parameter argc not used
A function parameter is not used in the body of the function. If the
argument is needed for type compatibility or future plans, use
/*@unused@*/ in the argument declaration. (Use -paramuse to inhibit
warning)
```

False Positive

```
1  void foo(char c) {
2    unsigned uppercase = 0;
3    unsigned keycode = 0;
4
5    switch(c) {
6      case 'A': uppercase = 1;
7      case 'a': keycode = 65;
8          break;
9      case 'B': uppercase = 1;
10     case 'b': keycode = 66;
11         break;
12     /* ... */
13   }
14   /* stuff */
15 }
```

False Positive

```
1  void foo(char c) {
2    unsigned uppercase = 0;
3    unsigned keycode = 0;
4
5    switch(c) {
6      case 'A': uppercase = 1;
7      case 'a': keycode = 65;
8          break;
9      case 'B': uppercase = 1;
10     case 'b': keycode = 66;
11         break;
12     /* ... */
13   }
14   /* stuff */
15 }
```

```
fallthrough.c: (in function foo)
fallthrough.c:7:10: Fall through case (no preceding break)
Execution falls through from the previous case (use /*@fallthrough@*/
to mark fallthrough cases). (Use -casebreak to inhibit warning)
fallthrough.c:10:10: Fall through case (no preceding break)
```

(Hoffmann, „Software-Qualität“, Springer Vieweg, 2013)

False Positive

Können false positives verhindert werden?

- Schwierig bis unmöglich
- Code, der in einem Fall korrekt ist, kann in einem anderen Fall ein Fehler sein
- es ist möglich einzelne Meldungen zu unterdrücken

False Positive

```
1  void foo(char c) {
2    unsigned uppercase = 0;
3    unsigned keycode = 0;
4
5    switch(c) {
6      case 'A': uppercase = 1;
7           /*@fallthrough@*/
8      case 'a': keycode = 65;
9           break;
10     case 'B': uppercase = 1;
11          /*@fallthrough@*/
12     case 'b': keycode = 66;
13          break;
14     /* ... */
15     }
16     /* stuff */
17 }
```

False Positive

Können false positives verhindert werden?

- schwierig bis unmöglich
