Programme, die Programme prüfen

Andreas Zeller • Universität des Saarlandes

Programme, die Programme prüfen

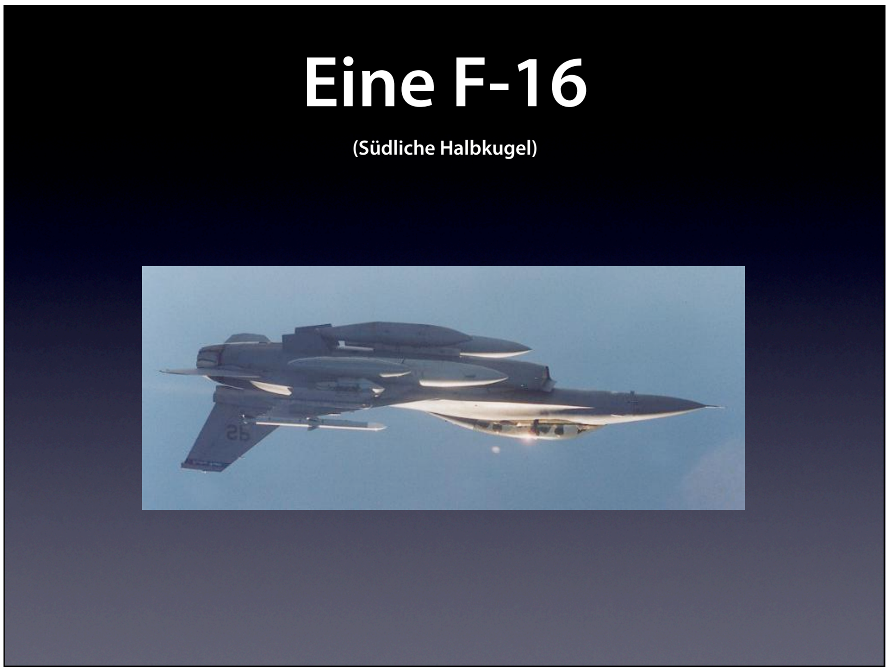
Jeder Programmierer kennt die Situation: Ein Programm läuft nicht so, wie es soll. Ich stelle Techniken vor, die automatisch

(a) die Ursachen eines Fehlverhaltens finden - indem wir genau die Aspekte isolieren, die das Zustandekommen eines Fehlers verursachen;
(b) Programmfehler finden - indem wir Anwendungen systematisch und vollautomatisch testen; und
(c) vorhersagen, wo in Zukunft Fehler auftreten werden - indem wir maschinell lernen, welche Code- und Prozesseigenschaften bisher mit

Eine F-16

(Nördliche Halbkugel)

An F-16 fighter plane on the northern hemisphere.
Why the northern hemisphere, you ask?



Eine F-16

(Südliche Halbkugel)

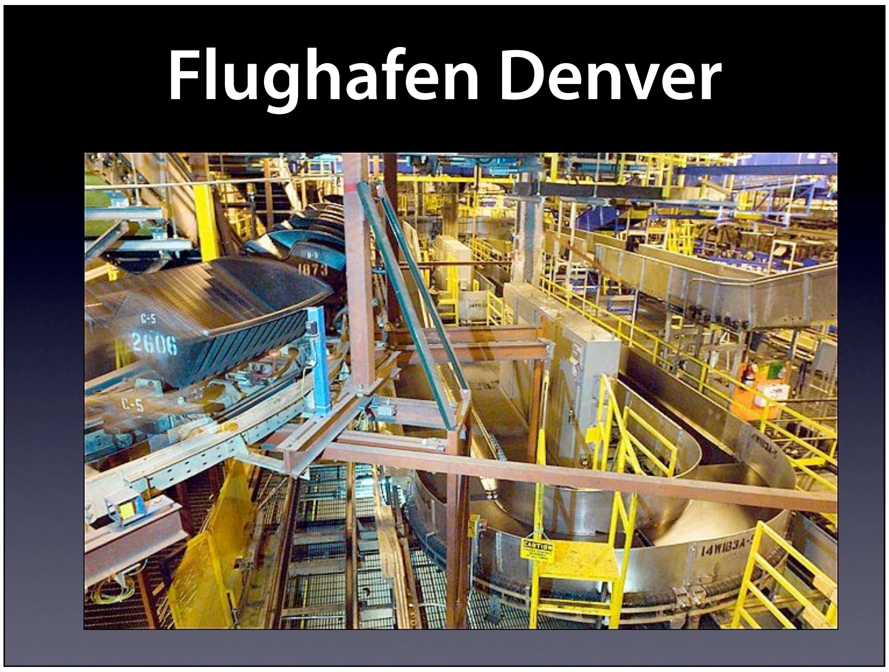
Because this is what an F-16 on the southern hemisphere would look like.
(BTW, interesting effect if you drop a bomb :-)

From risks.digest, volume 3, issue 44:
o Since the F-16 is a fly-by-wire aircraft, the computer keeps the pilot from
  doing dumb things to himself. So if the pilot jerks hard over on the
  joystick, the computer will instruct the flight surfaces to make a nice and
  easy 4 or 5 G flip. But the plane can withstand a much higher flip than that.
  So when they were 'flying' the F-16 in simulation over the equator, the
  computer got confused and instantly flipped the plane over, killing the

F-16 Fahrgestell

From risks.digest, volume 3, issue 44:
o One of the first things the Air Force test pilots tried on an early F-16 was to tell the computer to raise the landing gear while standing still on the runway. Guess what happened? Scratch one F-16. (my friend says there is a new subroutine in the code called 'wait_on_wheels' now...) [weight?]

(Folklore has it that the programmer checked the height above sea level rather than the height above ground - AZ)

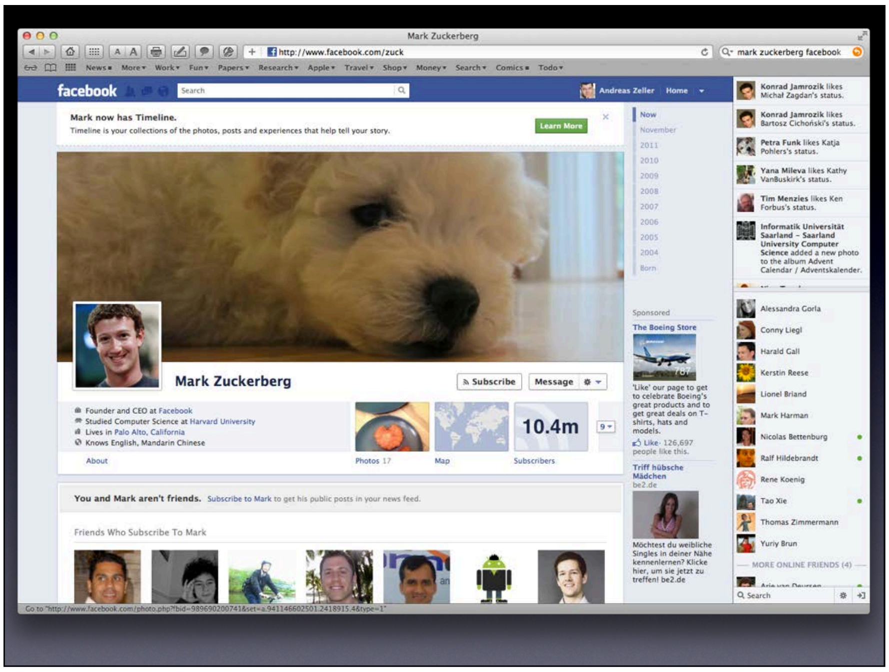


Flughafen Denver

Airport opened 16 mos later;
Software firm (BEA) got bankrupt
Overall damage 1.8 bln

What camera crews depicted was truly a disaster; carts jammed together, damaged luggage everywhere, some bags literally split in half, and the tattered remains of clothing strewn about causing subsequent carts to derail. Finally, adding insult to injury, half the luggage that survived the ordeal ended up at the wrong terminal.

http://www.aeroxp.org/2009/01/lesson-on-infinite-loops/
http://www.youtube.com/watch?v=fYTJ9v2vsaE


Mark Zuckerberg
http://www.facebook.com/zuck
mark zuckerberg facebook
News More Work Fun Papers Research Apple Travel Shop Money Search Comics Todo
facebook
Search
Andreas Zeller
Home

Mark now has Timeline.
Timeline is your collections of the photos, posts and experiences that help tell your story.
Learn More

Now
November
2011
2010
2009
2008
2007
2006
2005
2004
Born

Mark Zuckerberg
Subscribe
Message

Founder and CEO at Facebook
Studied Computer Science at Harvard University
Lives in Palo Alto, California
Knows English, Mandarin Chinese
About
Photos 17
Map
10.4m
Subscribers

You and Mark aren't friends. Subscribe to Mark to get his public posts in your news feed.

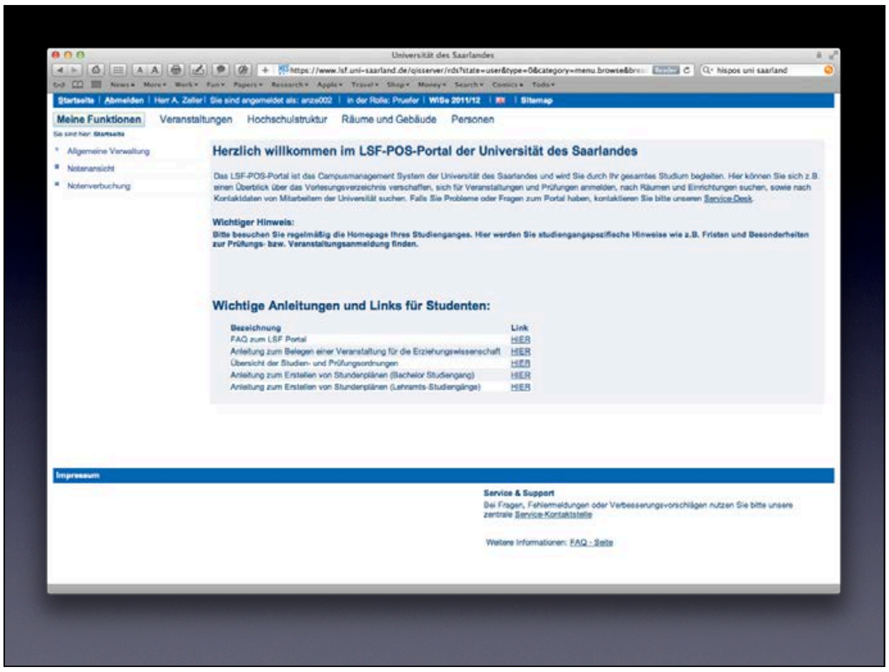
Friends Who Subscribe To Mark

Sponsored
The Boeing Store
'Like' our page to get to celebrate Boeing's great products and to get great deals on T-shirts, hats and models.
Like · 126,697 people like this.

Triff hübsche Mädchen
be2.de
Möchtest du weibliche Singles in deiner Nähe kennenlernen? Klicke hier, um sie jetzt zu treffen! be2.de

Konrad Jamrozik likes Michał Zagdan's status.
Konrad Jamrozik likes Bartosz Cichoński's status.
Petra Funk likes Katja Pohlers's status.
Yana Mileva likes Kathy VanBuskirk's status.
Tim Menzies likes Ken Forbus's status.
Informatik Universität Saarland – Saarland University Computer Science added a new photo to the album Advent Calendar / Adventskalender.

Alessandra Gorla
Conny Liegl
Harald Gall
Kerstin Reese
Lionel Briand
Mark Harman
Nicolas Bettenburg
Ralf Hildebrandt
Rene Koenig
Tao Xie
Thomas Zimmermann
Yuriy Brun
MORE ONLINE FRIENDS (4)
Search


Universität des Saarlandes
https://www.lsf.uni-saarland.de/qisserver/rds?state=user&type=0&category=menu.browse&bre
hispos uni saarland
News More Work Fun Papers Research Apple Travel Shop Money Search Comics Todo

Startseite | Abmelden | Herr A. Zeller | Sie sind angemeldet als: anze002 | in der Rolle: Pruefer | WiSe 2011/12 | | Sitemap

Meine Funktionen   Veranstaltungen   Hochschulstruktur   Räume und Gebäude   Personen

Sie sind hier: Startseite

- Allgemeine Verwaltung
- Notenansicht
- Notenverbuchung

Herzlich willkommen im LSF-POS-Portal der Universität des Saarlandes

Das LSF-POS-Portal ist das Campusmanagement System der Universität des Saarlandes und wird Sie durch Ihr gesamtes Studium begleiten. Hier können Sie sich z.B. einen Überblick über das Vorlesungsverzeichnis verschaffen, sich für Veranstaltungen und Prüfungen anmelden, nach Räumen und Einrichtungen suchen, sowie nach Kontaktdaten von Mitarbeitern der Universität suchen. Falls Sie Probleme oder Fragen zum Portal haben, kontaktieren Sie bitte unseren Service-Desk.

Wichtiger Hinweis:
Bitte besuchen Sie regelmäßig die Homepage Ihres Studienganges. Hier werden Sie studiengangspezifische Hinweise wie z.B. Fristen und Besonderheiten zur Prüfungs- bzw. Veranstaltungsanmeldung finden.

