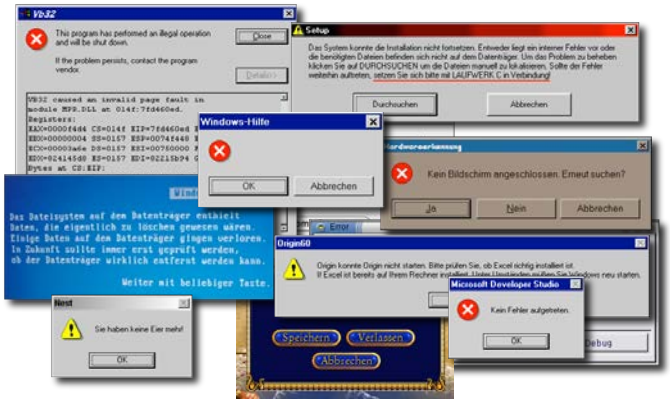
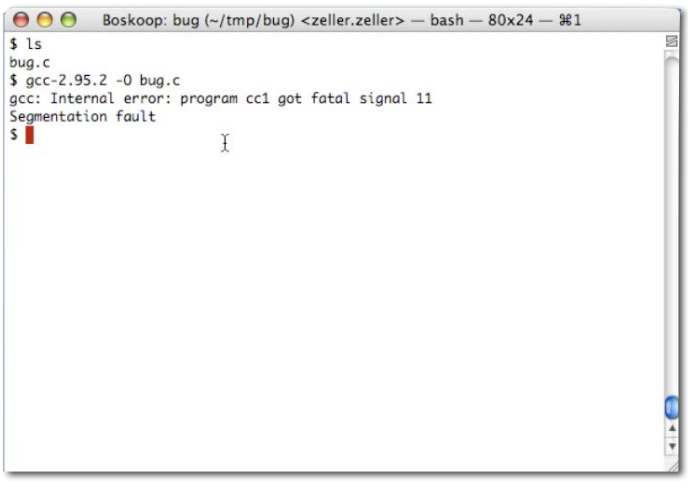
Fehlersuche

Software-Praktikum
Andreas Zeller, Universität des Saarlandes

Das Problem



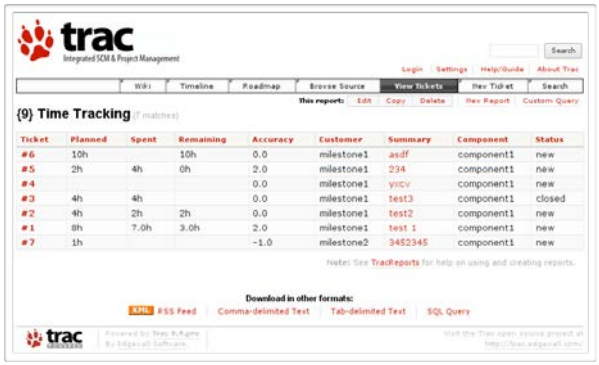


Vorgehensweise

T rack the problem	*Problem verfolgen*
R eproduce	*Reproduzieren*
A utomate	*Automatisieren*
F ind Origins	*Ursprünge finden*
F ocus	*Fokussieren*
I solate	*Isolieren*
C orrect	*Korrigieren*

T
R
A
F
F
I
C

Problem verfolgen



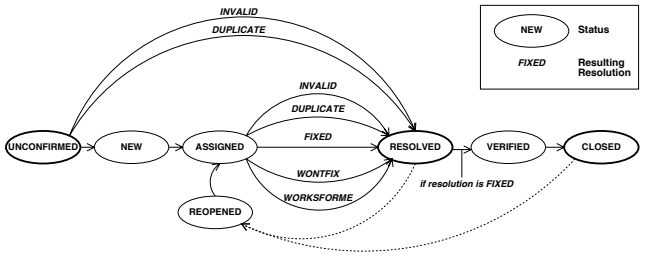
T
R
A
F
F
I
C

Problem verfolgen

- Jedes Problem wird in die Fehler-Datenbank eingetragen
- Die Priorität bestimmt, welches Problem als nächstes bearbeitet wird
- Sind alle Probleme behoben, ist das Produkt fertig

