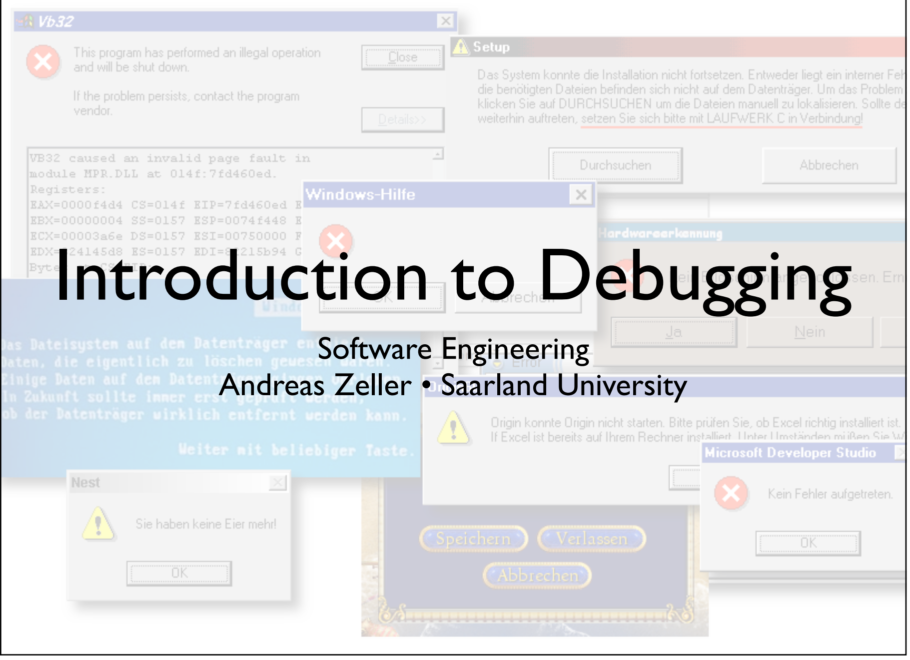
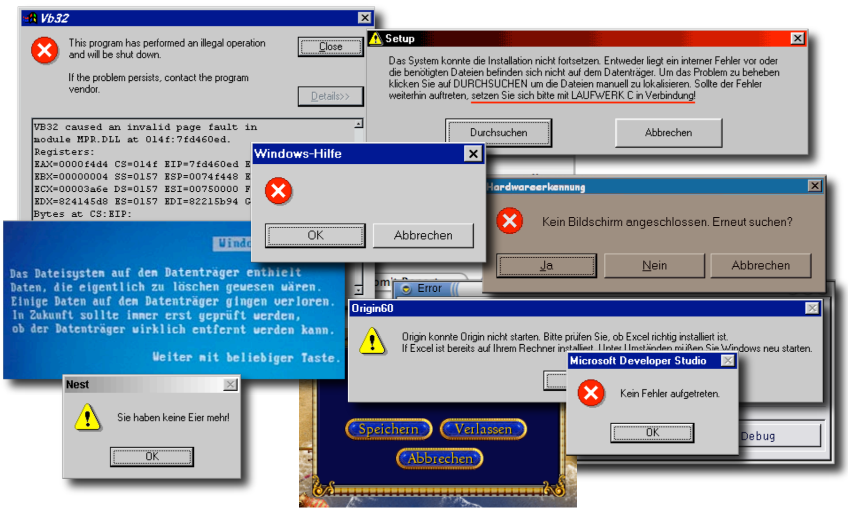


Introduction to Debugging

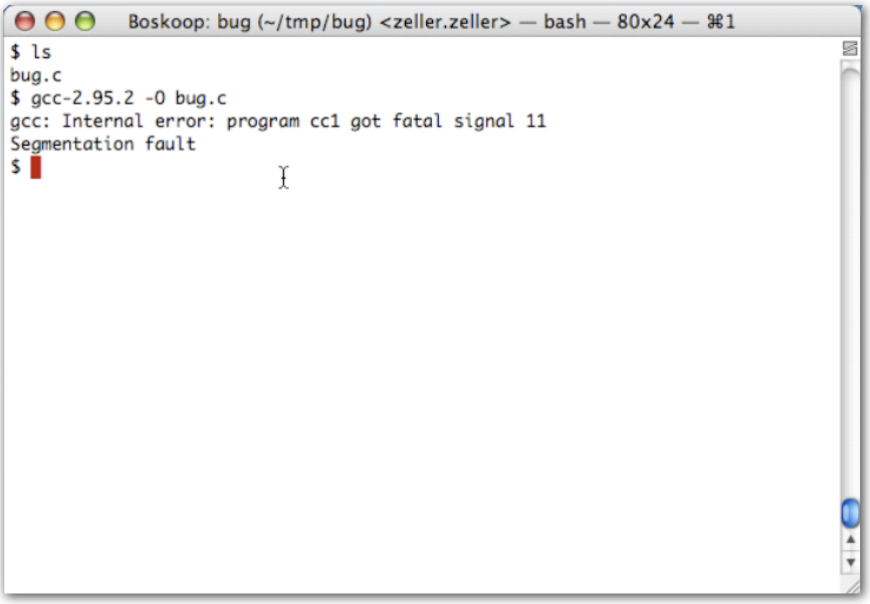
Software Engineering
Andreas Zeller • Saarland University

The Problem



Facts on Debugging

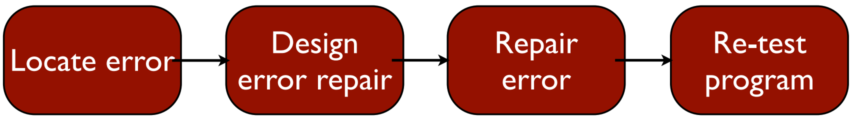
- Software bugs cost ~60 bln US$/yr in US
- Improvements could reduce cost by 30%
- Validation (including debugging) can easily take up to 50-75% of the development time
- When debugging, some people are *three times* as efficient than others



4

How to Debug

(Sommerville 2004)