Wichtige Anleitungen und Links für Studenten:

Bezeichnung	Link
FAQ zum LSF Portal	HIER
Anleitung zum Belegen einer Veranstaltung für die Erziehungswissenschaft	HIER
Übersicht der Studien- und Prüfungsordnungen	HIER
Anleitung zum Erstellen von Stundenplänen (Bachelor Studiengang)	HIER
Anleitung zum Erstellen von Stundenplänen (Lehramts-Studiengänge)	HIER

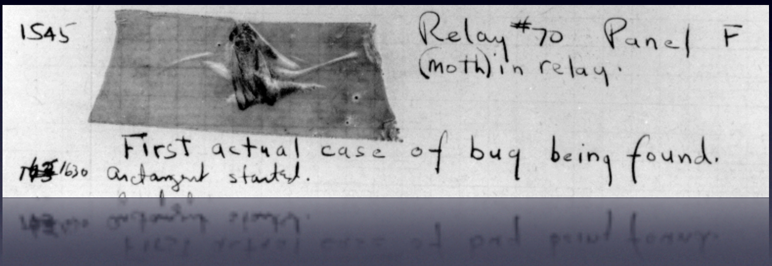
Impressum

Service & Support
Bei Fragen, Fehlermeldungen oder Verbesserungsvorschlägen nutzen Sie bitte unsere zentrale Service-Kontaktstelle

Weitere Informationen: FAQ - Seite

Der erste Bug

9. September 1947


1545

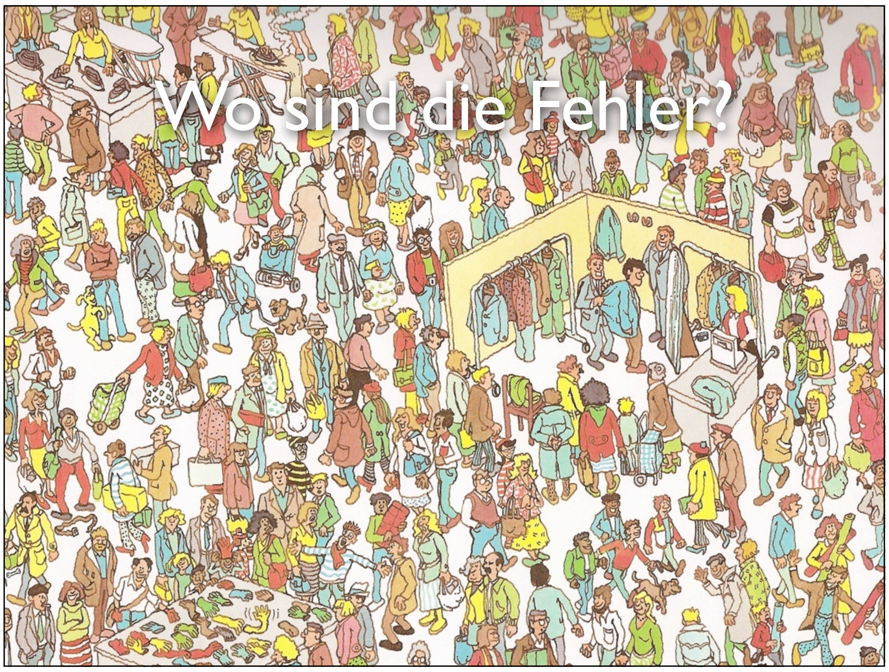
Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

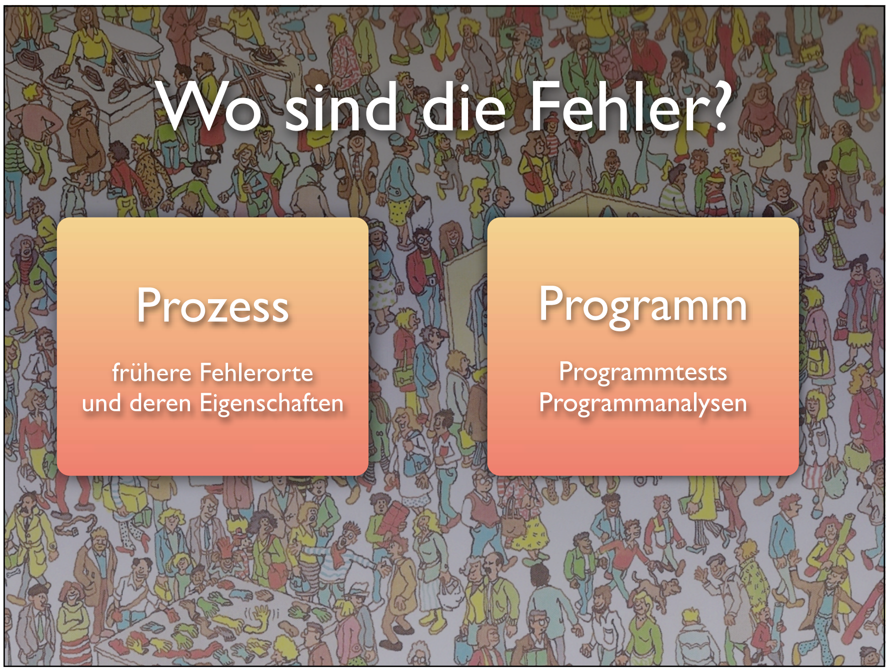
1630 antangent started.

Retrieved by a technician from the Harvard Mark II machine on September 9, 1947.

Now on display at the Smithsonian, Washington



Wo sind die Fehler?



Wo sind die Fehler?

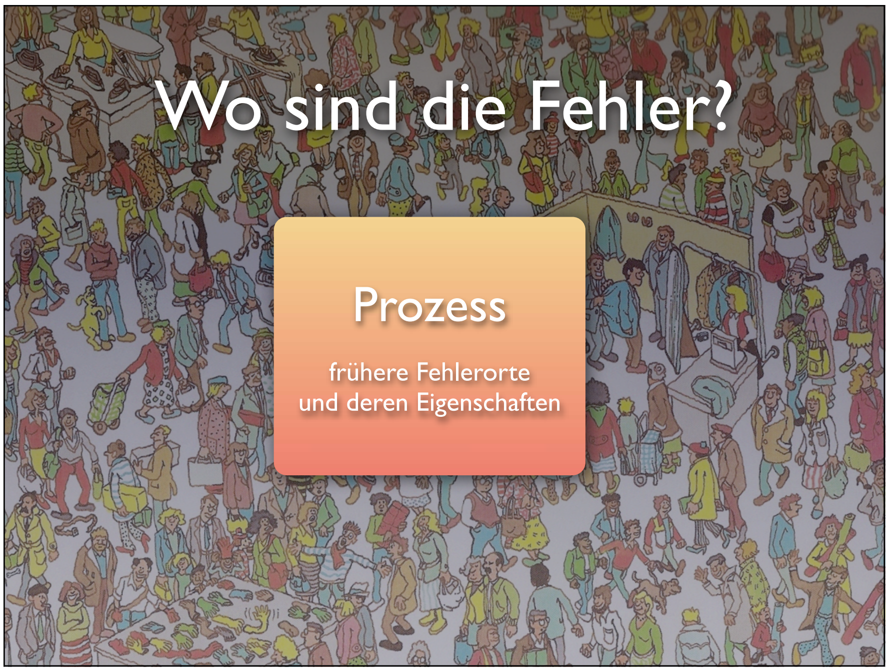
Prozess

frühere Fehlerorte
und deren Eigenschaften

Programm

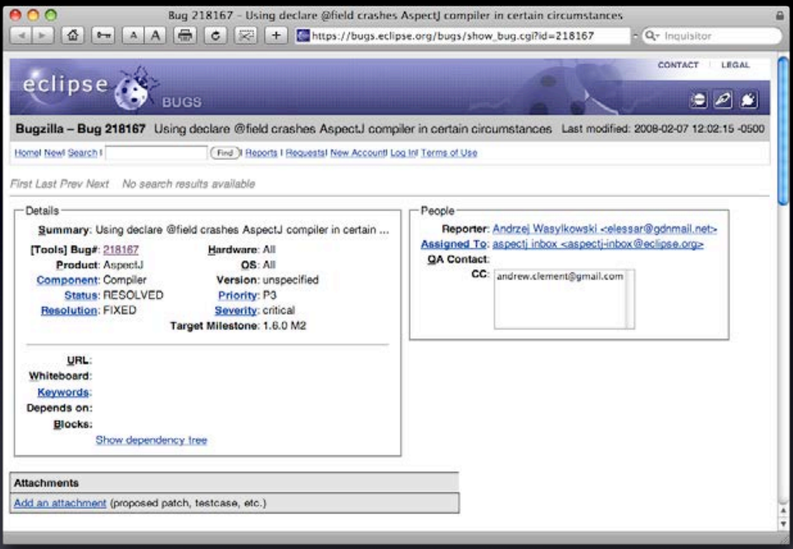
Programmtests
Programmanalysen

Wo sind die Fehler?



Prozess

frühere Fehlerorte
und deren Eigenschaften



Bug 218167 – Using declare @field crashes AspectJ compiler in certain circumstances

https://bugs.eclipse.org/bugs/show_bug.cgi?id=218167

Bugzilla – Bug 218167 Using declare @field crashes AspectJ compiler in certain circumstances Last modified: 2008-02-07 12:02:15 -0500

Home | New | Search | Find | Reports | Requests | New Account | Log In | Terms of Use

First Last Prev Next No search results available

Details

Summary: Using declare @field crashes AspectJ compiler in certain ...

[Tools] Bug#: 218167
Product: AspectJ
Component: Compiler
Status: RESOLVED
Resolution: FIXED

Hardware: All
OS: All
Version: unspecified
Priority: P3
Severity: critical
Target Milestone: 1.6.0 M2

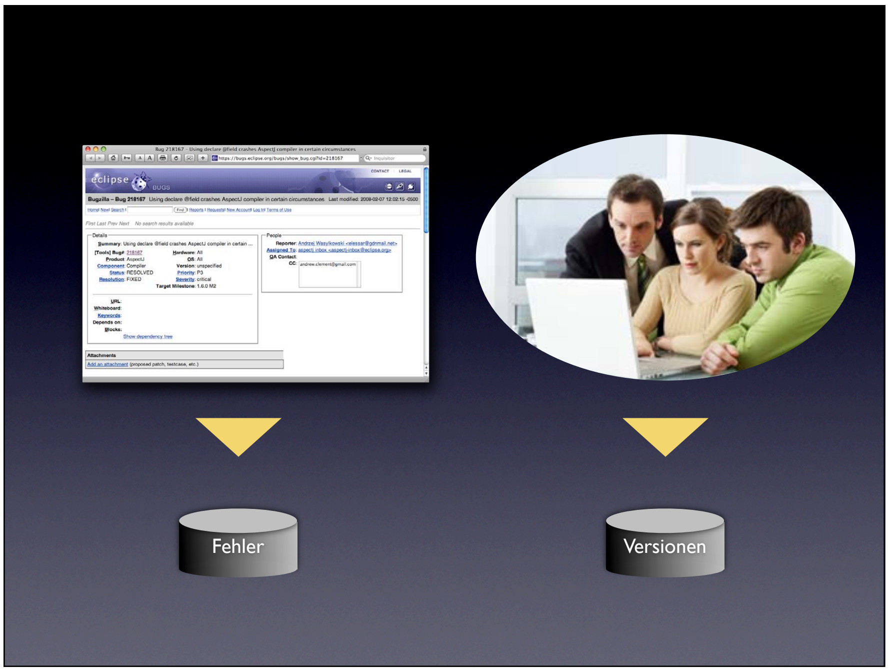
URL:
Whiteboard:
Keywords:
Depends on:
Blocks:
Show dependency tree

People

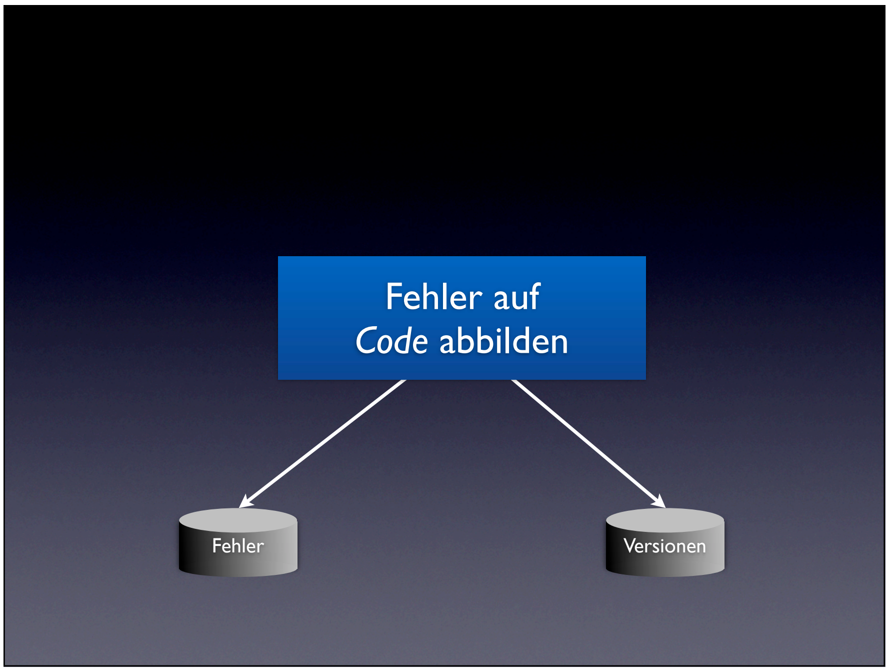
Reporter: Andrzej Wasylkowski <elessar@gdnmail.net>
Assigned To: aspectj inbox <aspectj-inbox@eclipse.org>
QA Contact:
CC: andrew.clement@gmail.com

Attachments

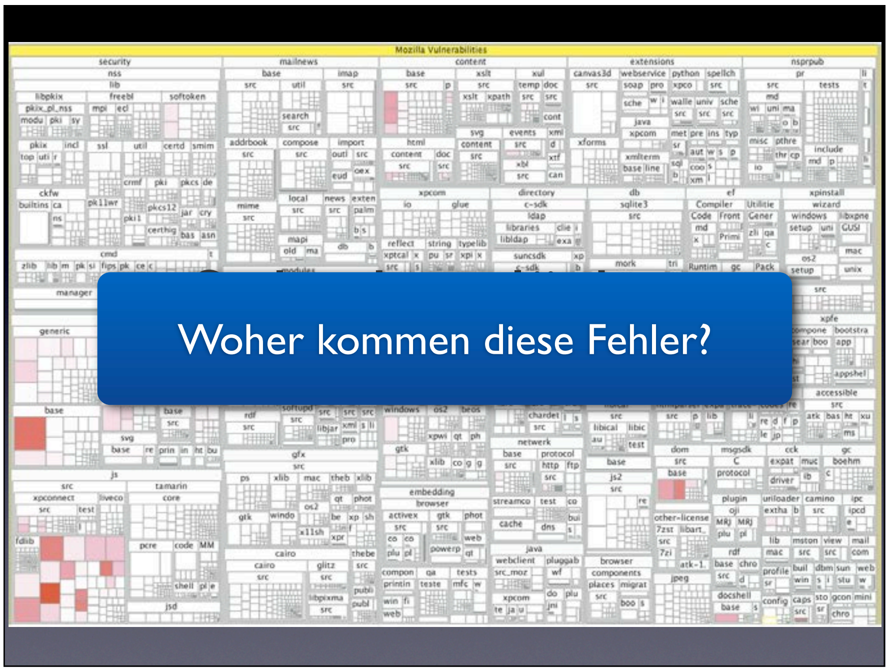
Add an attachment (proposed patch, testcase, etc.)



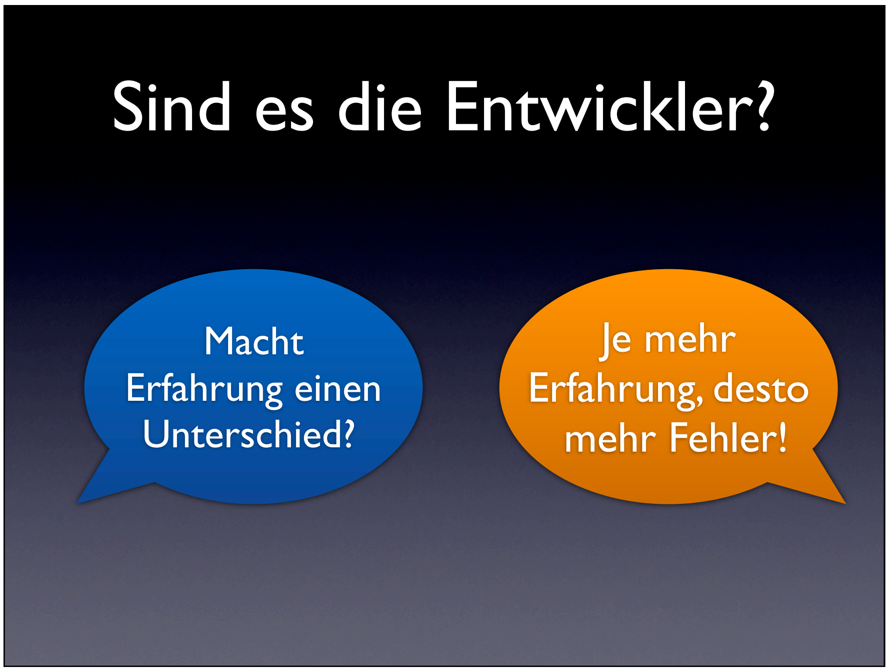
Such software archives are being used in practice all the time. If you file a bug, for instance, the report is stored in a bug database, and the resulting fix is stored in the version archive.