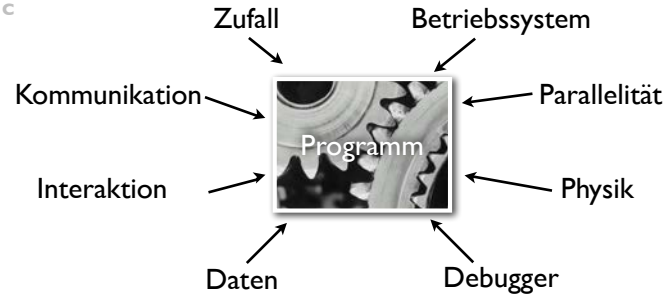
T R A F F I C

Lebenszyklus eines Problems



T R A F F I C

Reproduzieren



T R A F F I C

Automatisieren

```
// 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");
}
```

T R A F F I C

Automatisieren

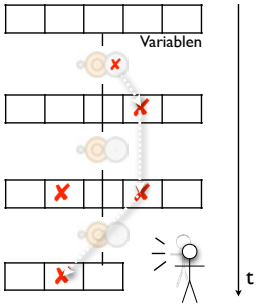
- Jedes Problem sollte automatisch reproduzierbar sein
- Dies geschieht über geeignete JUnit-Testfälle
- Nach jeder Änderung werden die Testfälle ausgeführt

T R A F F I C

Ursprung finden

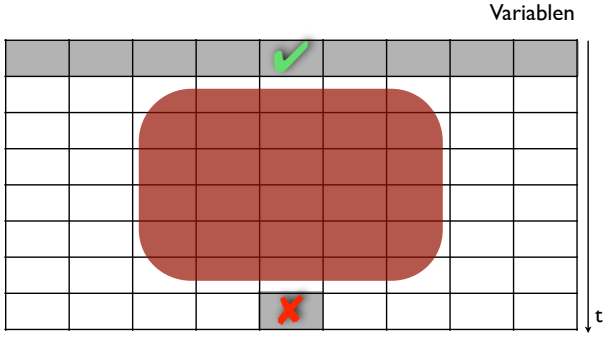
1. Der Programmierer erzeugt einen *Defekt* – einen Fehler im Code
2. Der ausgeführte Defekt erzeugt eine *Infektion* – einen Fehler im Zustand
3. Die Infektion breitet sich aus…
4. …und wird als *Fehlverhalten*

Diese Infektionskette müssen wir *brechen*.