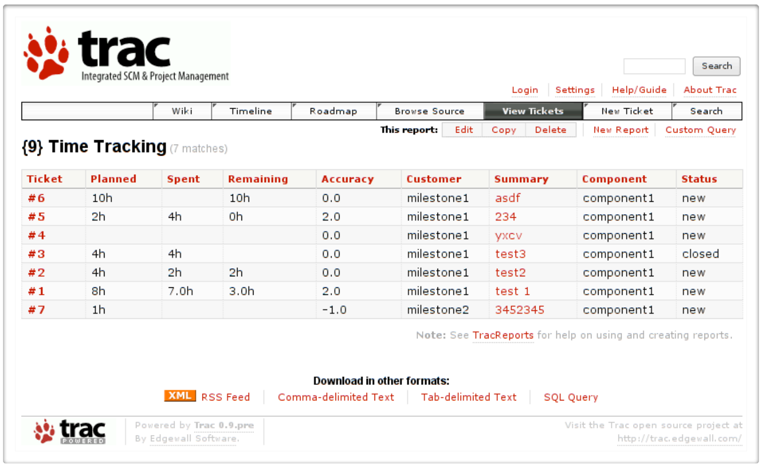
5

The Process

T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C orrect

6

Tracking Problems

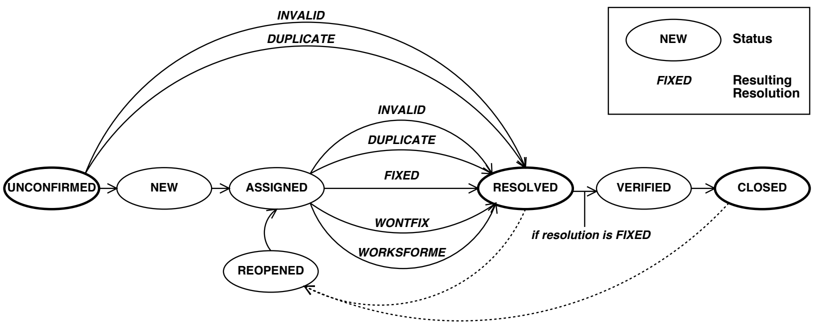


Ticket	Planned	Spent	Remaining	Accuracy	Customer	Summary	Component	Status
#6	10h		10h	0.0	milestone1	asdf	component1	new
#5	2h	4h	0h	2.0	milestone1	234	component1	new
#4				0.0	milestone1	yxcv	component1	new
#3	4h	4h		0.0	milestone1	test3	component1	closed
#2	4h	2h	2h	0.0	milestone1	test2	component1	new
#1	8h	7.0h	3.0h	2.0	milestone1	test 1	component1	new
#7	1h			-1.0	milestone2	3452345	component1	new

Tracking Problems

- Every problem gets entered into a *problem database*
- The *priority* determines which problem is handled next
- The product is ready when all problems are resolved

Problem Life Cycle



Reproduce

Randomness

Operating System

Communication

Concurrency

Interaction

Physics

Data

Debugger

Automate

```
// Test for host
public void testHost() {
  int noPort = -1;
  assertEquals(askigor_url.getHost(), "www.askigor.org");
  assertEquals(askigor_url.getPort(), noPort);
}

// Test for path
public void testPath() {
  assertEquals(askigor_url.getPath(), "/status.php");
}

// Test for query part
public void testQuery() {
  assertEquals(askigor_url.getQuery(), "id=sample");
}
```

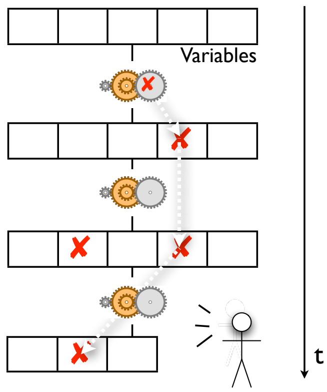

Automate

- Every problem should be *reproducible automatically*
- Achieved via appropriate (unit) tests
- After each change, we re-run the tests

Finding Origins

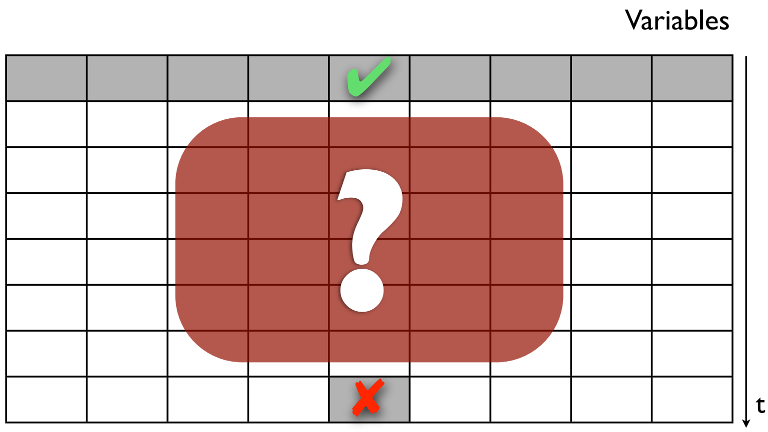
1. The programmer creates a *defect* in the code.
2. When executed, the defect creates an *infection*.
3. The infection *propagates*.
4. The infection causes a *failure*.

This infection chain must be traced back – and broken.