These databases can then be mined to extract interesting information. From bugs and changes, for instance, we can tell how many bugs were fixed in a particular location.

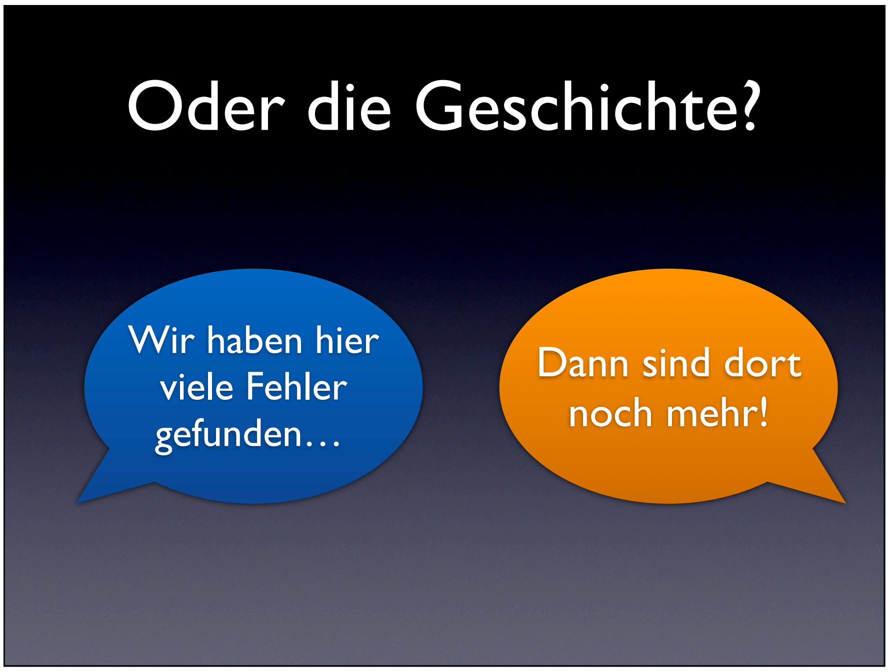


Woher kommen diese Fehler?



Sind es die Entwickler?

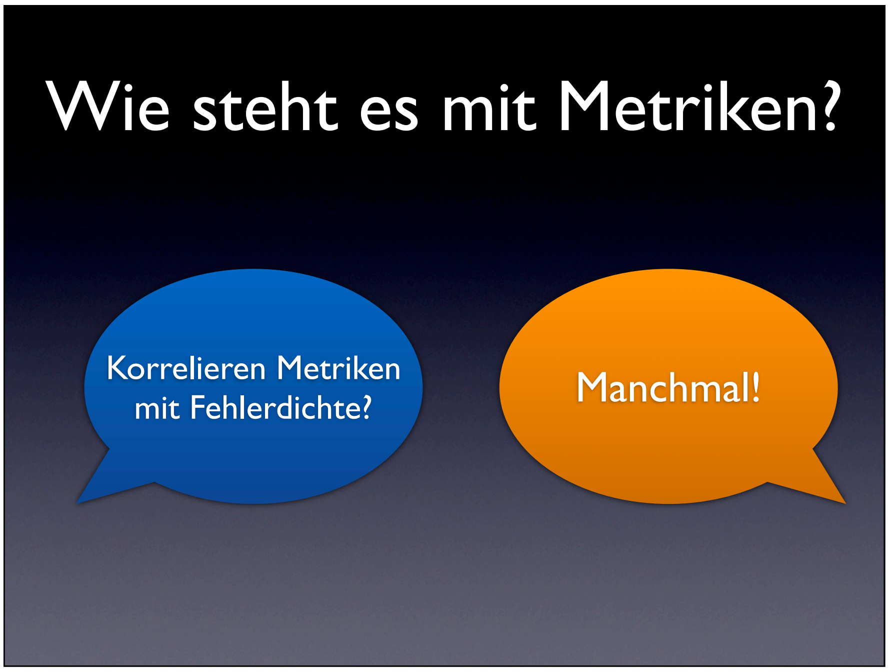
Macht Erfahrung einen Unterschied?
Je mehr Erfahrung, desto mehr Fehler!



Oder die Geschichte?

Wir haben hier viele Fehler gefunden…

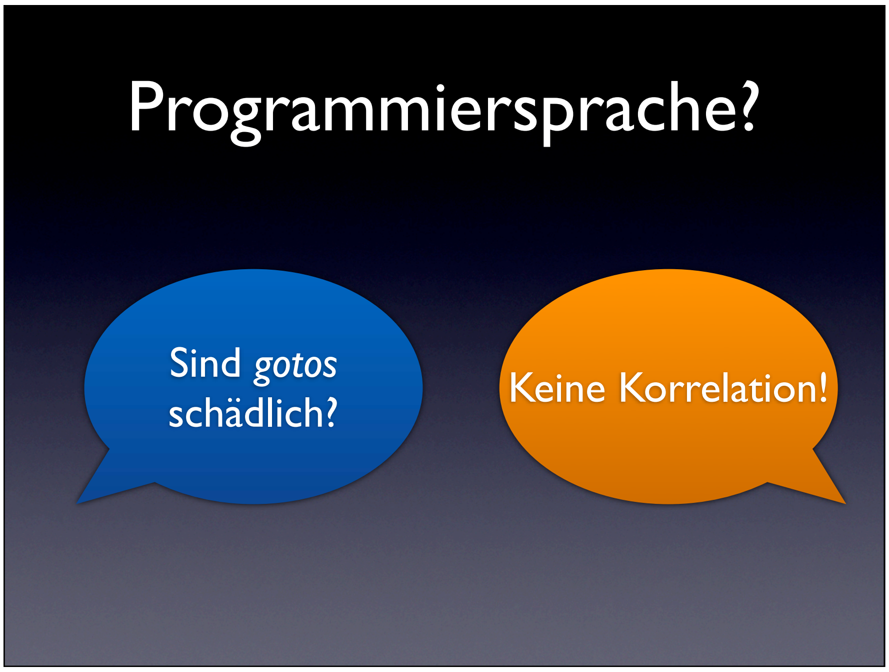
Dann sind dort noch mehr!



Wie steht es mit Metriken?

Korrelieren Metriken mit Fehlerdichte?

Manchmal!



Programmiersprache?

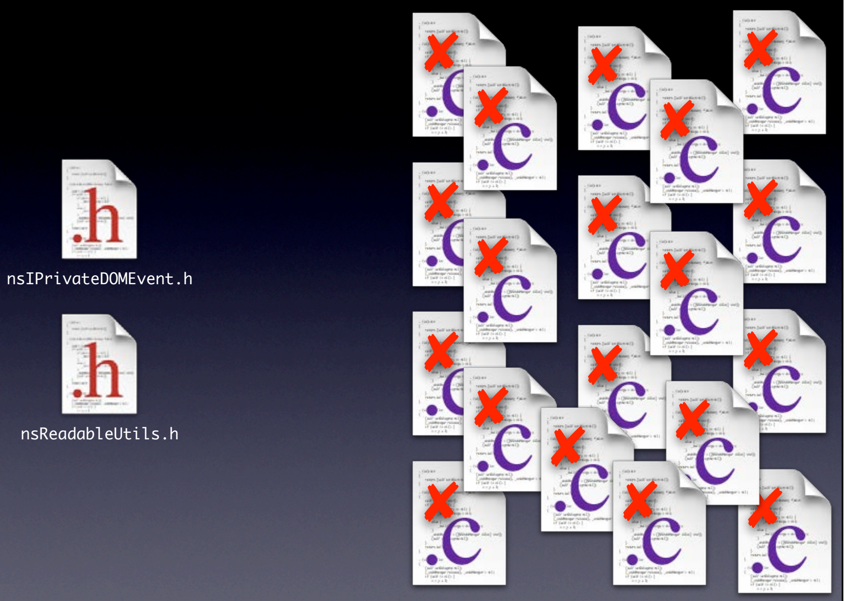
Sind *gotos* schädlich?

Keine Korrelation!

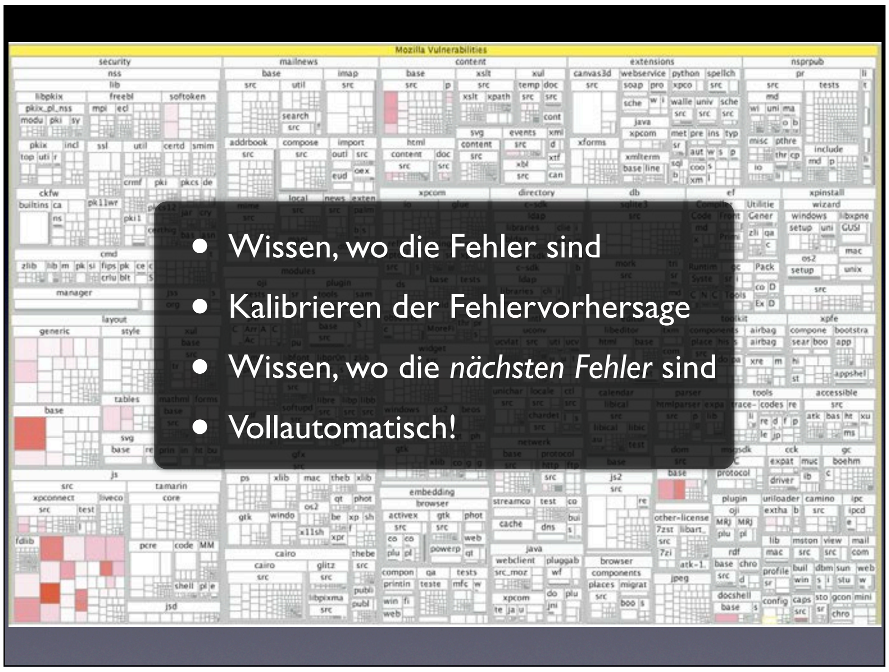
Ok. Problembereich?

Welche Tokens sind relevant?

import • extends • implements

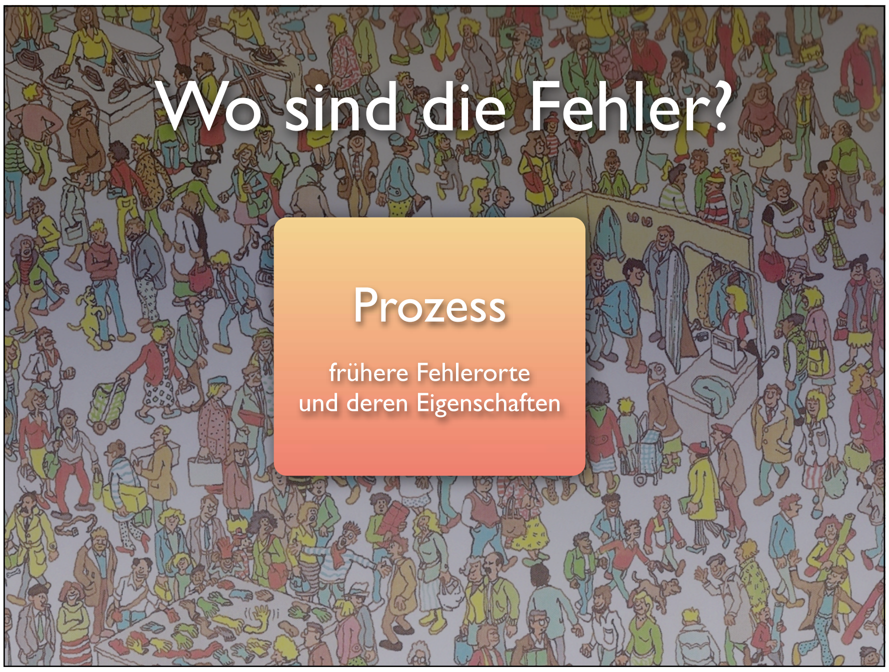


Vorhersage	Komponente	Tatsache
1	nsDOMClassInfo	3
2	SGridRowLayout	95
3	xpcprivate	6
4	jsxml	2
5	nsGenericHTMLElement	8
6	jsgc	3
7	nsISEnvironment	12
8	jsfun	1
9	nsHTMLLabelElement	18
10	nsHttpTransaction	35



- Wissen, wo die Fehler sind
- Kalibrieren der Fehlervorhersage
- Wissen, wo die *nächsten Fehler* sind
- Vollautomatisch!

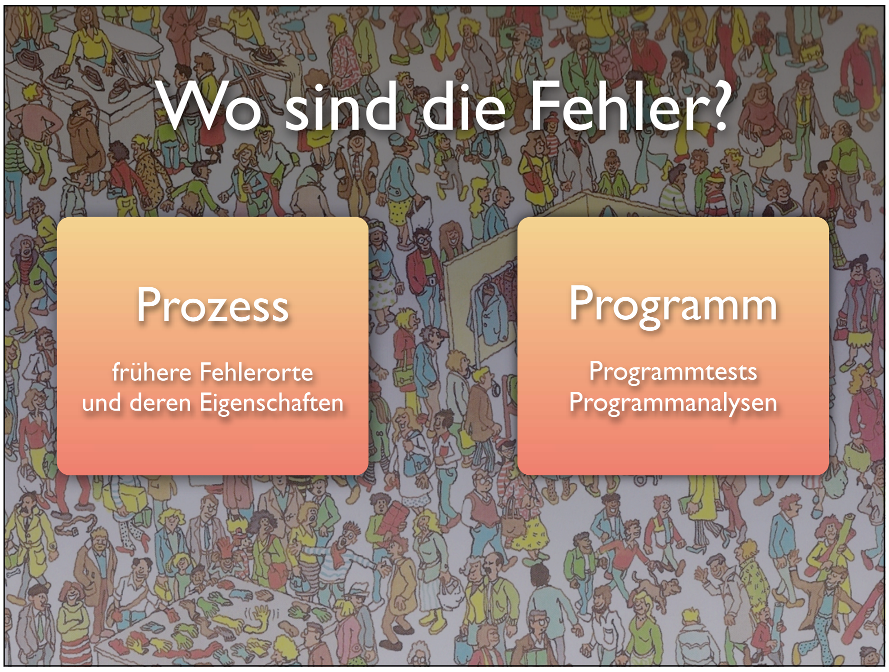
Sogar das heute journal hat einen Bericht gebracht – seitdem bin ich bekannt aus Funk und Fernsehen :-)



Wo sind die Fehler?