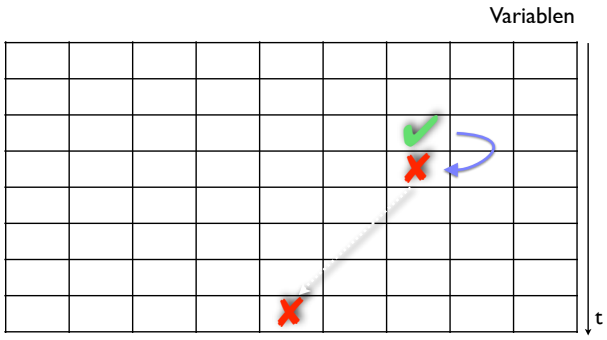
T R A F F I C

Ursprung finden



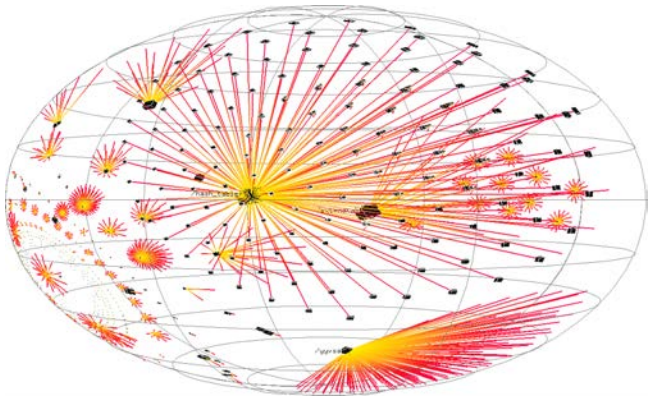
TRAFFIC

Der Defekt



TRAFFIC

Ein Programmzustand

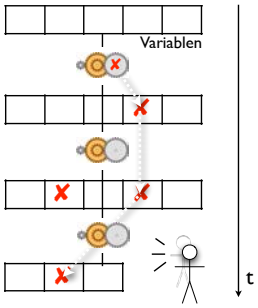


TRAFFIC

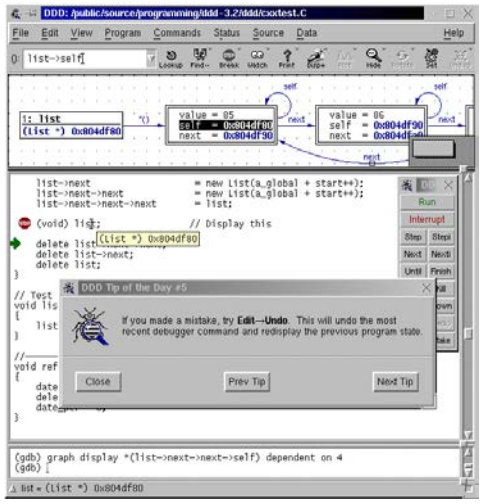
TRAFFIC

Ursprung finden

1. Wir beginnen mit einer *bekannten Infektion* (etwa am *Ende der Ausführung*)
2. Wir suchen die Infektion im *vorherigen Zustand*

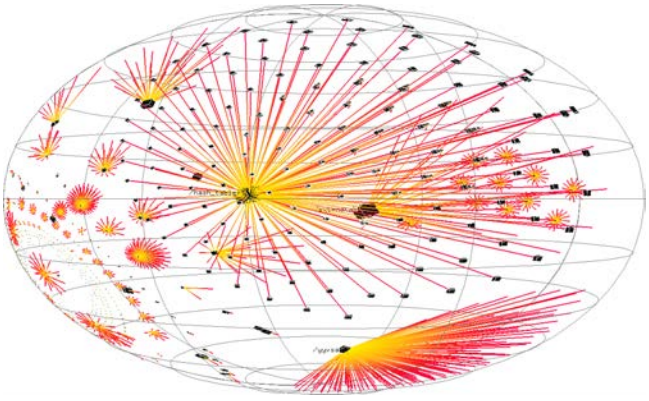


TRAFFIC



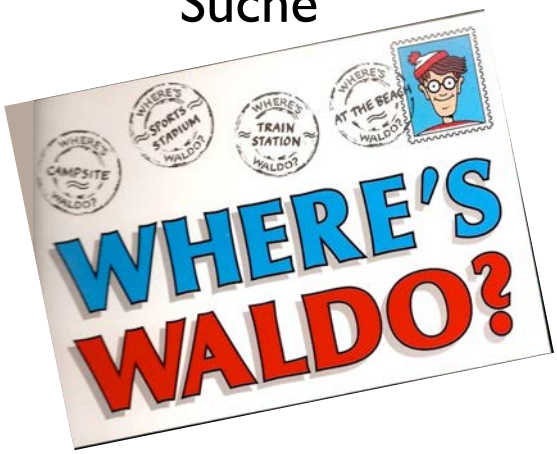
TRAFFIC

Ein Programmzustand



TRAFFIC

Suche



TRAFFIC

Fokussieren

Bei der Suche nach Infektionen konzentrieren uns auf Stellen im Zustand, die

- *wahrscheinlich falsch* sind (z.B. weil hier früher Fehler aufgetreten sind)
- *explizit falsch* sind (z.B. weil sie eine *Zusicherung* verletzen)

Zusicherungen sind das effektivste Mittel, Infektionen zu finden.

TRAFFIC

Infektionen finden

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

    void set_hour(int h);
    …
}
```

Jede Zeit von 00:00:00 bis 23:59:60 ist gültig

TRAFFIC

Ursprung finden

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

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

TRAFFIC

Ursprung finden

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

sane() ist die *Invariante* eines Time-Objekts:

- gilt *vor* jeder öffentlichen Methode
- gilt *nach* jeder öffentlichen Methode

TRAFFIC

Ursprung finden

- Vorbedingung schlägt fehl = Infektion *vor* Methode
- Nachbedingung schlägt fehl = Infektion *nach* Methode
- Alle Zusicherungen ok = keine Infektion

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

TRAFFIC

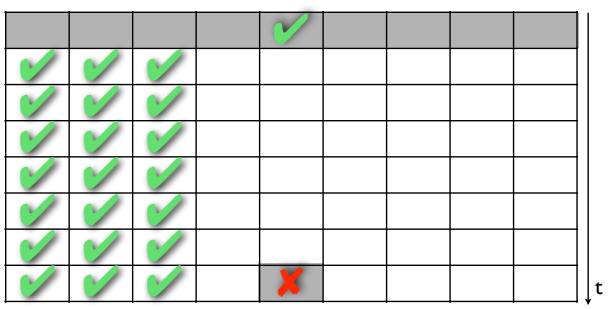
Komplexe Invarianten

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

      return true;
  }
}
```

TRAFFIC

Zusicherungen



T
R
A
F
F
I
C

Fokussieren

- Alle möglichen Einflüsse müssen geprüft werden
- Konzentration auf wahrscheinlichste Kandidaten
- Zusicherungen helfen schnell, Infektionen zu finden

T
R
A
F
F
I
C

Isolieren

- Fehlerursachen sollen *systematisch* eingeengt werden – mit Beobachtungen und Experimenten.