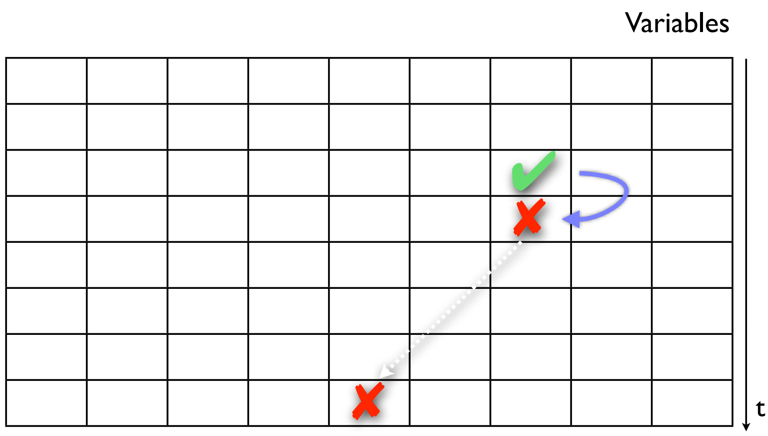


Not every defect creates an infection – not every infection results in a failure

Finding Origins

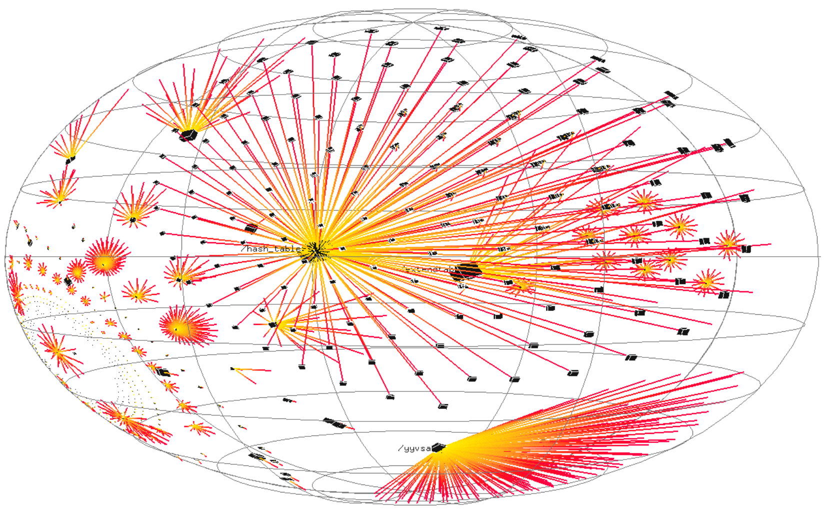


The Defect



TRAFFIC

A Program State



16

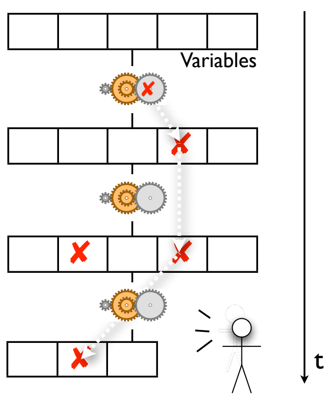
TRAFFIC

17

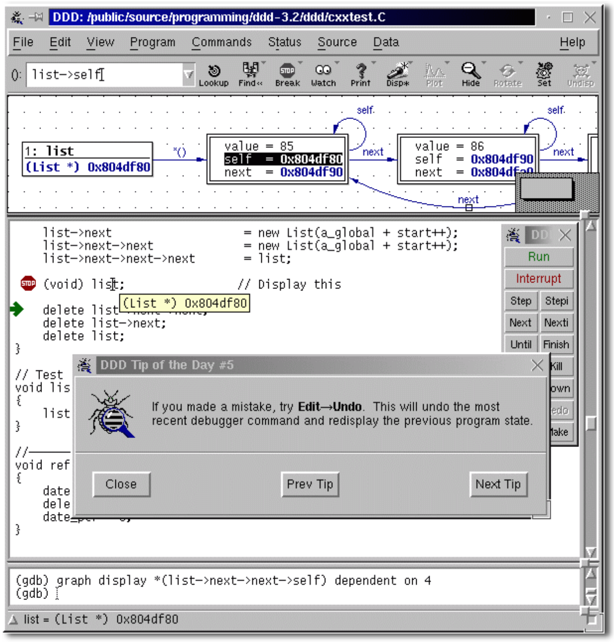
TRAFFIC

Finding Origins

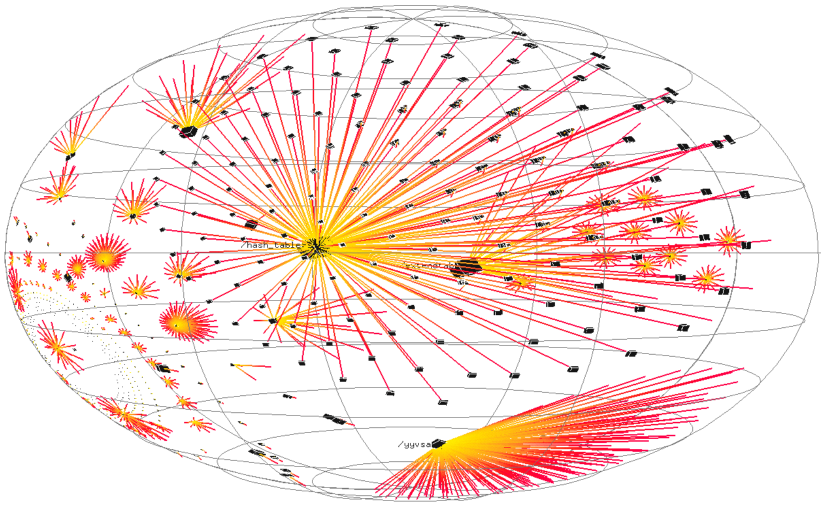
1. We start with a *known infection* (say, at the failure)
2. We search the infection in the *previous state*



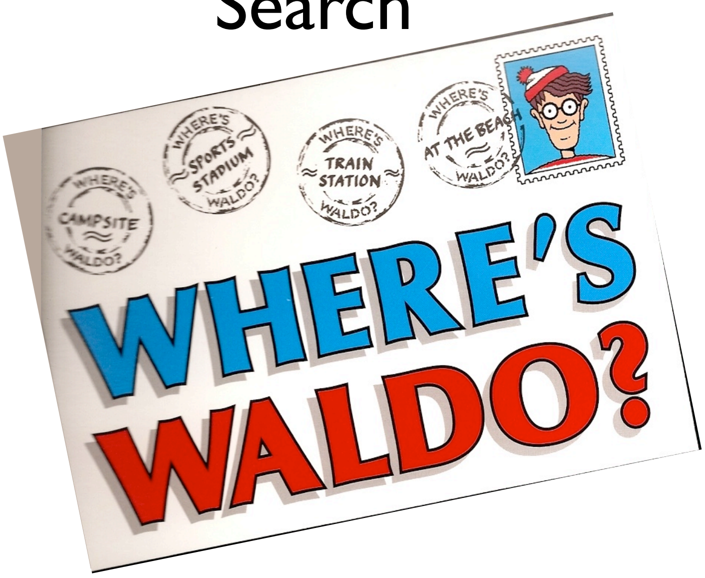
18



A Program State



Search



22

T R A F **F** I C

Focus

During our search for infection, we focus upon locations that

- *are possibly wrong*
  (e.g., because they were buggy before)
- *are explicitly wrong*
  (e.g., because they violate an *assertion*)

Assertions are the best way to find infections!