Prozess

frühere Fehlerorte
und deren Eigenschaften



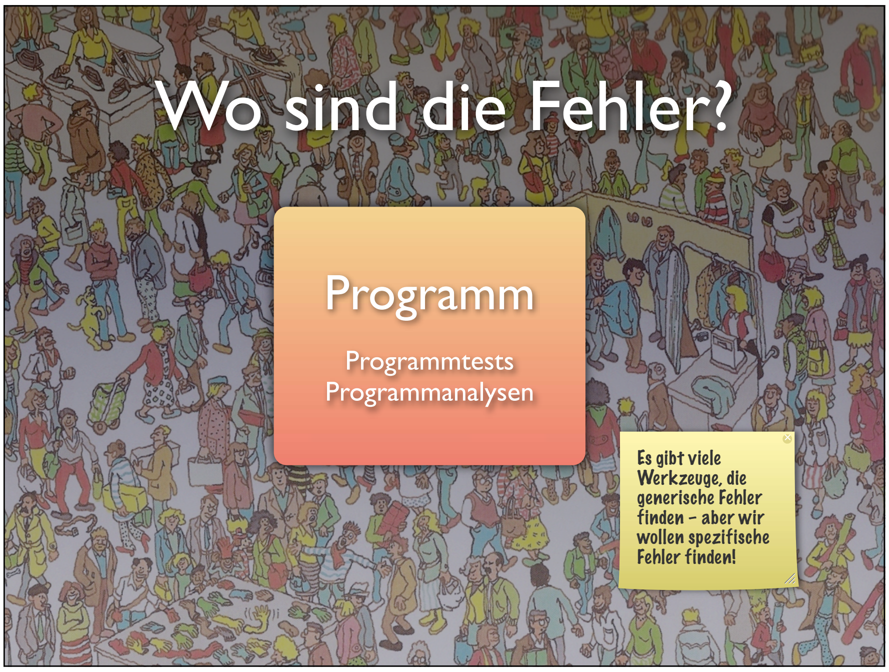
Wo sind die Fehler?

Prozess

frühere Fehlerorte
und deren Eigenschaften

Programm

Programmtests
Programmanalysen

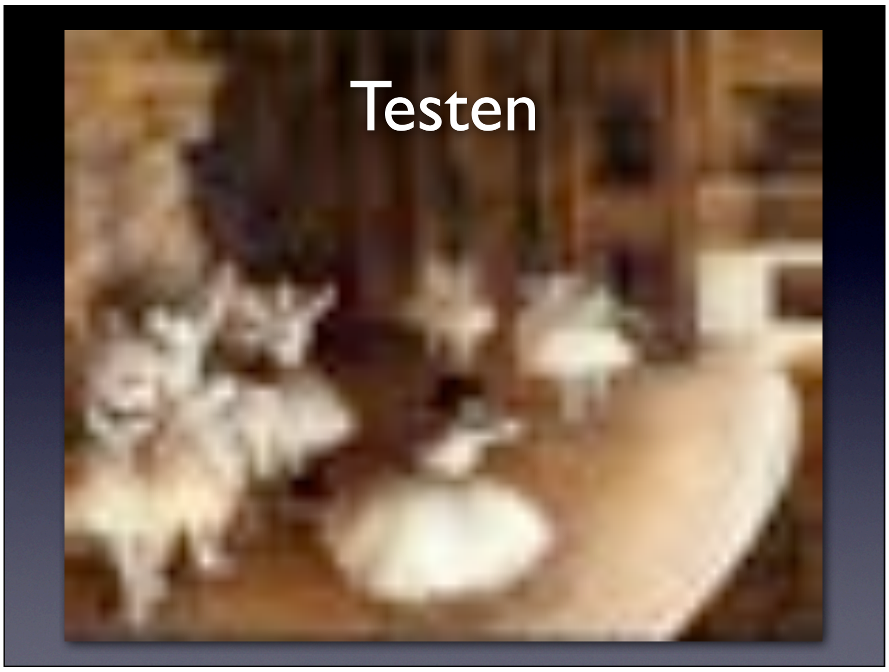


Wo sind die Fehler?

Programm

Programmtests
Programmanalysen

Es gibt viele Werkzeuge, die generische Fehler finden – aber wir wollen spezifische Fehler finden!



Testen

Edgar Degas: The Rehearsal. With a rehearsal, we want to check whether everything will work as expected. This is a test.

Mehr Testen

Again, a test. We test whether we can evacuate 500 people from an Airbus A380 in 90 seconds. This is a test.

Noch mehr Testen

And: We test whether a concrete wall (say, for a nuclear reactor) withstands a plane crash at 900 km/h. Indeed, it does.

Software ist vielfältig

We can also test software this way. But software is not a planned linear show – it has a multitude of possibilities. So: if it works once, will it work again? This is the central issue

Software ist vielfältig

We can also test software this way. But software is not a planned linear show – it has a multitude of possibilities. So: if it works once, will it work again? This

Software ist vielfältig

The problem is: There are many possible executions. And as the number grows…

Software ist vielfältig

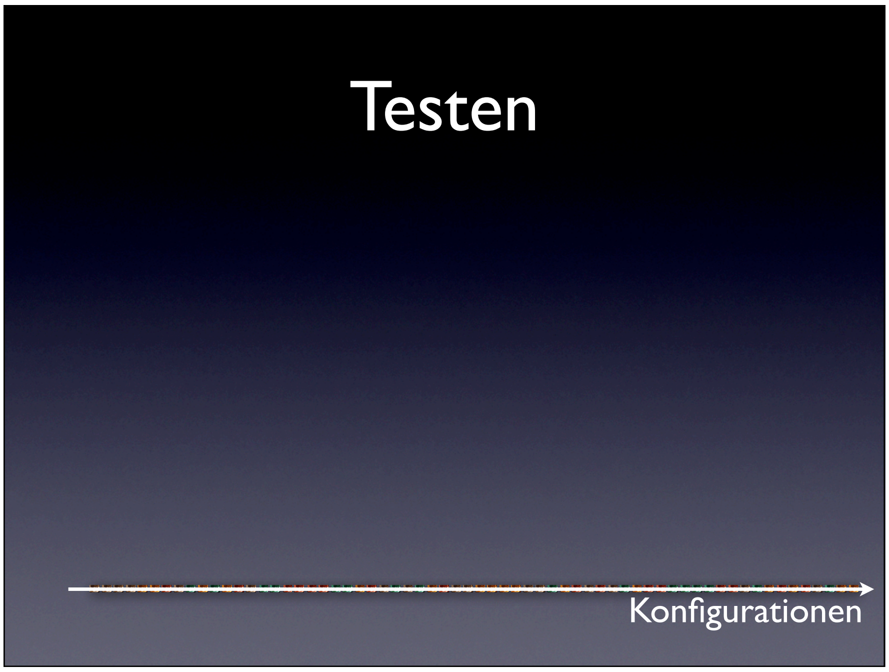
and grows…

Software ist vielfältig

and grows…

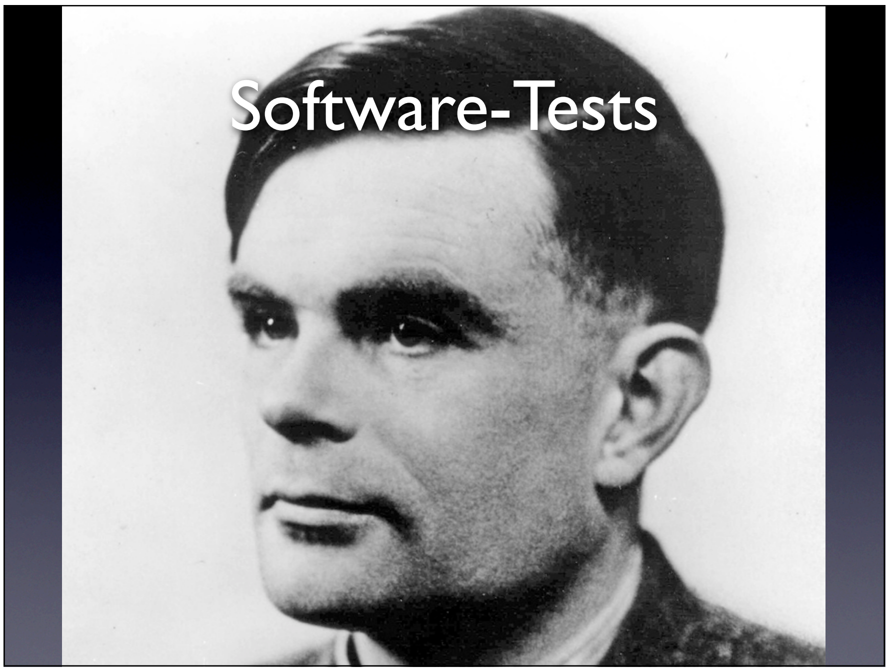
Software ist vielfältig

and grows…



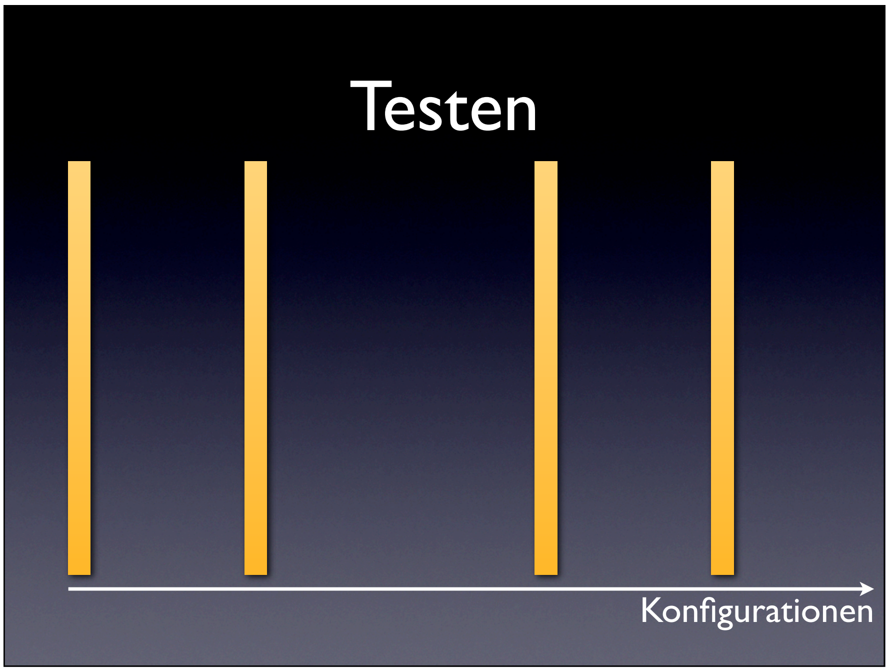
Testen

…you get an infinite number of possible executions, but you can only conduct a finite number of tests.



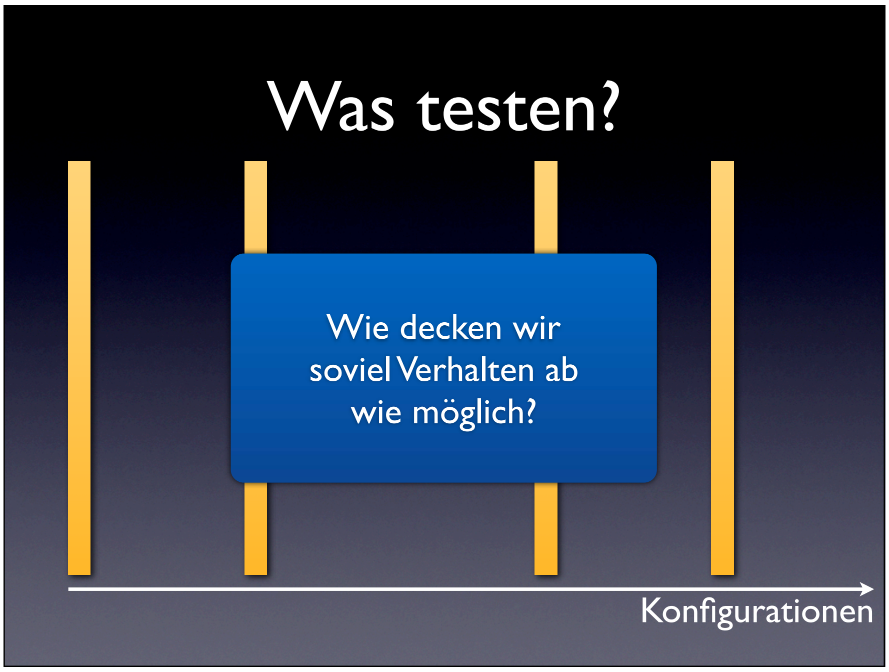
Software-Tests

…and this was something the first testers also needed to realize.



Testen

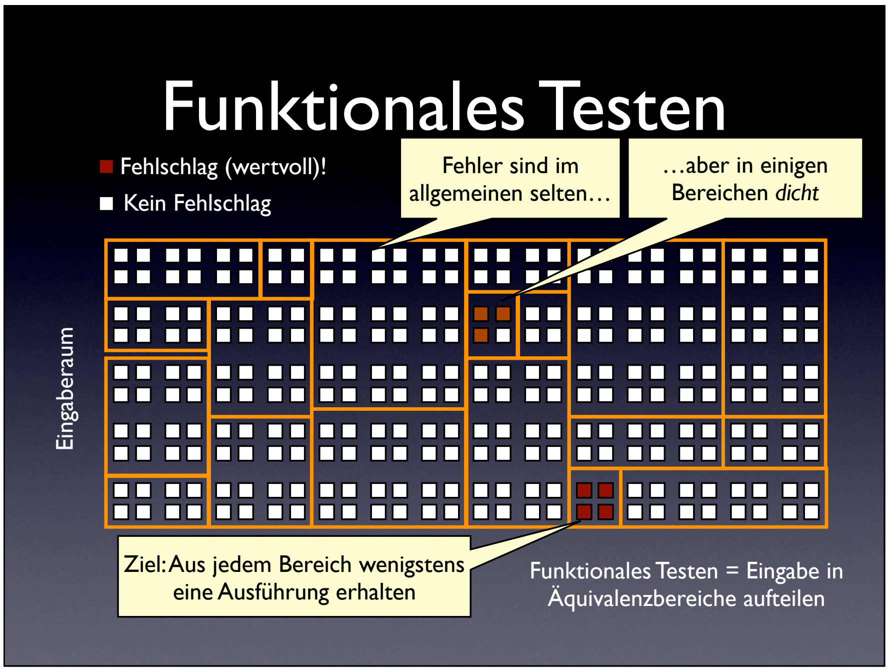
With testing, you pick a few of these configurations – and test them.



Was testen?

Wie decken wir soviel Verhalten ab wie möglich?

So, how can we cover as much behavior as possible?