- Code, der in einem Fall korrekt ist, kann in einem anderen Fall ein Fehler sein
- es ist möglich einzelne Meldungen zu unterdrücken

- nicht sehr komfortabel
- der Code wird unübersichtlich

Überflutung

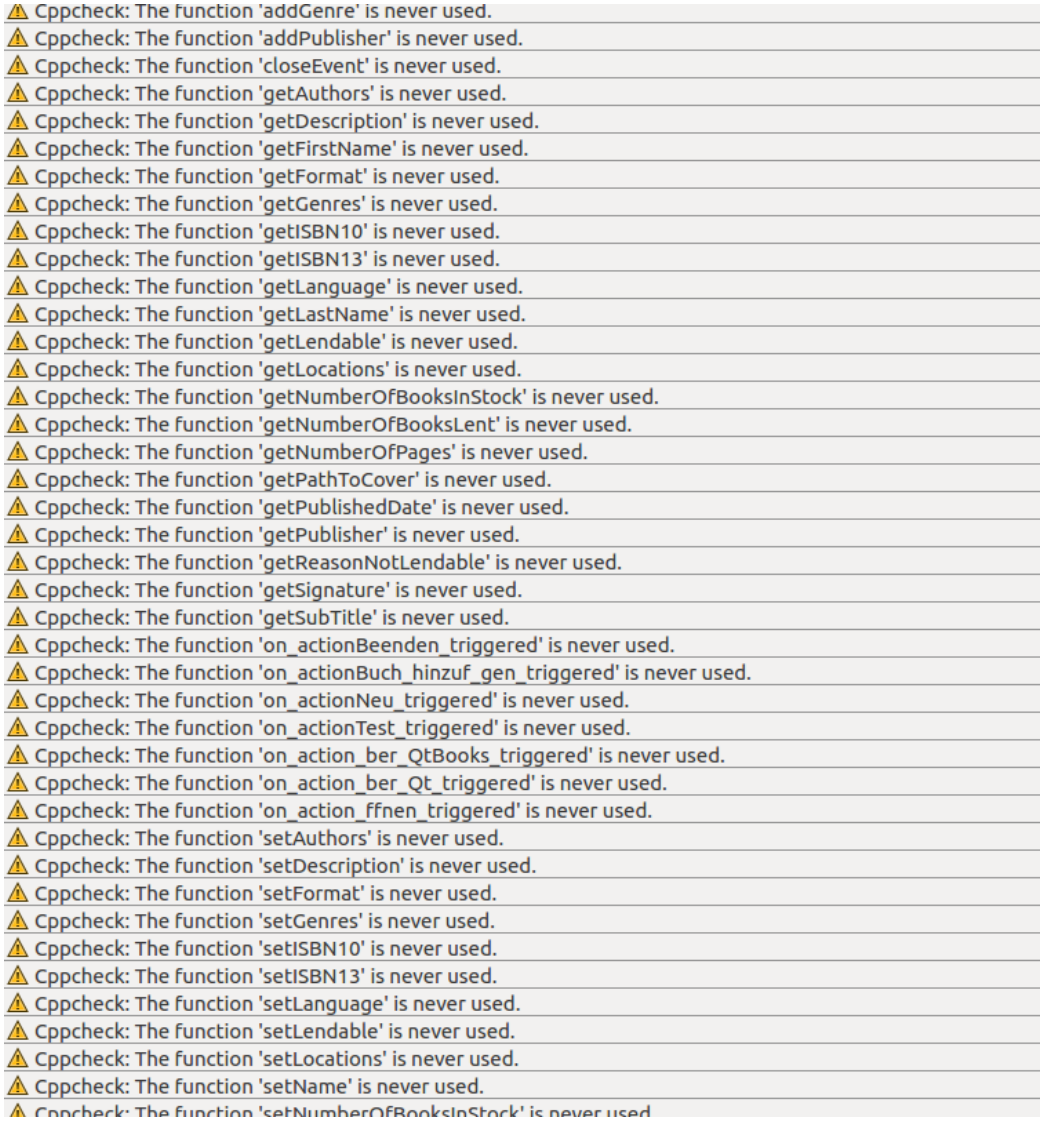
Was passiert, wenn eine neue Klasse hinzugefügt wird?

Überflutung

Was passiert, wenn eine neue Klasse hinzugefügt wird?

- Wir bekommen für jede nicht benutzte Funktion eine Warnung.

Überflutung



Quick Fixes

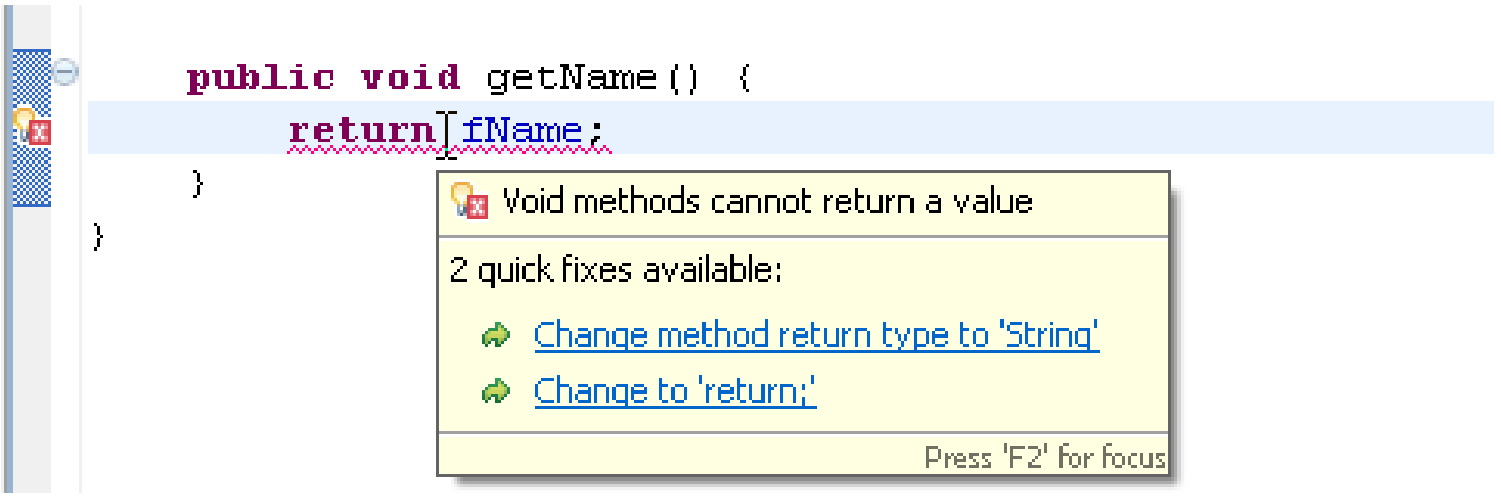


Abbildung: quick fix in eclipse (help.eclipse.org)

Quick Fixes

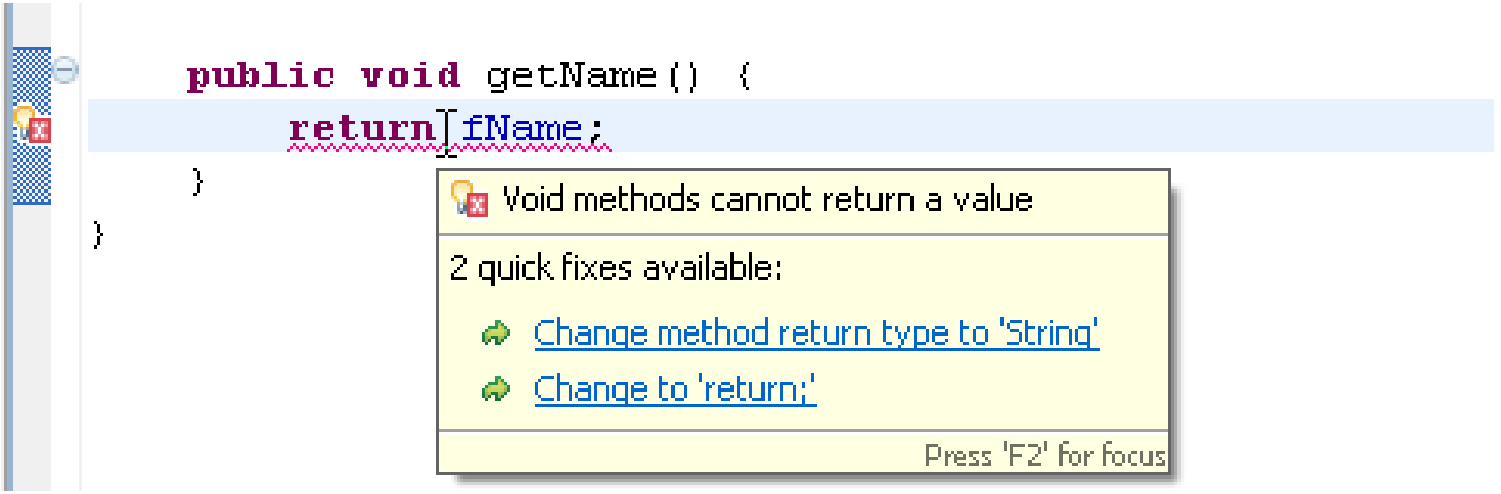


Abbildung: quick fix in eclipse (help.eclipse.org)

- Vorschläge, den Code automatisch anzupassen
- bei größeren Änderungen sehr riskant
  - Ist die Semantik erhalten geblieben?
  - Kommt es dadurch zu neuen Fehlern?

schlechte Fehlermeldungen

- ”D’oh! A nonsensical method invocation”
  - This partical method invocation doesn’t make sense, for reasons that should be apparent from inspection.