23

T R A F **F** I C

Finding Infections

```
class Time {
public:
    int hour();     // 0..23
    int minutes();  // 0..59
    int seconds();  // 0..60 (incl. leap seconds)

    void set_hour(int h);
    …
}
```

Every time between 00:00:00 and 23:59:60 is valid

24

T R A F F I C

Finding Origins

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}

void Time::set_hour(int h)
{
    assert (sane());  // Precondition
    …
    assert (sane());  // Postcondition
}
```


T R A F F I C

Finding Origins

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}
```

sane() is the *invariant* of a Time object:

- valid *before* every public method
- valid *after* every public method

T R A F F I C

Finding Origins

- Precondition fails = Infection *before* method
- Postcondition fails = Infection *after* method
- All assertions pass = no infection

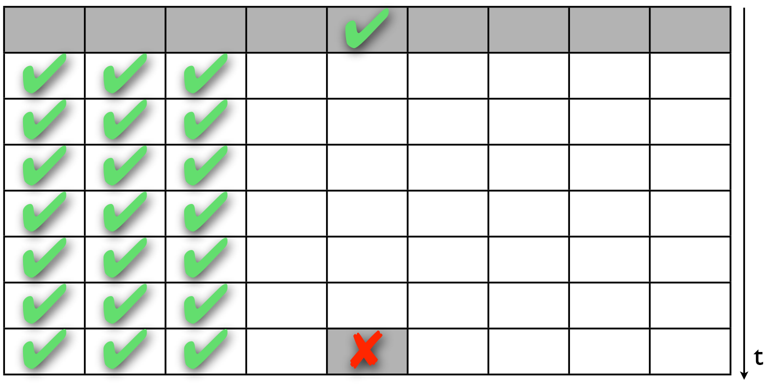
```
void Time::set_hour(int h)
{
    assert (sane());  // Precondition
    …
    assert (sane());  // Postcondition
}
```

Complex Invariants

```
class RedBlackTree {
  …
  boolean sane() {
      assert (rootHasNoParent());
      assert (rootIsBlack());
      assert (redNodesHaveOnlyBlackChildren());
      assert (equalNumberOfBlackNodesOnSubtrees());
      assert (treeIsAcyclic());
      assert (parentsAreConsistent());

      return true;
  }
}
```


Assertions



Focusing

- All possible influences must be checked
- Focusing on most likely candidates
- Assertions help in finding infections fast

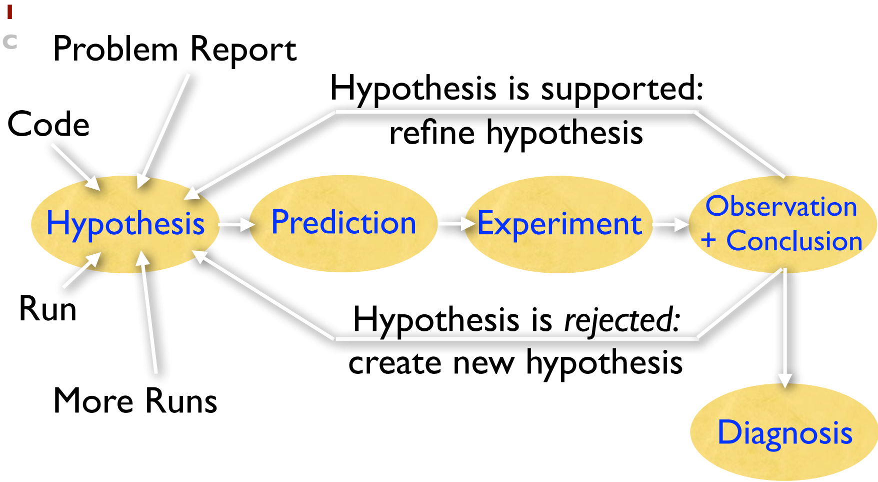
Isolation

- Failure causes should be *narrowed down systematically*
- Use *observation* and *experiments*

Scientific Method

1. Observe some aspect of the universe.
2. Invent a *hypothesis* that is consistent with the observation.
3. Use the hypothesis to make *predictions*.
4. Tests the predictions by experiments or observations and modify the hypothesis.
5. Repeat 3 and 4 to refine the hypothesis.

Scientific Method



Explicit Hypotheses

Hypothesis	The execu… auses a[0] = 0
Prediction	At L… should hold.
Experiment	… Line 37.
Observati…	…olds as predicted.
Conclusion	…hesis is confirmed.

Keeping everything in memory is like playing mastermind blind!

Explicit Hypotheses

Isolate

- We repeat the search for infection origins until we found the defect
- We proceed *systematically* along the scientific method
- *Explicit steps* guide the search – and make it repeatable at any time

Correction

Before correcting the defect, we must check whether the defect

- actually is an *error* and
- *causes* the failure

Only when we understood both, can we correct the defect

The Devil's Guide to Debugging

Find the defect by guessing:

- Scatter debugging statements everywhere
- Try changing code until something works
- Don't back up old versions of the code
- Don't bother understanding what the program should do

The Devil's Guide to Debugging

Don't waste time understanding the problem.

- Most problems are trivial, anyway.

The Devil's Guide to Debugging

Use the most obvious fix.

- Just fix what you see:

```
x = compute(y)
// compute(17) is wrong – fix it
if (y == 17)
    x = 25.15
```

Why bother going into compute()?

Successful Correction

T R A F F I C

Homework

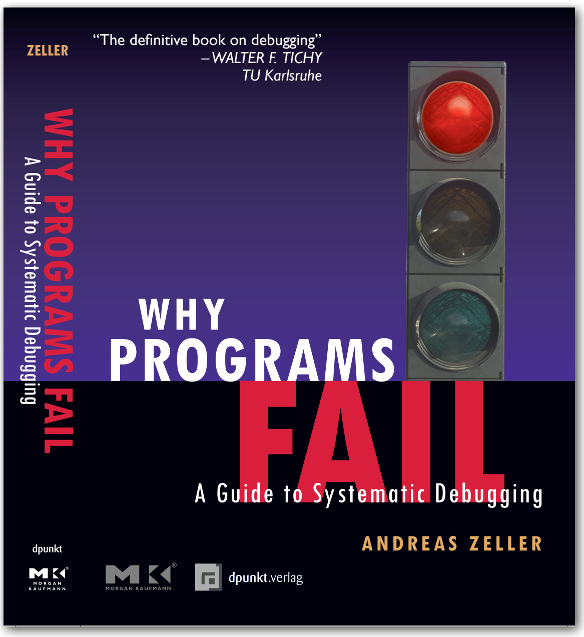
- Does the failure no longer occur?
  (If it does still occur, this should come as a big surprise)
- Did the correction introduce new problems?
- Was the same mistake made elsewhere?
- Did I commit the change to version control and problem tracking?