Funktionales Testen

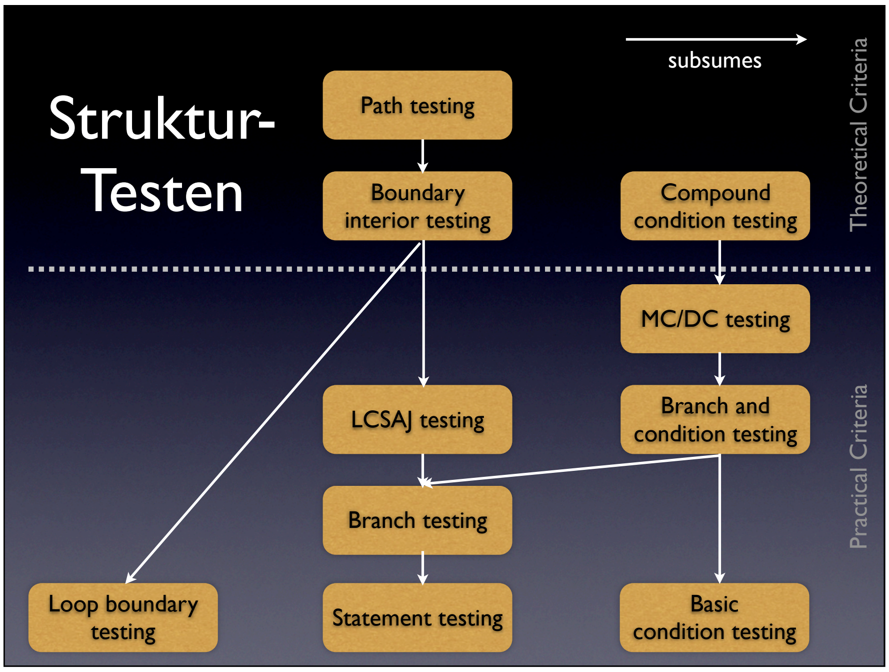
- 🟥 Fehlschlag (wertvoll)!
- ⬜ Kein Fehlschlag

Fehler sind im allgemeinen selten…

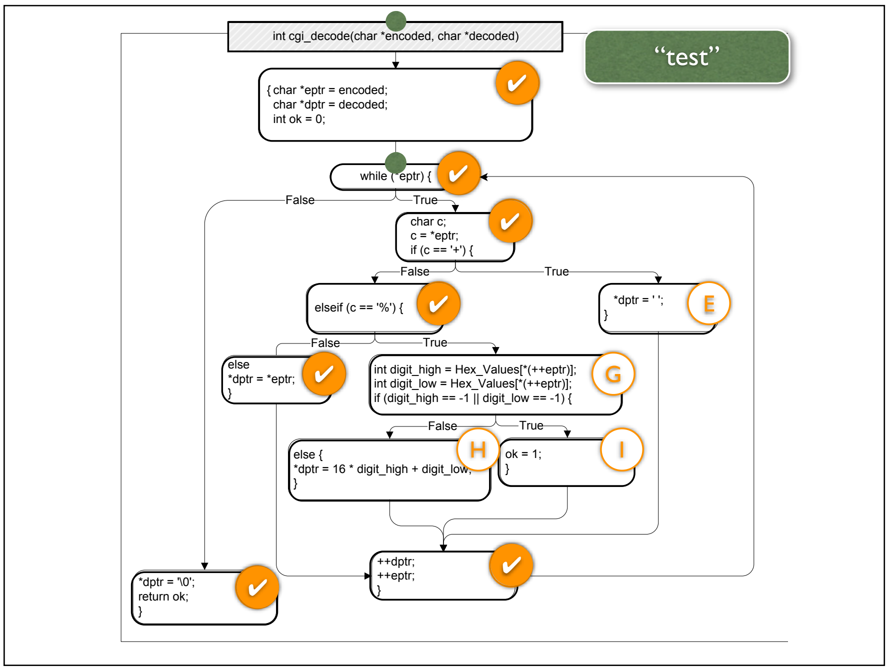
…aber in einigen Bereichen *dicht*

Ziel: Aus jedem Bereich wenigstens eine Ausführung erhalten

Funktionales Testen = Eingabe in Äquivalenzbereiche aufteilen

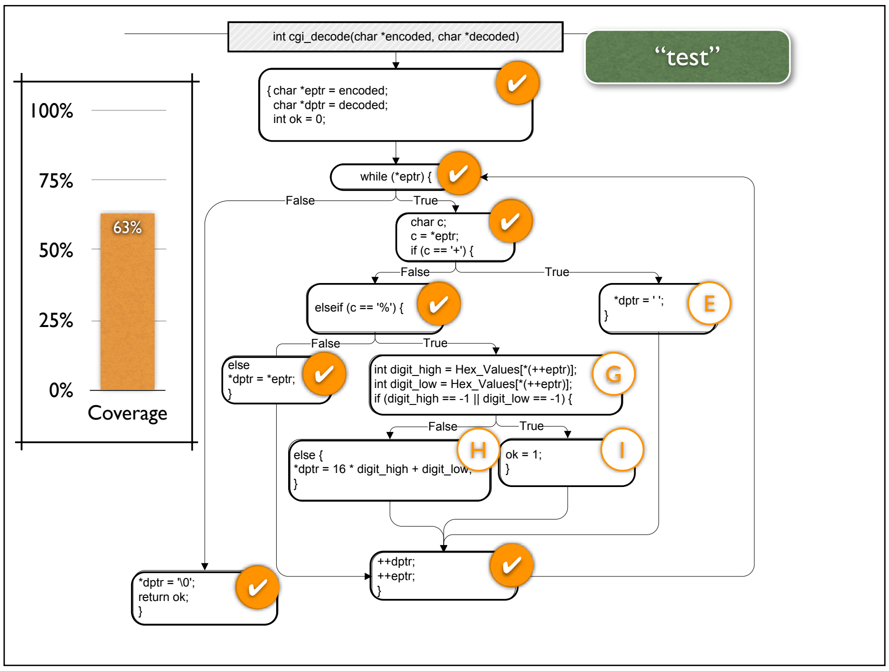


Struktur-Testen



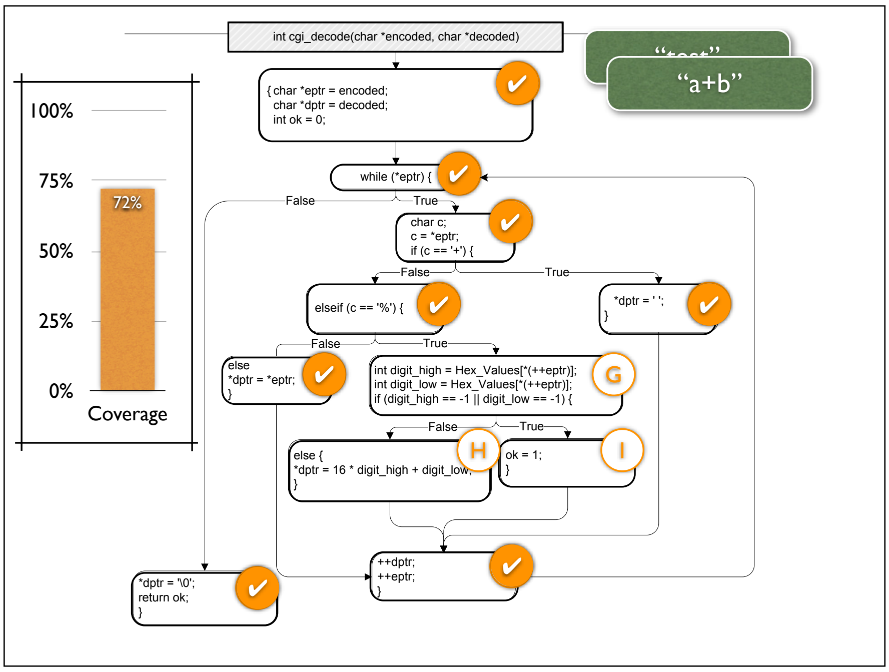
```
int cgi_decode(char *encoded, char *decoded)

{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;

while (*eptr) {

char c;
c = *eptr;
if (c == '+') {

elseif (c == '%') {

*dptr = ' ';
}

else
*dptr = *eptr;
}

int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {

else {
*dptr = 16 * digit_high + digit_low;
}

ok = 1;
}

++dptr;
++eptr;
}

*dptr = '\0';
return ok;
}
```

While the program is executed, one statement (or basic block) after the other is covered – i.e., executed at least once – but not all of them. Here, the input is "test"; checkmarks indicate executed blocks.

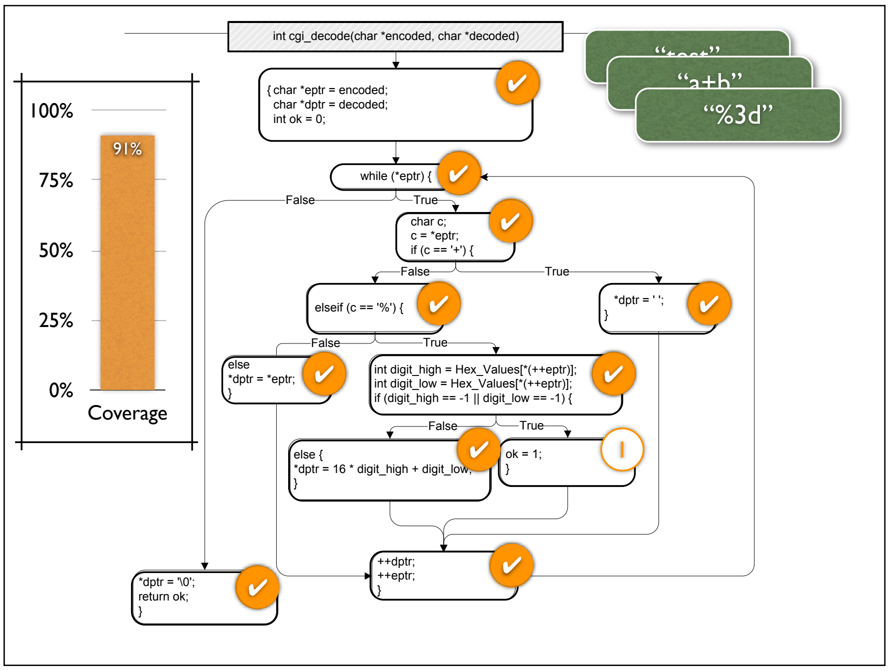


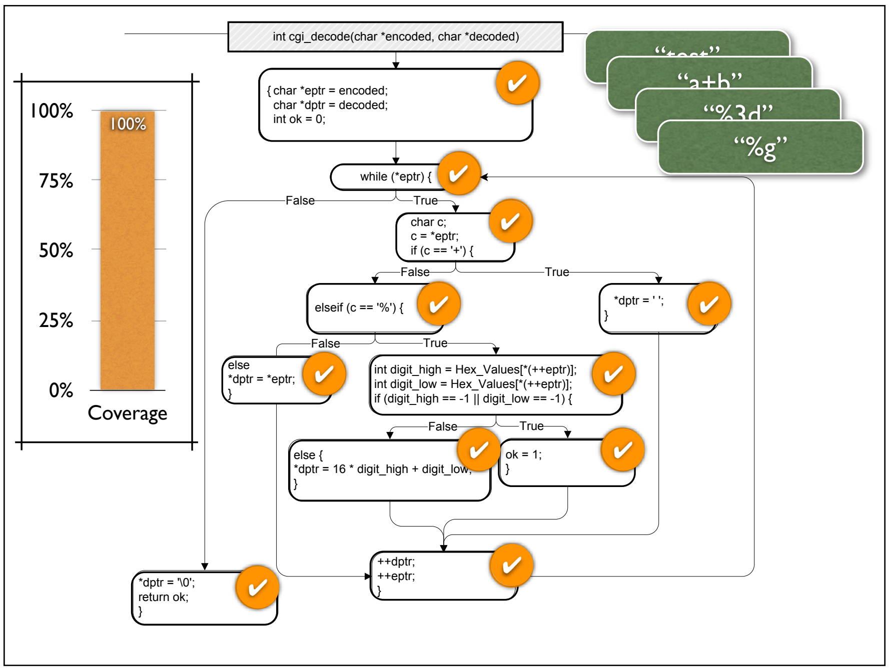
```
{ char *eptr = encoded;
char *dptr = decoded;
int ok = 0;

while (*eptr) {

char c;
c = *eptr;
if (c == '+') {

*dptr = ' ';
}

elseif (c == '%') {

else
*dptr = *eptr;
}

int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {

else {
*dptr = 16 * digit_high + digit_low;
}

ok = 1;
}

++dptr;
++eptr;
}

*dptr = '\0';
return ok;
}
```

The initial coverage is 7/11 blocks = 63%. We could also count the statements instead (here: 14/20 = 70%), but conceptually, this makes no difference.



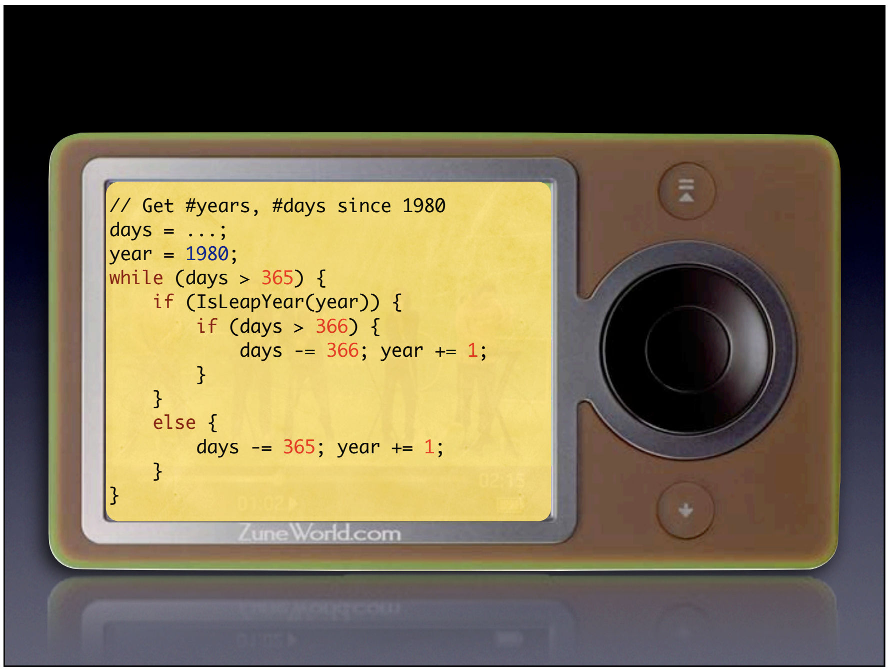
and the coverage increases with each additionally executed statement…





```
int cgi_decode(char *encoded, char *decoded)
{ char *eptr = encoded;
  char *dptr = decoded;
  int ok = 0;
while (*eptr) {
char c;
c = *eptr;
if (c == '+') {
elseif (c == '%') {
*dptr = ' ';
}
else
*dptr = *eptr;
}
int digit_high = Hex_Values[*(++eptr)];
int digit_low = Hex_Values[*(++eptr)];
if (digit_high == -1 || digit_low == -1) {
else {
*dptr = 16 * digit_high + digit_low;
}
ok = 1;
}
++dptr;
++eptr;
}
*dptr = '\0';
return ok;
}
```

… until we reach 100% block coverage (which is 100% statement coverage, too).