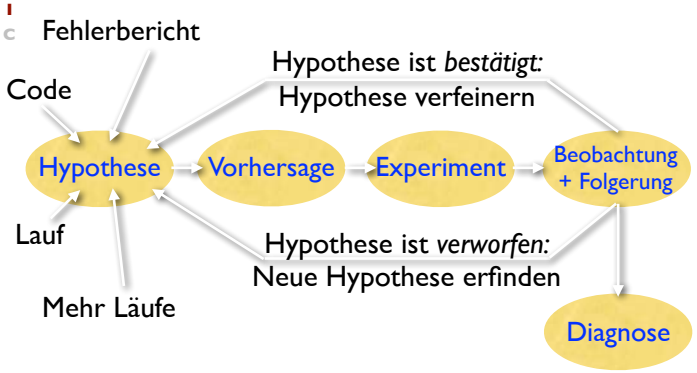
T
R
A
F
F
I
C

Wissenschaftliche Methode

1. Beobachte einen Teil des Universums
2. Erfinde eine *Hypothese*, die mit der Beobachtung übereinstimmt
3. Nutze die Hypothese, um Vorhersagen zu machen.
4. Teste die Vorhersagen durch Experimente oder Beobachtungen und passe die Hypothese an.
5. Wiederhole 3 and 4, bis die Hypothese zur *Theorie* wird.

TRAFFIC

Wissenschaftliche Methode



TRAFFIC

TRAFFIC

Explizite Hypothesen

Hypothesis	The exec… …uses a[0] = 0
Prediction	At l… …hould hold.
Experiment	…me 37.
Observat…	…as as predicted.
Conclusion	…nesis is **confirmed**.

Wer alles im Kopf behält, spielt Mastermind blind!

T R A F F I C

Explizite Hypothesen

T R A F F I C

Isolieren

- Wir wiederholen die Suche nach Infektions-Ursprüngen, bis wir den Defekt gefunden haben.
- Wir gehen *systematisch* vor – im Sinne der wissenschaftlichen Methode
- Durch *explizite* Schritte leiten wir die Suche und können sie jederzeit nachvollziehen

T R A F F I C

Korrektur

Vor der Korrektur müssen wir prüfen, ob der Defekt

- tatsächlich ein *Fehler* ist und
- das Fehlverhalten *verursacht*

Erst wenn beides verstanden ist, dürfen wir den Fehler korrigieren.

TRAFFIC

The Devil's Guide to Debugging

Finde den Defekt durch Raten:

- Verstreue überall Debugging-Anweisungen
- Ändere den Code, bis etwas funktioniert
- Mache keine Kopien von alten Versionen
- Versuche gar nicht erst zu verstehen, was das Programm tun soll

---

TRAFFIC

The Devil's Guide to Debugging

Verschwende keine Zeit damit, dem Problem auf den Grund zu gehen

- Die meisten Probleme sind ohnehin trivial

TRAFFIC

The Devil's Guide to Debugging

Benutze die offensichtlichste Reparatur:

- Repariere nur das, was Du siehst:

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

Warum sich mit compute() beschäftigen?

T R A F F I C

Erfolgreiche Korrektur

http://www.theverge.com/2014/7/5/5873259/watch-these-drones-fly-through-breathtaking-fireworks-displays

---

T R A F F I C

Hausaufgaben

- Tritt das Fehlverhalten nicht mehr auf?
  (Falls doch, sollte dies eine große Überraschung sein)
- Könnte die Korrektur neue Fehler einführen?
- Wurde derselbe Fehler woanders gemacht?
- Ist meine Korrektur ins Versionsmanagement und Problem-Tracking eingespielt?

Vorgehensweise

T rack the problem	*Problem verfolgen*
R eproduce	*Reproduzieren*
A utomate	*Automatisieren*
F ind Origins	*Ursprünge finden*
F ocus	*Fokussieren*
I solate	*Isolieren*
C orrect	*Korrigieren*

Vorgehensweise

T rack the problem	*Problem verfolgen*
R eproduce	*Reproduzieren*
A utomate	*Automatisieren*
F ind Origins	*Ursprünge finden*
F ocus	*Fokussieren*
I solate	*Isolieren*
C orrect	*Korrigieren*