43

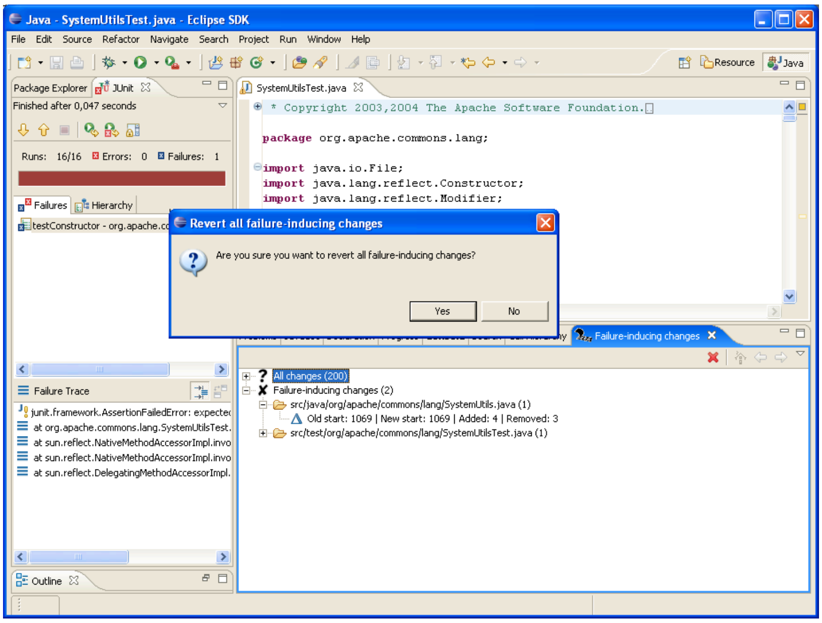
The Process

T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C orrect

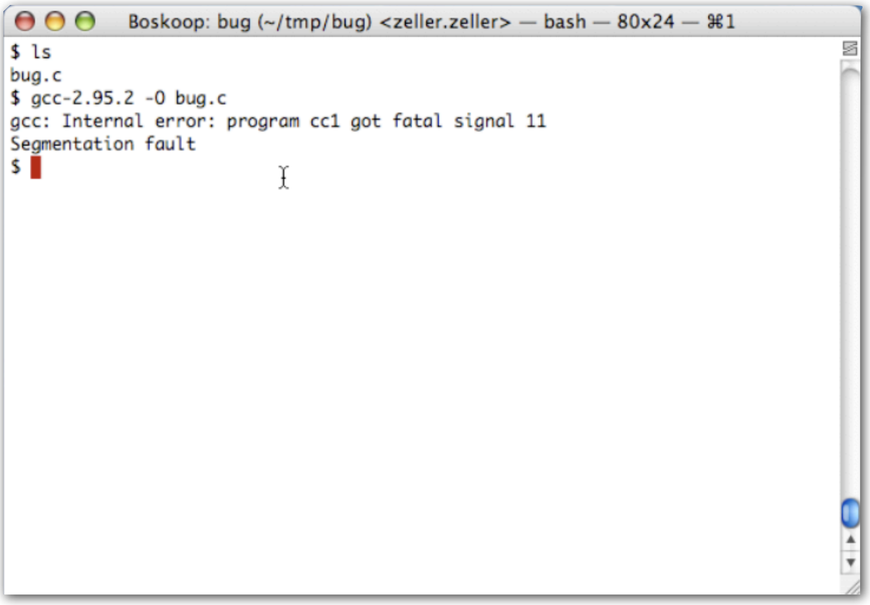
44



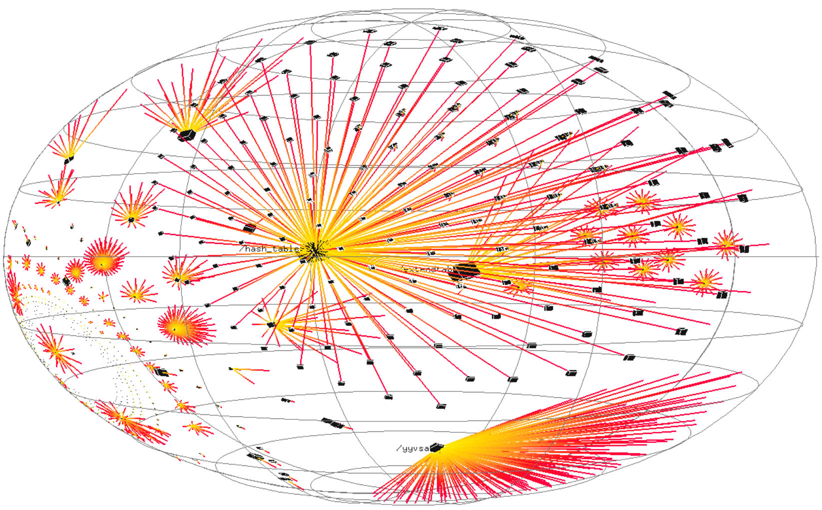
45


Java - SystemUtilsTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Resource
Java
Package Explorer
JUnit
Finished after 0,047 seconds
Runs: 16/16
Errors: 0
Failures: 1
Failures
Hierarchy
testConstructor - org.apache.co
Failure Trace
junit.framework.AssertionFailedError: expected
at org.apache.commons.lang.SystemUtilsTest.
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.DelegatingMethodAccessorImpl.
Outline
SystemUtilsTest.java
* Copyright 2003,2004 The Apache Software Foundation.
package org.apache.commons.lang;
import java.io.File;
import java.lang.reflect.Constructor;
import java.lang.reflect.Modifier;
Revert all failure-inducing changes
Are you sure you want to revert all failure-inducing changes?
Yes
No
Failure-inducing changes
All changes (200)
Failure-inducing changes (2)
src/java/org/apache/commons/lang/SystemUtils.java (1)
Old start: 1069 | New start: 1069 | Added: 4 | Removed: 3
src/test/org/apache/commons/lang/SystemUtilsTest.java (1)

46


Boskoop: bug (~/tmp/bug) <zeller.zeller> — bash — 80x24 — ⌘1
$ ls
bug.c
$ gcc-2.95.2 -O bug.c
gcc: Internal error: program cc1 got fatal signal 11
Segmentation fault
$