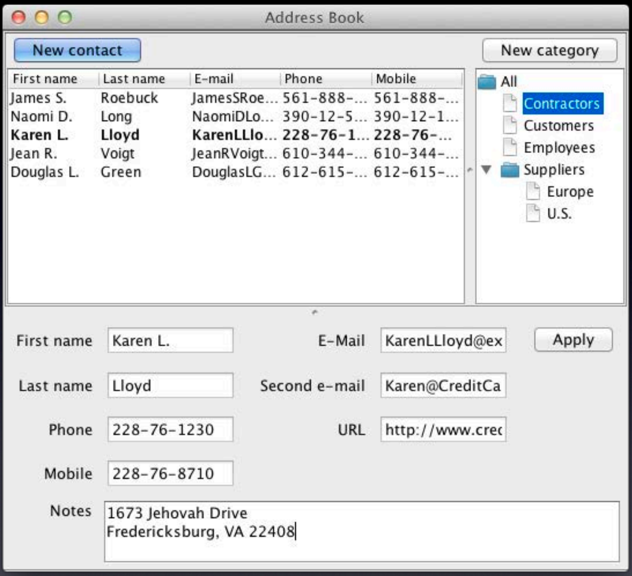
```
// Get #years, #days since 1980
days = ...;
year = 1980;
while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366; year += 1;
        }
    }
    else {
        days -= 365; year += 1;
    }
}
```

http://www.aeroxp.org/2009/01/lesson-on-infinite-loops/
http://www.youtube.com/watch?v=fYTJ9v2vsaE

All these techniques attempt to find the needle in the haystack…

Automatisierung

- Automatisches Ausführen
- Automatisches Generieren
- Automatisches Prüfen



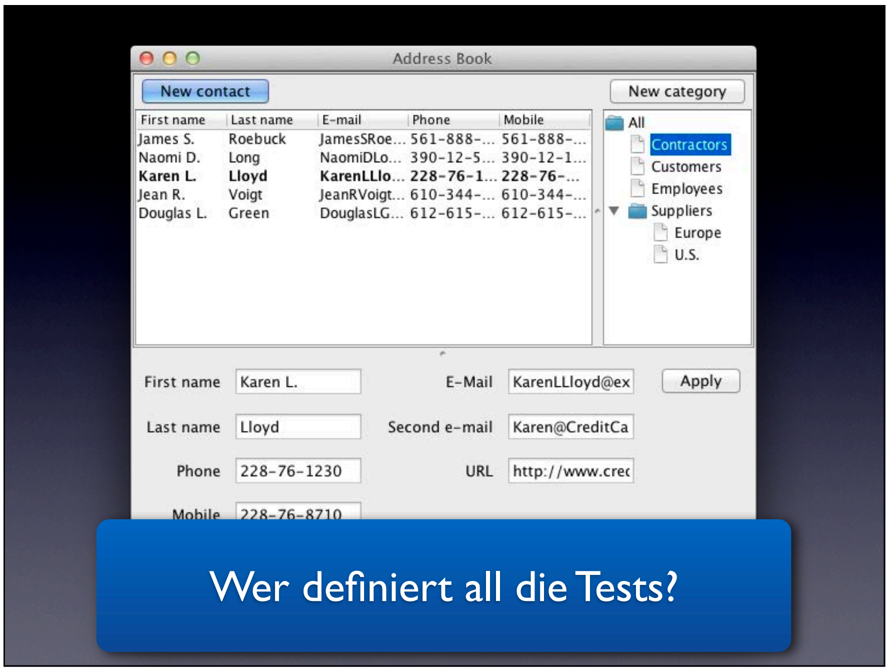
First name	Last name	E-mail	Phone	Mobile
James S.	Roebuck	JamesSRoe…	561-888-…	561-888-…
Naomi D.	Long	NaomiDLo…	390-12-5…	390-12-1…
Karen L.	**Lloyd**	**KarenLLlo…**	**228-76-1…**	**228-76-…**
Jean R.	Voigt	JeanRVoigt…	610-344-…	610-344-…
Douglas L.	Green	DouglasLG…	612-615-…	612-615-…

First name	Karen L.
Last name	Lloyd
Phone	228-76-1230
Mobile	228-76-8710
E-Mail	KarenLLloyd@ex
Second e-mail	Karen@CreditCa
URL	http://www.crec
Notes	1673 Jehovah Drive Fredericksburg, VA 22408

As an example, here’s the Addressbook program: a simple Java application which manages a set of contacts that can be entered, searched, and grouped into

Capture + Replay

- Wir können Tastatur und Maus *aufzeichnen*
- …und nach Belieben wieder *abspielen*!



Address Book

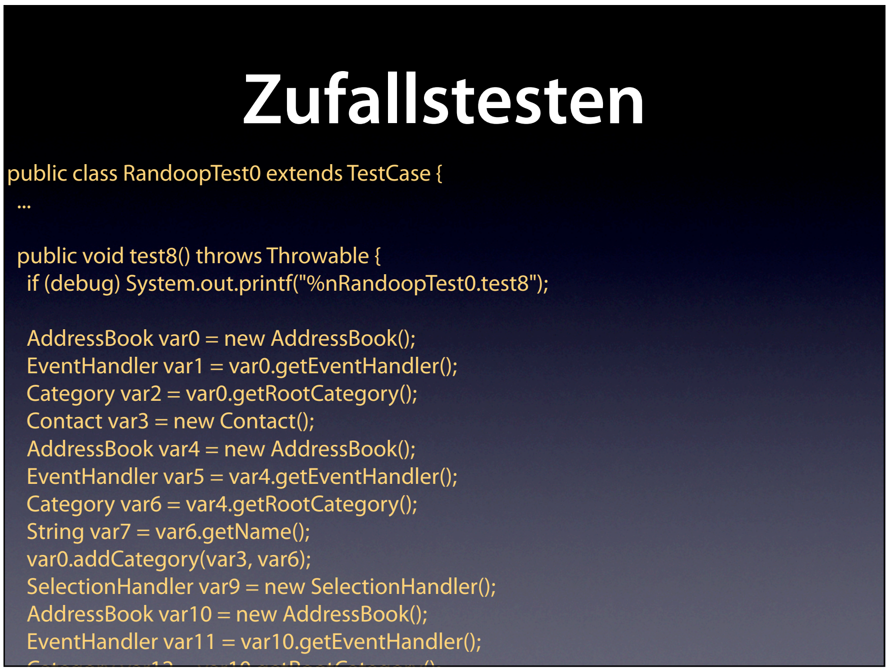
New contact | New category

First name	Last name	E-mail	Phone	Mobile
James S.	Roebuck	JamesSRoe...	561-888-...	561-888-...
Naomi D.	Long	NaomiDLo...	390-12-5...	390-12-1...
Karen L.	**Lloyd**	**KarenLLlo...**	**228-76-1...**	**228-76-...**
Jean R.	Voigt	JeanRVoigt...	610-344-...	610-344-...
Douglas L.	Green	DouglasLG...	612-615-...	612-615-...

First name: Karen L.
E-Mail: KarenLLloyd@ex
Apply
Last name: Lloyd
Second e-mail: Karen@CreditCa
Phone: 228-76-1230
URL: http://www.crec
Mobile: 228-76-8710

Wer definiert all die Tests?

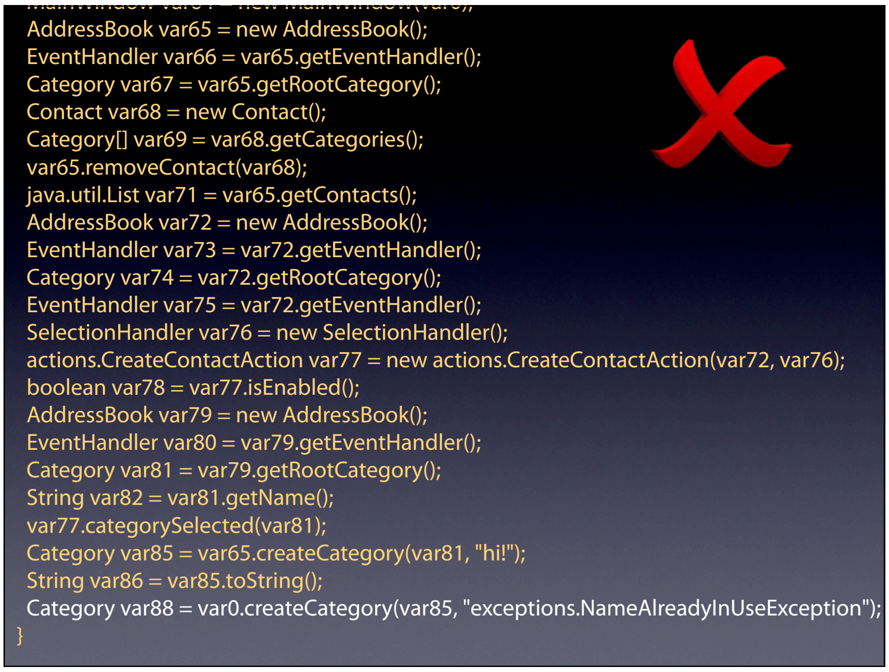
As an example, here's the Addressbook program: a simple Java application which manages a set of contacts that can be entered, searched, and grouped into



Zufallstesten

```
public class RandoopTest0 extends TestCase {
 ...

 public void test8() throws Throwable {
  if (debug) System.out.printf("%nRandoopTest0.test8");

  AddressBook var0 = new AddressBook();
  EventHandler var1 = var0.getEventHandler();
  Category var2 = var0.getRootCategory();
  Contact var3 = new Contact();
  AddressBook var4 = new AddressBook();
  EventHandler var5 = var4.getEventHandler();
  Category var6 = var4.getRootCategory();
  String var7 = var6.getName();
  var0.addCategory(var3, var6);
  SelectionHandler var9 = new SelectionHandler();
  AddressBook var10 = new AddressBook();
  EventHandler var11 = var10.getEventHandler();
```

Here's a test case generated by Randoop. It's >200 lines long…

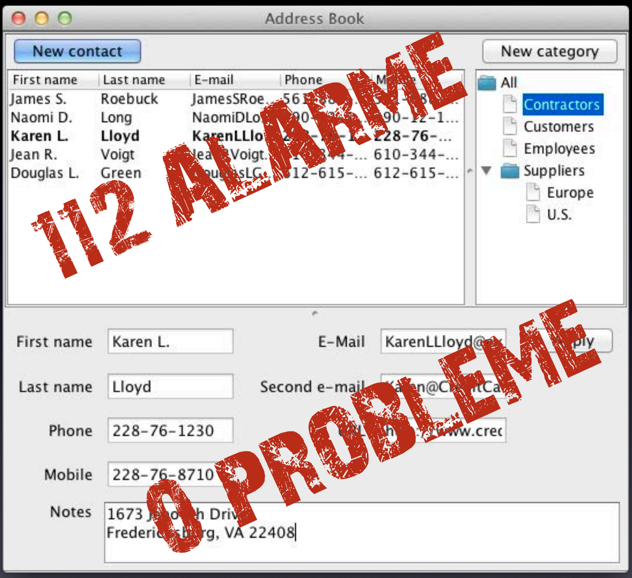


```
  AddressBook var65 = new AddressBook();
  EventHandler var66 = var65.getEventHandler();
  Category var67 = var65.getRootCategory();
  Contact var68 = new Contact();
  Category[] var69 = var68.getCategories();
  var65.removeContact(var68);
  java.util.List var71 = var65.getContacts();
  AddressBook var72 = new AddressBook();
  EventHandler var73 = var72.getEventHandler();
  Category var74 = var72.getRootCategory();
  EventHandler var75 = var72.getEventHandler();
  SelectionHandler var76 = new SelectionHandler();
  actions.CreateContactAction var77 = new actions.CreateContactAction(var72, var76);
  boolean var78 = var77.isEnabled();
  AddressBook var79 = new AddressBook();
  EventHandler var80 = var79.getEventHandler();
  Category var81 = var79.getRootCategory();
  String var82 = var81.getName();
  var77.categorySelected(var81);
  Category var85 = var65.createCategory(var81, "hi!");
  String var86 = var85.toString();
  Category var88 = var0.createCategory(var85, "exceptions.NameAlreadyInUseException");
 }
```

… and in the end, it fails. What do you do now?

Zufallstesten

- Einfach zu realisieren…
- …aber erzeugt viele unsinnige Tests!



First name	Last name	E-mail	Phone	Mobile
James S.	Roebuck	JamesSRoe…	561[illegible]	[illegible]
Naomi D.	Long	NaomiDLo[illegible]	[illegible]	290-12-1…
Karen L.	**Lloyd**	**KarenLLlo[illegible]**	[illegible]	**228-76-…**
Jean R.	Voigt	Jea[illegible]Voigt…	[illegible]	610-344-…
Douglas L.	Green	[illegible]sLG…	612-615-…	612-615-…

First name	Karen L.
Last name	Lloyd
Phone	228-76-1230
Mobile	228-76-8710
Notes	1673 J[illegible]h Driv[illegible] Frederi[illegible]rg, VA 22408
E-Mail	KarenLLloyd@[illegible]
Second e-mail	[illegible]n@C[illegible]tCa
[illegible]	[illegible]www.cre[illegible]

The catch is: There's never more than one addressbook! So the Randoop test makes little sense, because it violates an implicit precondition. When testing the Addressbook classes, Randoop detects * 112 failures. However, all of them are false, pointing to an error in the generated test case rather than the application itself, which has *0 problems.

Ein Fehlalarm

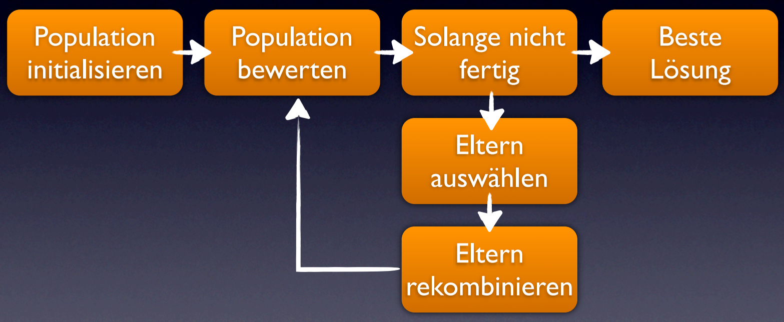
```
public class RandoopTest0 extends TestCase {
  public void test8() throws Throwable {
    if (debug) System.out.printf("%nRandoopTest0.test8");

    AddressBook a1 = new AddressBook();
    AddressBook a2 = new AddressBook();
    Category a1c = a1.createCategory(a1.getRootCategory(), "a1c");
    Category a2c = a2.createCategory(a1c, "a2c");
  }
}
```

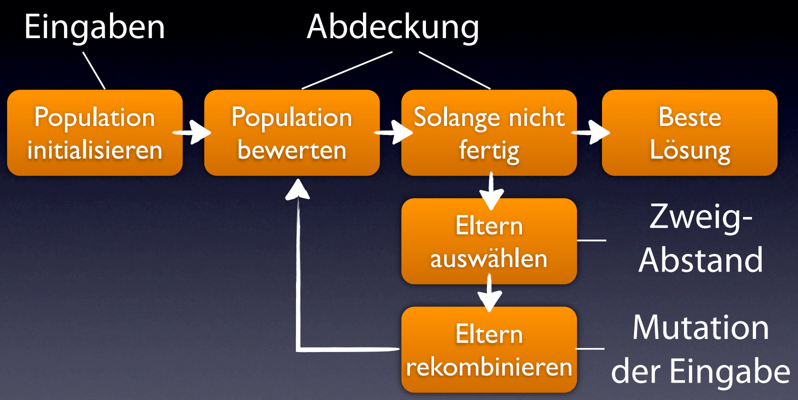
A simplified version of the above. If you use two address book objects and make one's category depend on one the other, it'll crash.