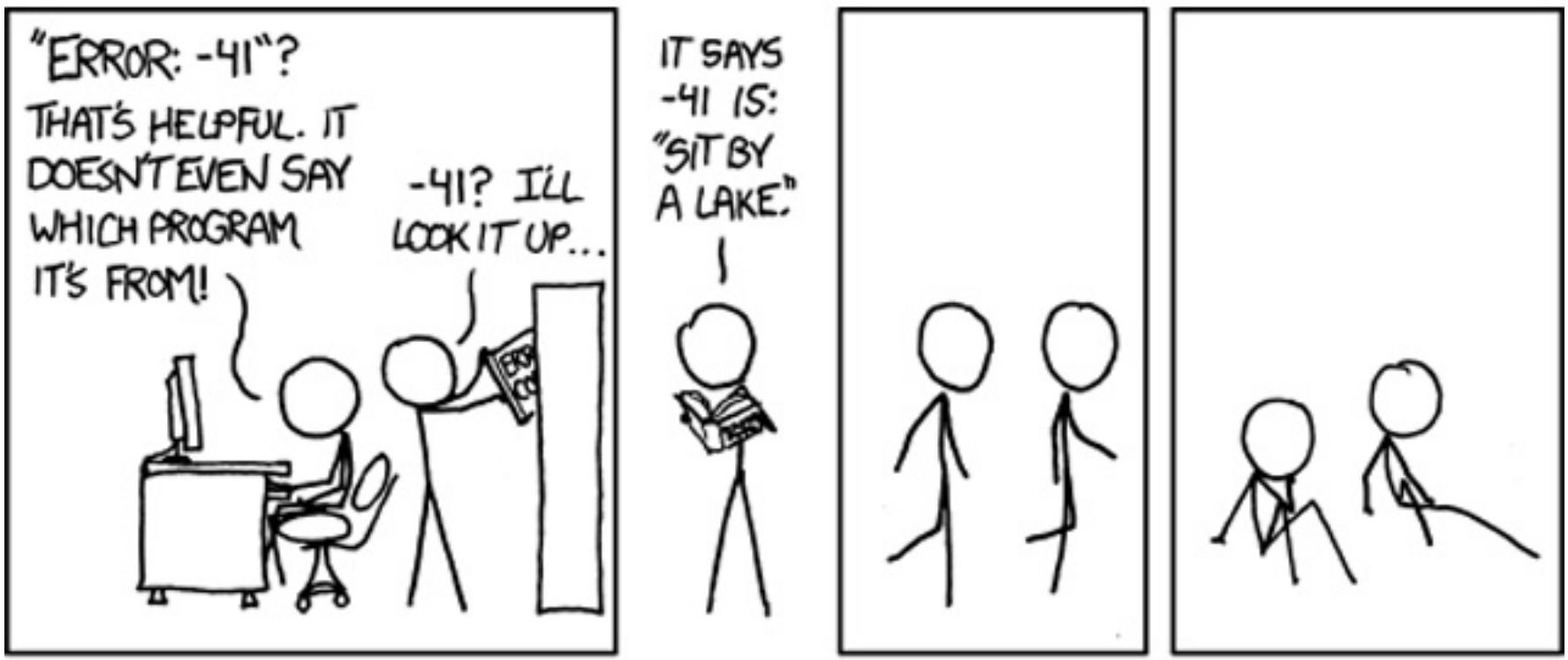
schlechte Fehlermeldungen

- ”D’oh! A nonsensical method invocation”
  - This partical method invocation doesn’t make sense, for reasons that should be apparent from inspection.

- Wäre der Funktionsaufruf so offensichtlich sinnlos, dann hätte der Programmierer ihn nicht gemacht.

Zusammenfassung

- statische Codeanalyse kann eine Wertvolle Hilfe sein
- statische Codeanalyse ist kein Allheilmittel, die eigentliche Arbeit liegt weiter beim Programmierer
- die größten Probleme
  - Anzahl der Fehlermeldungen
  - False Positives
  - umständliche Konfiguration


"ERROR: -41"?
THAT'S HELPFUL. IT DOESN'T EVEN SAY WHICH PROGRAM IT'S FROM!
-41? I'LL LOOK IT UP...
IT SAYS -41 IS: "SIT BY A LAKE."