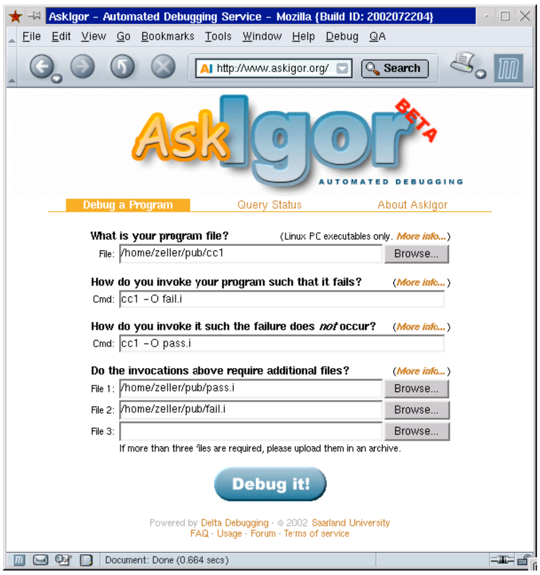
47

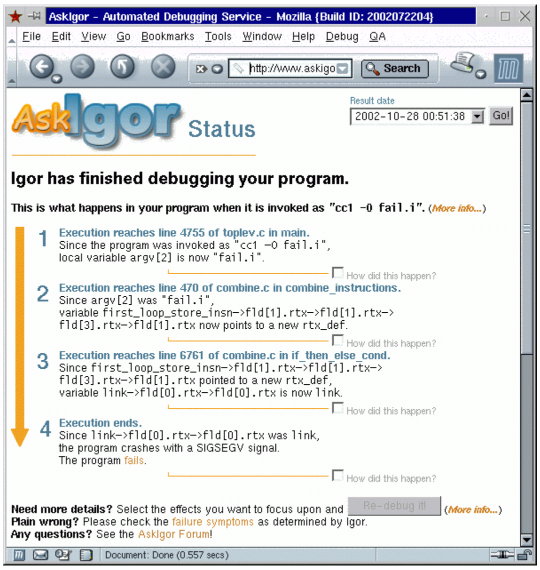


48

Failure Causes in GCC

Location	Failure Cause
<Start>	argv[3]
toplev.c:4755	name
toplev.c:2909	dump_base_name
c-lex.c:187	finput→_IO_buf_base
c-lex.c:1213	nextchar
c-lex.c:1213	yyssa[41]
c-typeck.c:3615	yyssa[42]
c-lex.c:1213	last_insn→fld[1].rtx→…→fld[1].rtx.code
c-decl.c:1213	sequence_result[2]→…→fld[1].rtx.code
combine.c:4271	x→fld[0].rtx→fld[0].rtx

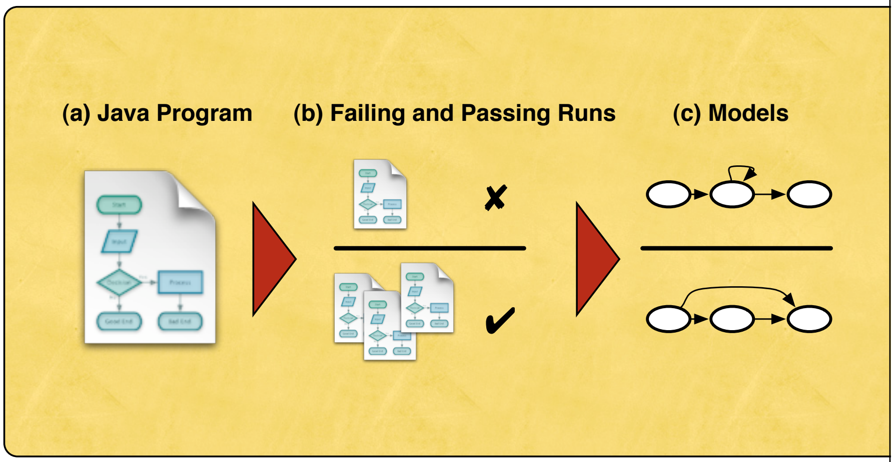




Automatic Fixes!

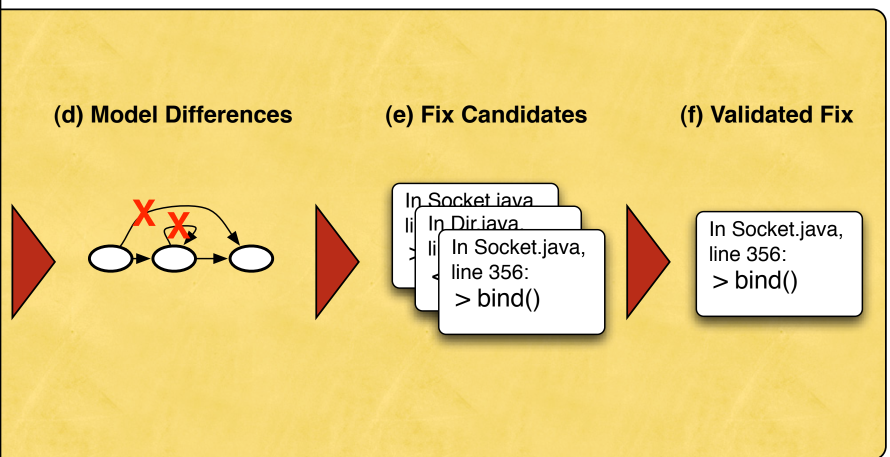
52

Automatic Fixes



53

Automatic Fixes



54

Mining Object Behavior

Mutators
change state

Inspectors
return state

Use static analysis to differentiate

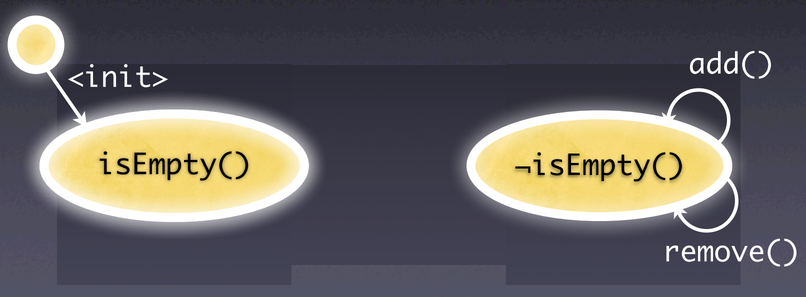
55

Building Models

- After each mutator call, we extract attributes and invoke the inspectors
- Extracted states form finite state machine