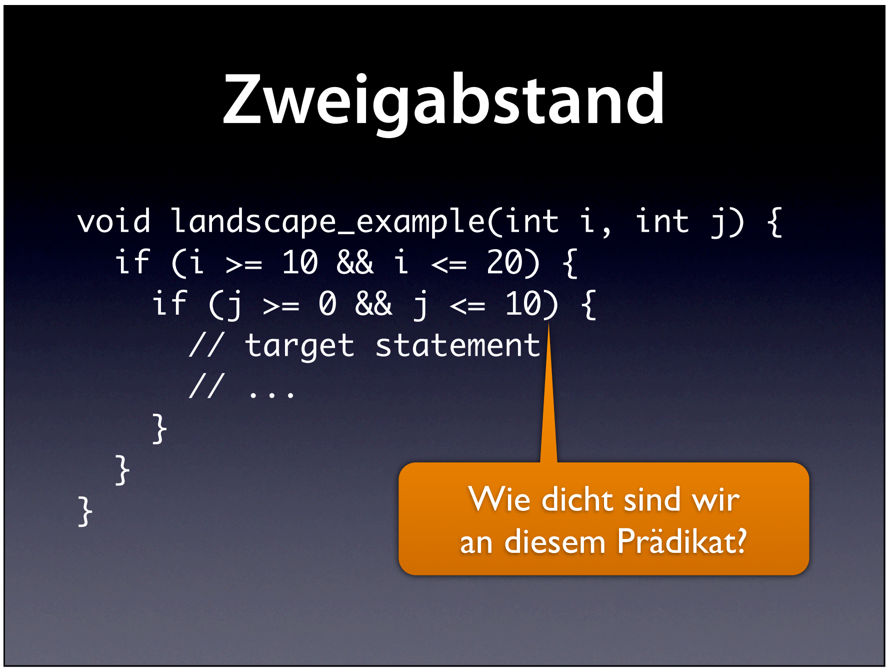
System-Tests

- Erzeuge Tests für die Bedienoberfläche
  - Jede Eingabe ist korrekt
  - Keine Fehlalarme

Genetische Algorithmen



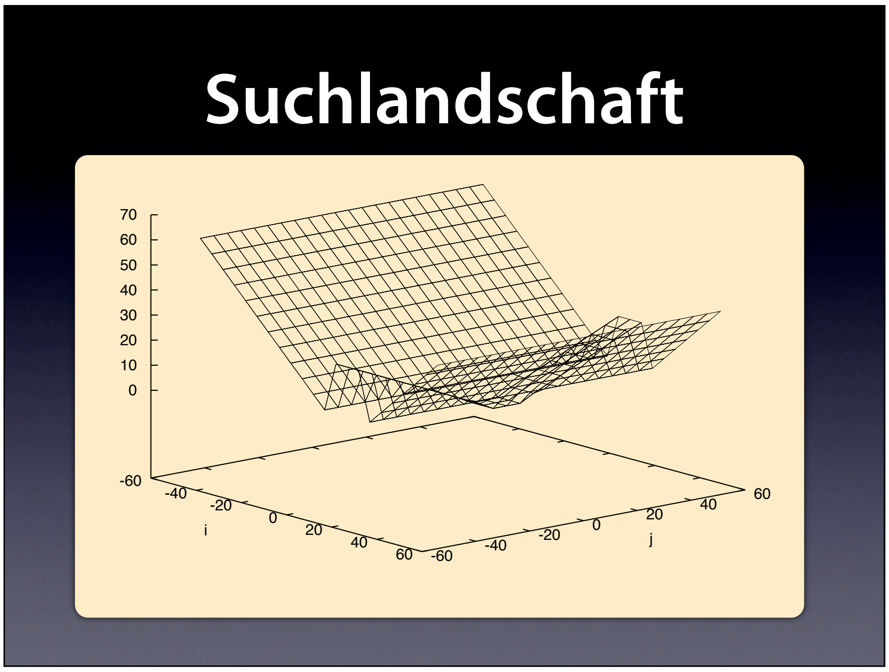




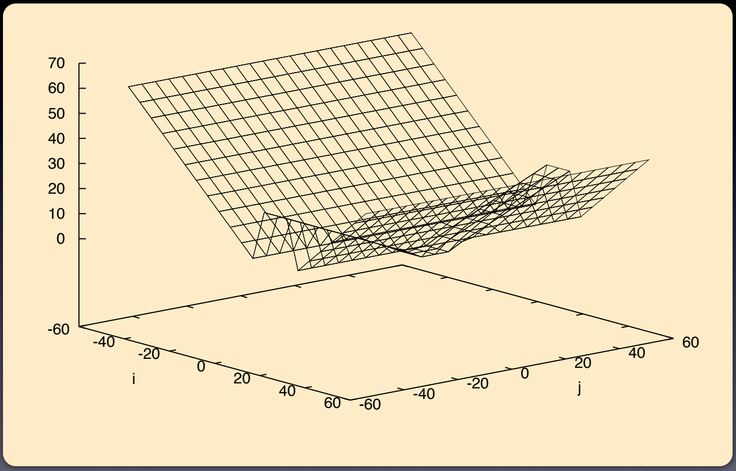
Zweigabstand

```
void landscape_example(int i, int j) {
  if (i >= 10 && i <= 20) {
    if (j >= 0 && j <= 10) {
      // target statement
      // ...
    }
  }
}
```

Wie dicht sind wir an diesem Prädikat?

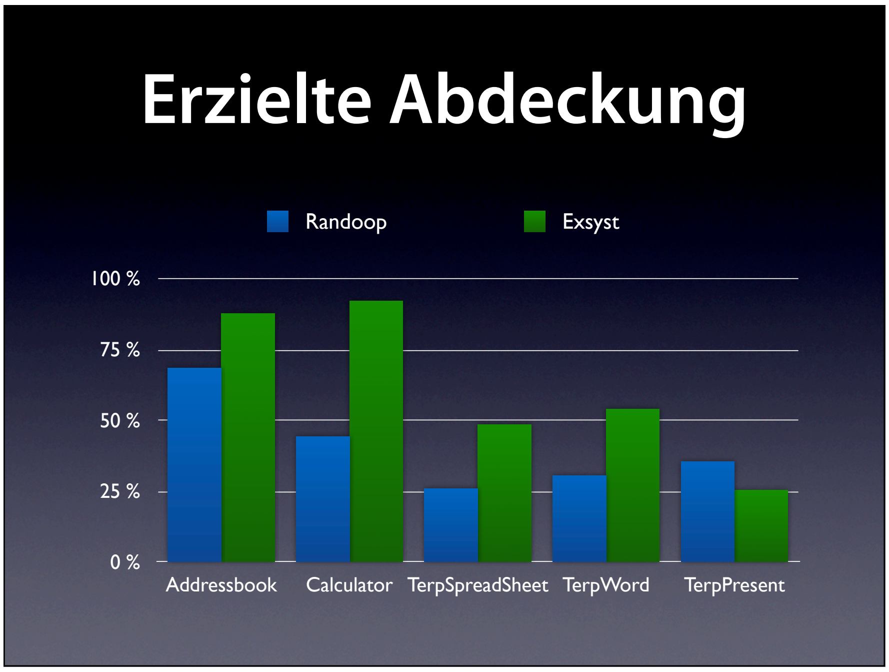


Suchlandschaft

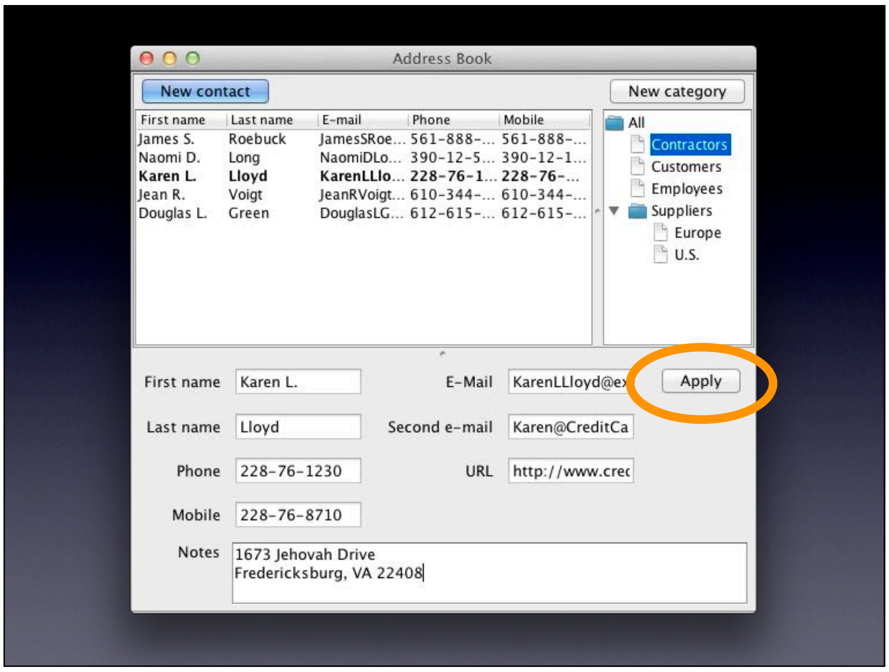


What I'm going to demo you now is our prototype called EXSYST, for Explorative SYStem Testing. EXSYST takes a Java program with a graphical user interface, such as our Addressbook example. It then generates user inputs such as mouse clicks or keystrokes and feeds them into the program. What you see here is EXSYST clicking and typing into the address book program; at the top, you see the statement coverage achieved so far. (Normally, all of this takes place in the background, so you don't see it, and it is also much much faster).

At first, these inputs are completely random, as you can see in these initial



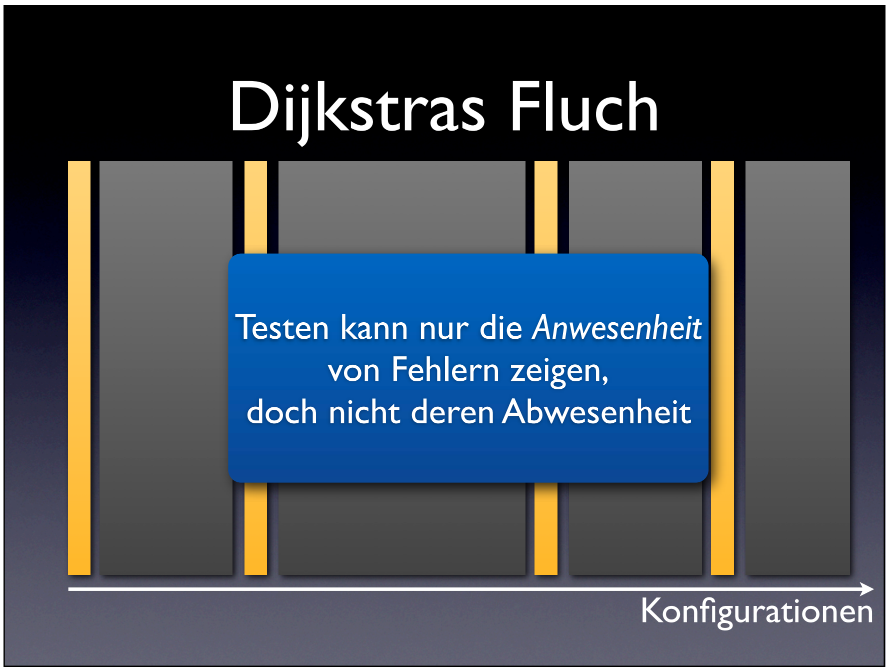
The results are clear. Although it's going through the GUI, EXSYST achieves a far higher coverage than Randoop. Here are the results for * Addressbook and *** three more



First name	Last name	E-mail	Phone	Mobile
James S.	Roebuck	JamesSRoe...	561-888-...	561-888-...
Naomi D.	Long	NaomiDLo...	390-12-5...	390-12-1...
Karen L.	**Lloyd**	**KarenLLlo...**	**228-76-1...**	**228-76-...**
Jean R.	Voigt	JeanRVoigt...	610-344-...	610-344-...
Douglas L.	Green	DouglasLG...	612-615-...	612-615-...

First name	Karen L.	E-Mail	KarenLLloyd@ex
Last name	Lloyd	Second e-mail	Karen@CreditCa
Phone	228-76-1230	URL	http://www.crec
Mobile	228-76-8710		
Notes	1673 Jehovah Drive Fredericksburg, VA 22408		

EXSYST found failures in all five programs which can be invoked with a few simple inputs. In AddressBook, for instance, if you press the Apply button without


Dijkstras Fluch
Testen kann nur die *Anwesenheit* von Fehlern zeigen, doch nicht deren Abwesenheit
Konfigurationen

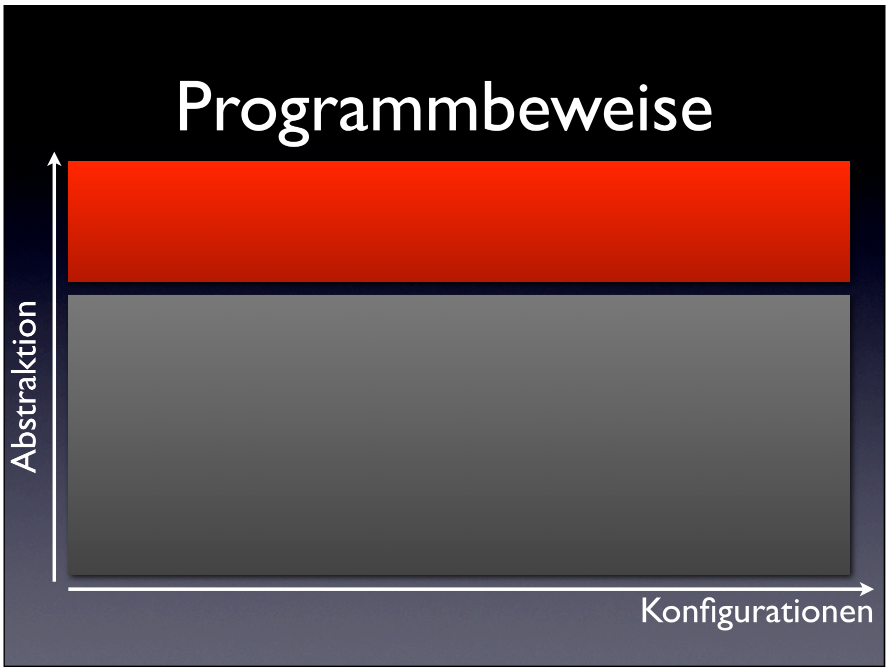
But still, testing suffers from what I call Dijkstra's curse – a double meaning, as it applies both to testing as to his famous quote. Is there something that can find the