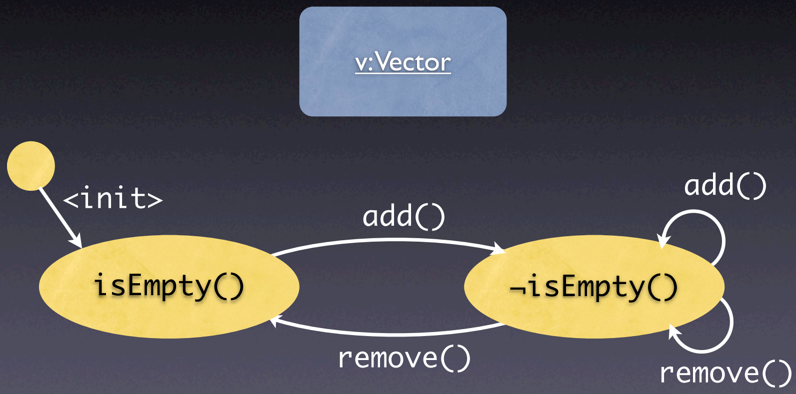
56

Building Models



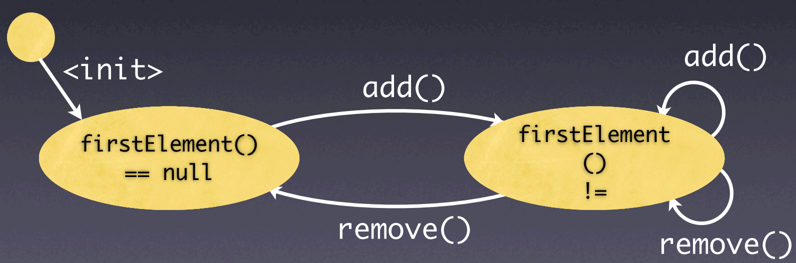
57

Building Models

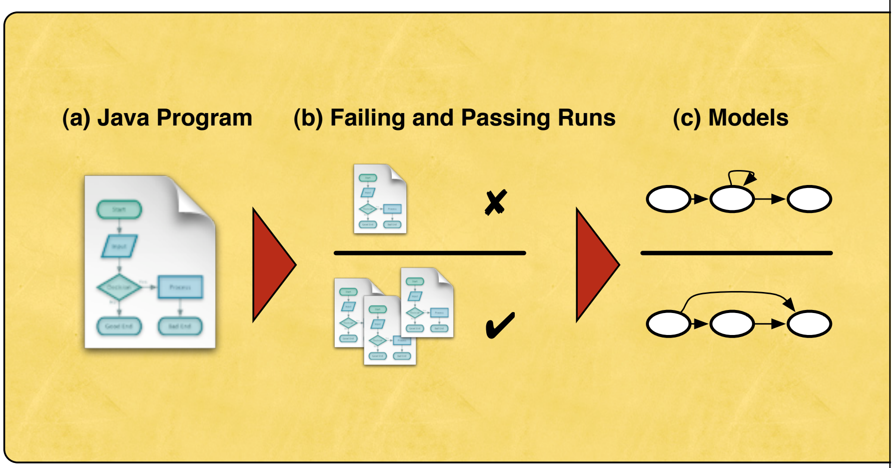


Equivalence Classes

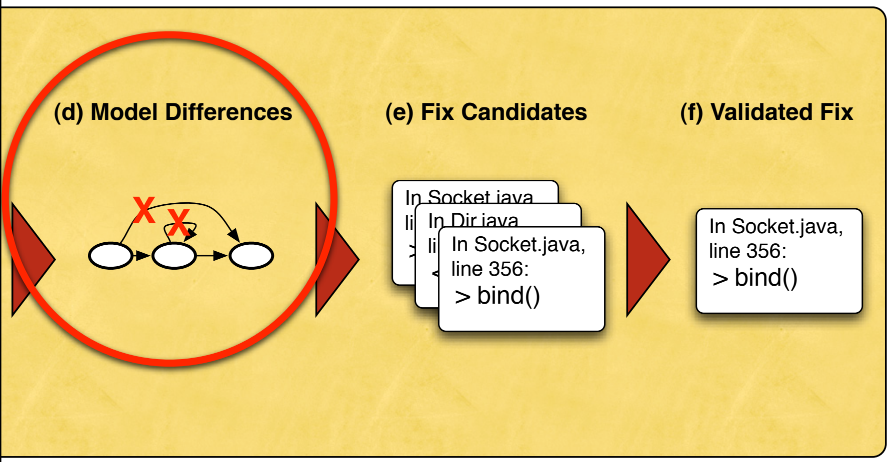
Inspector type	boolean	numeric	object
States	true | false	< 0 | = 0 | > 0	null | class

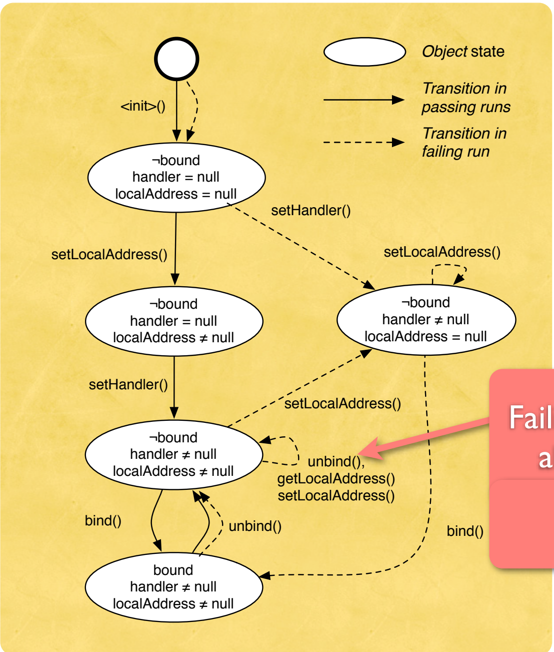


Automatic Fixes



Automatic Fixes





Mina

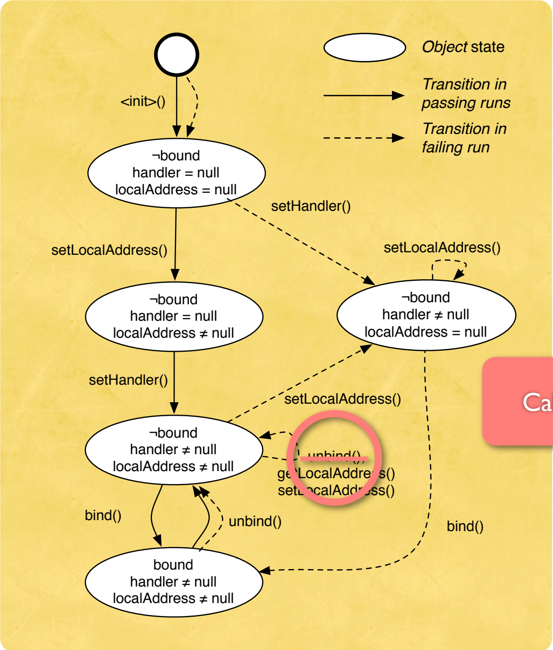
- Multipurpose Infrastructure for Network Applications

Failing run calls unbind() although not bound

Can we fix it?

Deleting Calls

- The first option to create fixes is to *delete calls:*
- Make calls dependent on precondition
- Or, make callees return when precondition does not hold



Fix it!

Deleting Calls

- The first option to create fixes is to *delete calls:*
- Make calls dependent on precondition
- Or, make callees return when precondition does not hold

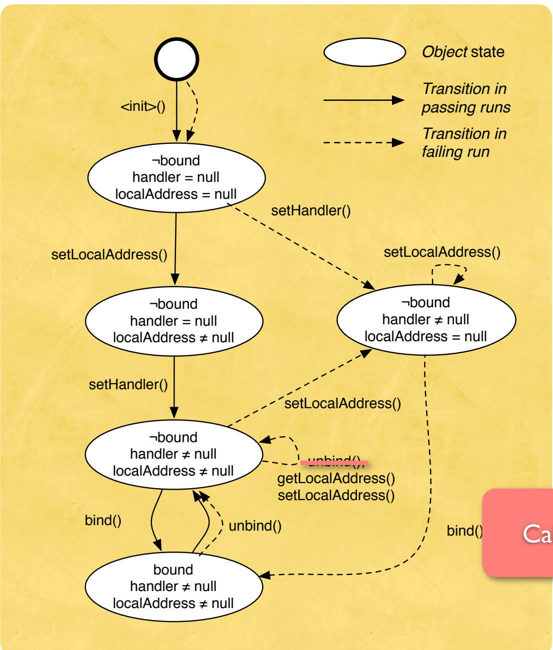
Call unbind() only if bound

64

Inserting Calls

- The second fix option is to *insert calls:*
- For a violated precondition, insert calls to reach that state
- May need to traverse model for that

65



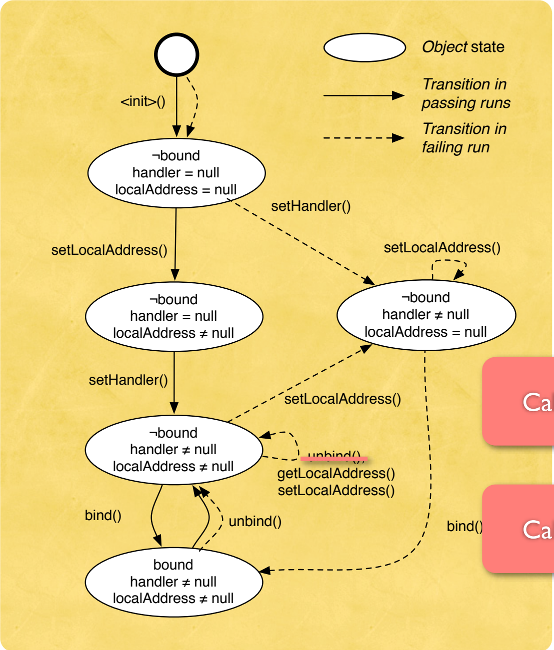
Fix it!

Inserting Calls

- The second fix option is to *insert calls:*
- For a violated precondition, insert calls to reach that state
- May need to traverse model for that

Call bind() before unbind()

66



Fix it!

Call unbind() only if bound

Call bind() before unbind()

Validating Fixes

All fix options must be *validated:*

We validate fix candidates

1. On failing test
2. On entire test suite

Only validated fixes remain

Call unbind() only if bound ✔

Call bind() before unbind() ✔

Pachika

Suaheli for "fix", "insert"

- Tool for automatic fixing of Java programs
- Takes a failing run and a test suite
- Produces either a validated fix – or nothing
- Available for download

Bug	Candidate Fixes Insert	Candidate Fixes Delete	Potential Fixes	Validated Fixes
34858	420	50	0	0
43033	219	65	0	0
51322	**112**	**190**	**56**	**1**
67774	0	72	0	0
70619	6	1	0	0
75129	0	0	0	0
87376	20	218	0	0
107858	405	235	0	0
109614	0	0	0	0
120474	0	0	0	0
121616	**123**	**0**	**38**	**1**
125475	72	122	7	0
128237	283	4	123	0
131933	0	50	0	0
152631	0	783	0	0
158412	2895	310	0	0
158624	0	0	0	0
173602	**17**	**13**	**7**	**1**

AspectJ

- Compiler for AOP programs
- Great source of bugs

Bug 173602

```
  public void resolve(ClassScope upperScope) {
>     // Fix from source repository
>     if (binding == null)
>        ignoreFurtherInvestigation = true;
>     // Fix generated by PACHIKA
>     if (binding == null)
>          return;
      if (munger == null)
          ignoreFurtherInvestigation = true;
      if (ignoreFurtherInvestigation) return;
         ...
      }
 }
```


Bug 121616

```
  public boolean visit(MethodDeclaration md,
                       ClassScope scope) {
>   // Fix generated by PACHIKA
>   // (same as in the source repository)
>   if (methodDeclaration.hasErrors())
>       return false;
    ContextToken tok = ...
    ...
  }
```

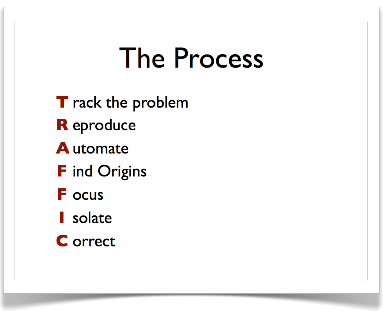
Bug 51322

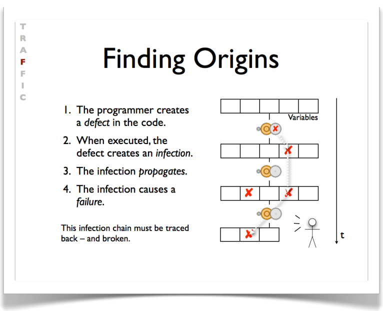
```
public EclipseTypeMunger build(ClassScope cs)
{
    ...
       binding = classScope.referenceContext.
                    binding.resolveTypesFor(binding);
>      // Fix generated by PACHIKA
>      binding.constantPoolDeclaringClass().
>              addDefaultAbstractMethods();
>      binding.constantPoolDeclaringClass().methods();
>      // Fix from source repository
>      if (binding == null)
>          throw new AbortCompilation();
       ResolvedMember sig = new ResolvedMember(...);
       ...
}
```


Automatic Fixing

- Adaptive fix generation
- Assessing the impact of fixes
- Leveraging contracts
- Programs that fix themselves

http://www.st.cs.uni-saarland.de/models/





Summary