Programmbeweise

Konfigurationen

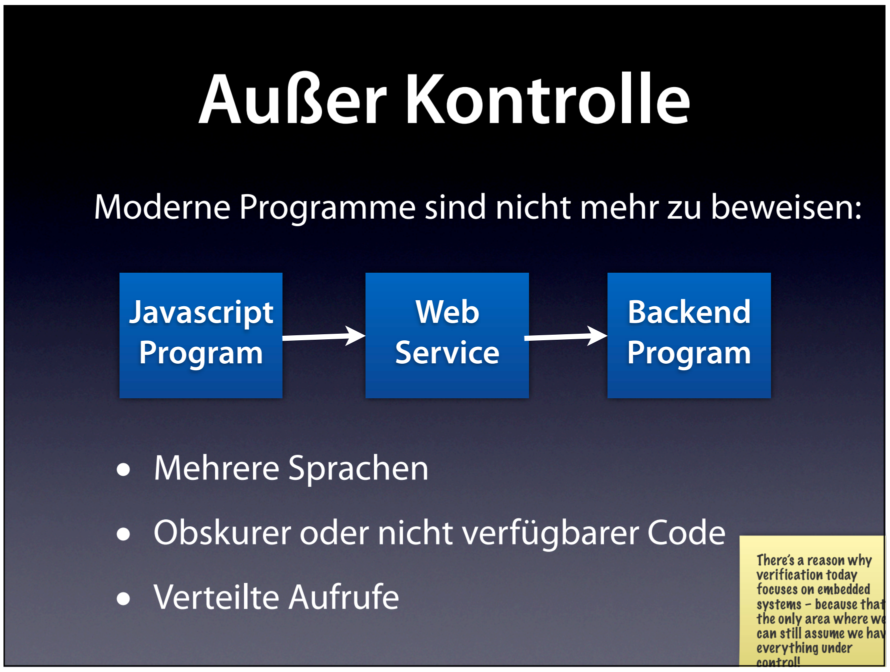
Programmbeweise

Abstraktion
Konfigurationen



Programmbeweise

Areas missing might be: the operating system, the hardware, all of the world the system is embedded in (including humans!)



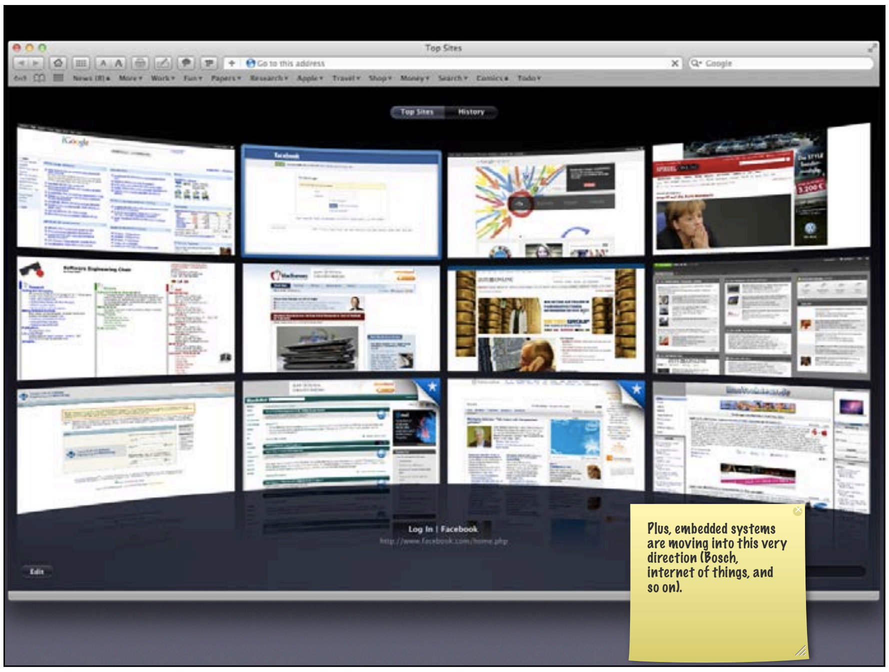
Außer Kontrolle

Moderne Programme sind nicht mehr zu beweisen:

$$\begin{tikzcd}
\boxed{\text{Javascript Program}} \arrow[r] & \boxed{\text{Web Service}} \arrow[r] & \boxed{\text{Backend Program}}
\end{tikzcd}$$

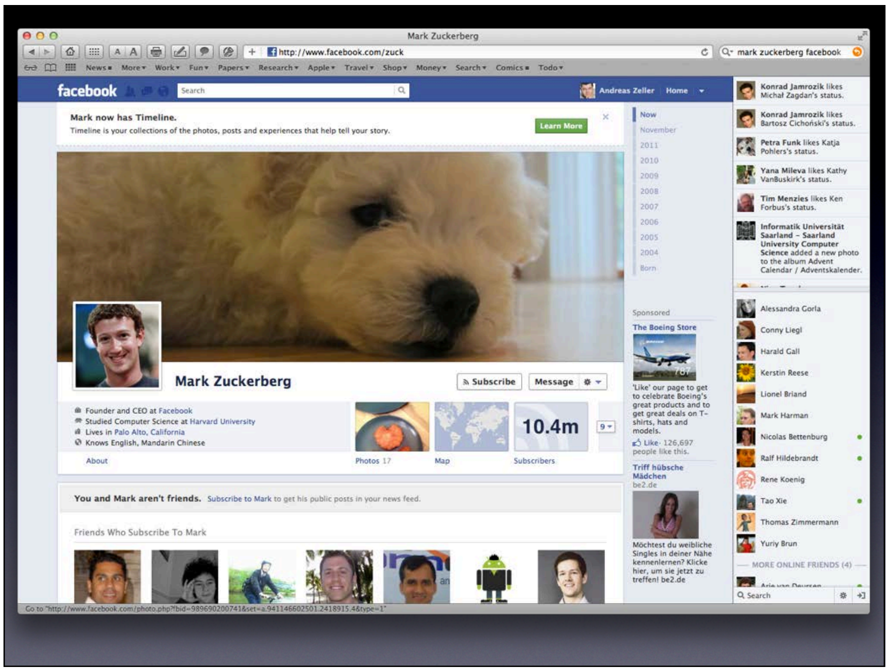
- Mehrere Sprachen
- Obskurer oder nicht verfügbarer Code
- Verteilte Aufrufe

There's a reason why verification today focuses on embedded systems – because that the only area where we can still assume we hav everything under control!

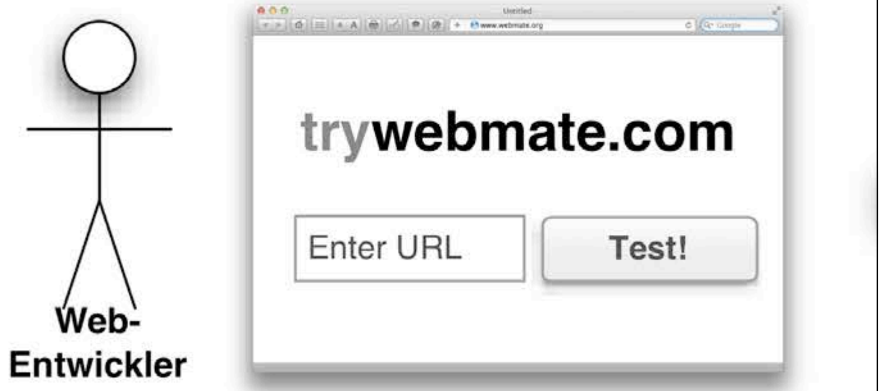


Plus, embedded systems are moving into this very direction (Bosch, internet of things, and so on).

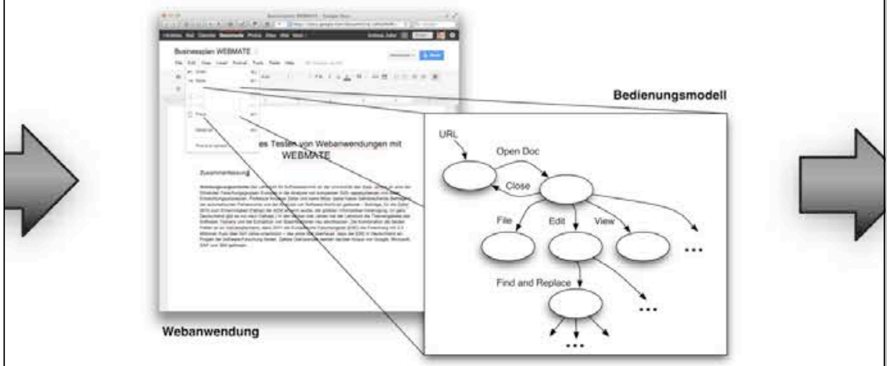
Well, everyone has. You start a browser, you have it all. None of this is what program analysis can handle these days. We're talking scripts, we're talking distributed, we're talking amateurs, we're talking security.


Mark Zuckerberg
http://www.facebook.com/zuck
mark zuckerberg facebook
facebook
Andreas Zeller
Home
Mark now has Timeline.
Timeline is your collection of the photos, posts and experiences that help tell your story.
Learn More
Mark Zuckerberg
Subscribe
Message
Founder and CEO at Facebook
Studied Computer Science at Harvard University
Lives in Palo Alto, California
Knows English, Mandarin Chinese
About
Photos 17
Map
10.4m
Subscribers
You and Mark aren't friends. Subscribe to Mark to get his public posts in your news feed.
Friends Who Subscribe To Mark
Now
November
2011
2010
2009
2008
2007
2006
2005
2004
Born
Sponsored
The Boeing Store
Triff hübsche Mädchen
Konrad Jamrozik likes Michał Zagdan's status.
Konrad Jamrozik likes Bartosz Cichoński's status.
Petra Funk likes Katja Pohlers's status.
Yana Mileva likes Kathy VanBuskirk's status.
Tim Menzies likes Ken Forbus's status.
Informatik Universität Saarland – Saarland University Computer Science added a new photo to the album Advent Calendar / Adventskalender.
Alessandra Gorla
Conny Liegl
Harald Gall
Kerstin Reese
Lionel Briand
Mark Harman
Nicolas Bettenburg
Ralf Hildebrandt
Rene Koenig
Tao Xie
Thomas Zimmermann
Yuriy Brun
MORE ONLINE FRIENDS (4)
Search

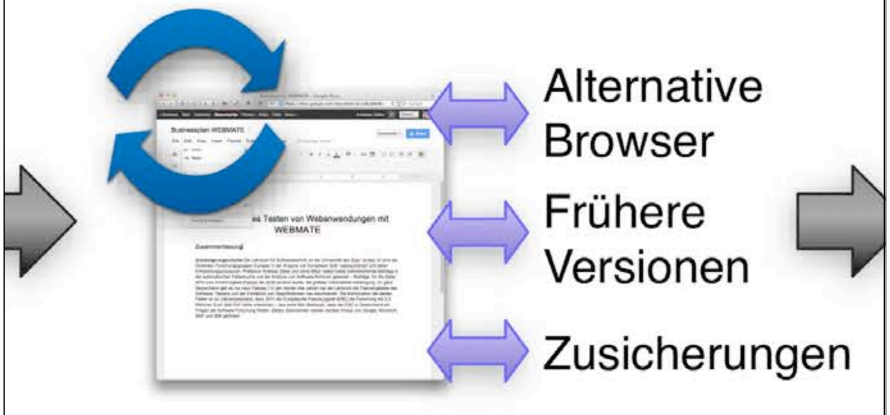
Webmate



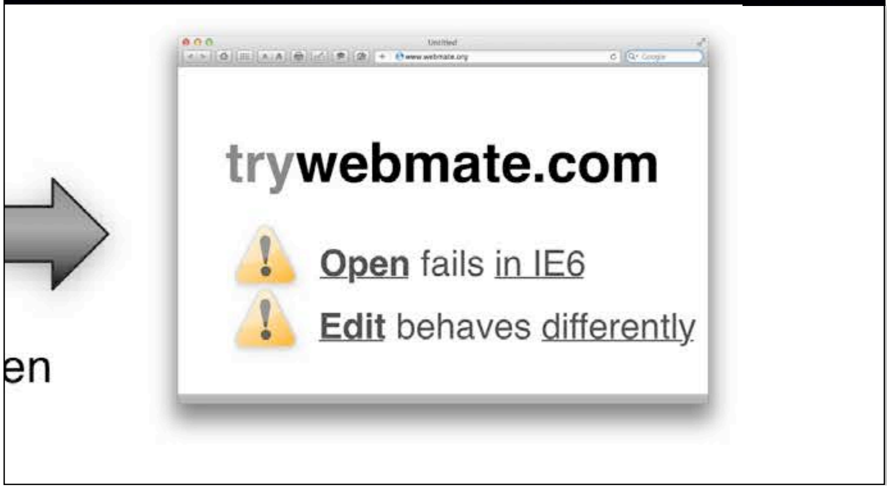
Webmate



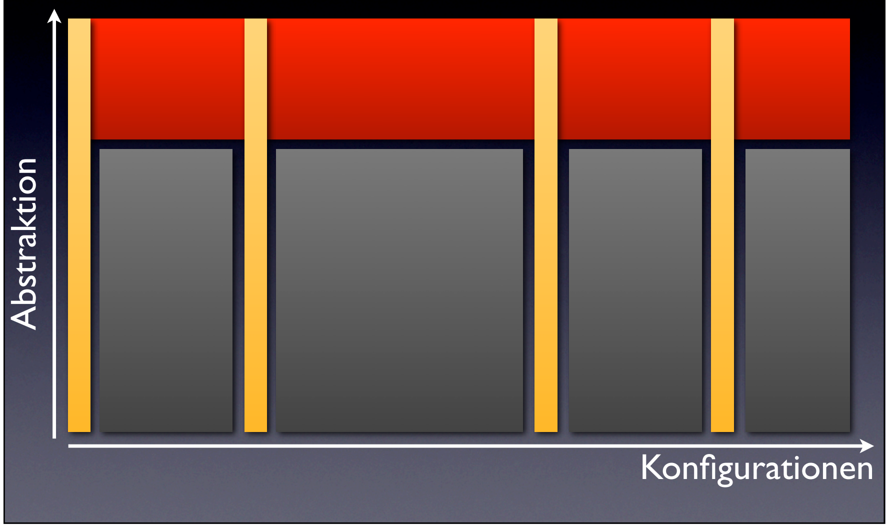
Webmate



Webmate



Das Beste aller Welten



We might not be able to cover **all** abstraction levels in **all** configurations, but we can do our best to cover as much as possible.

Wo sind die Fehler?
Prozess
frühere Fehlerorte und deren Eigenschaften
Programm
Programmtests Programmanalysen
Programme, die Programme prüfen
83.64 %

- Wissen, wo die Fehler sind
- Kalibrieren der Fehlervorhersage
- Wissen, wo die *nächsten* Fehler sind
- Vollautomatisch!