Dr. Hanno Schauer
Mons-Tabor-Gymnasium
Montabaur

Debugging-Aufgaben

Aus Programmfehlern lernen

No Machine is Perfect!

1970er-Jahre

- Der Autopilot von Prototypen des Kampfflugzeug F-16 brachte das Flugzeug in Rückenlage, wenn der Äquator überflogen wurde.

- Hintergrund: Bei der Programmierung waren „negative" Breitengrade nicht berücksichtigt worden. [2]

Januar 2010

- Deutsche Geldautomaten verweigern die Geldausgabe.
- Die EVM-Chips der EC-Karten verarbeiten die Jahreszahl `2010` fehlerhaft [1].

Bildquelle: https://commons.wikimedia.org/wiki/File:JGC_VISA01s.jpg; 663highland (GFDL & CC 2.5)

Mai 2015

- Ein Militär-Airbus A 400 M stürzte nahe Sevilla ab. Vier Insassen starben, zwei wurden schwer verletzt.
- Ein Softwarefehler in der Triebwerkssteuerung hatte kurz nach dem Start drei der vier Triebwerke abgeschaltet . [3]

Bildquelle: Julian Herzog [GFDL (http://www.gnu.org/copyleft/fdl.html)

Bedeutung von Wartung und Test in der Informatik-Praxis

- Gesamt-Projektaufwand [Balzert]
  - Entwicklung 20 – 33 %
  - Wartung und Pflege **67 – 80 %**
- Je langlebiger ein Produkt/Informationssystem, desto höher der Aufwand für Wartung und Pflege

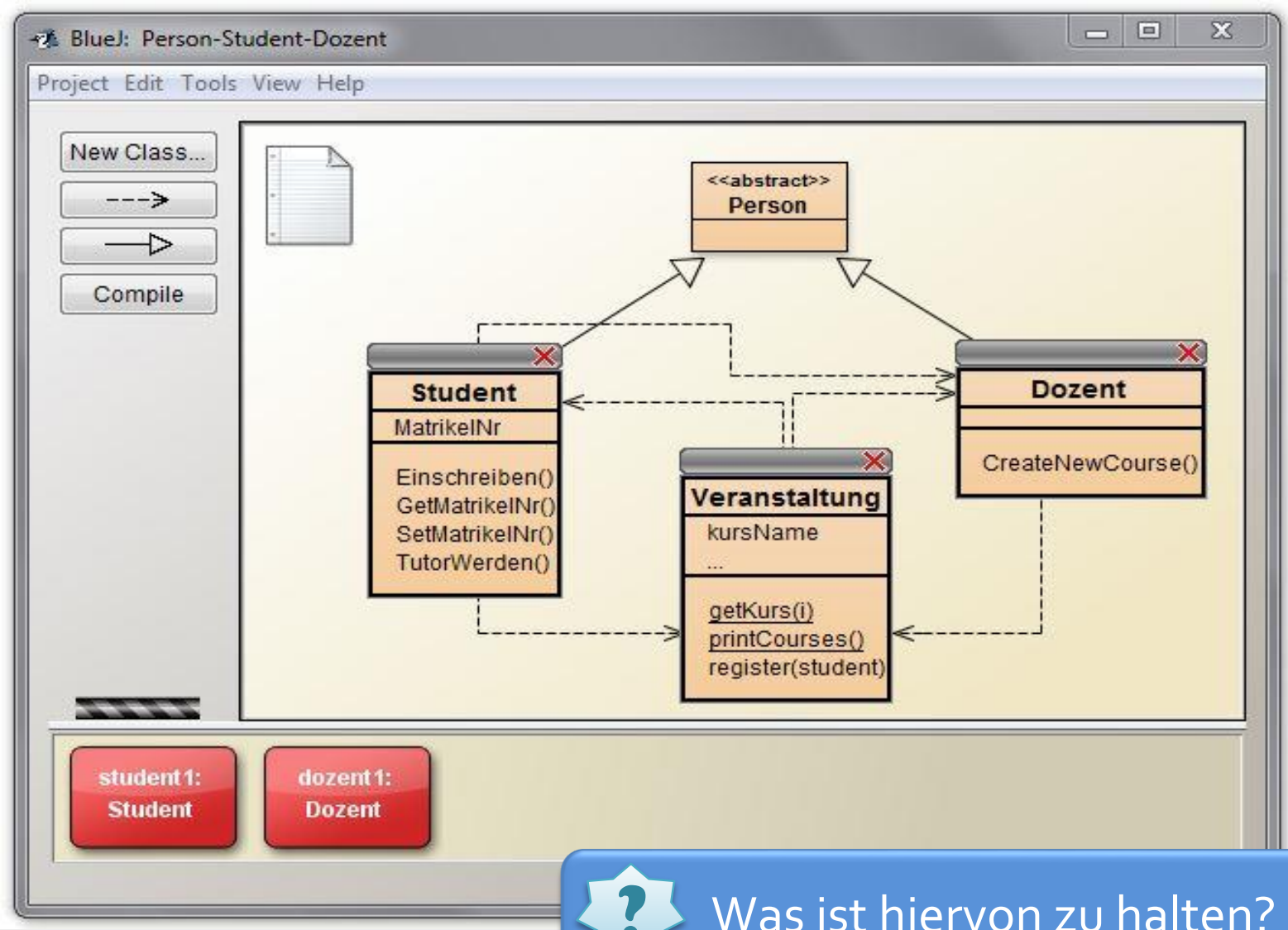
Gilt in ähnlicher Weise auch für viele technische Produkte – nicht nur Informationssysteme

Debugging-Aufgaben

Ansatz

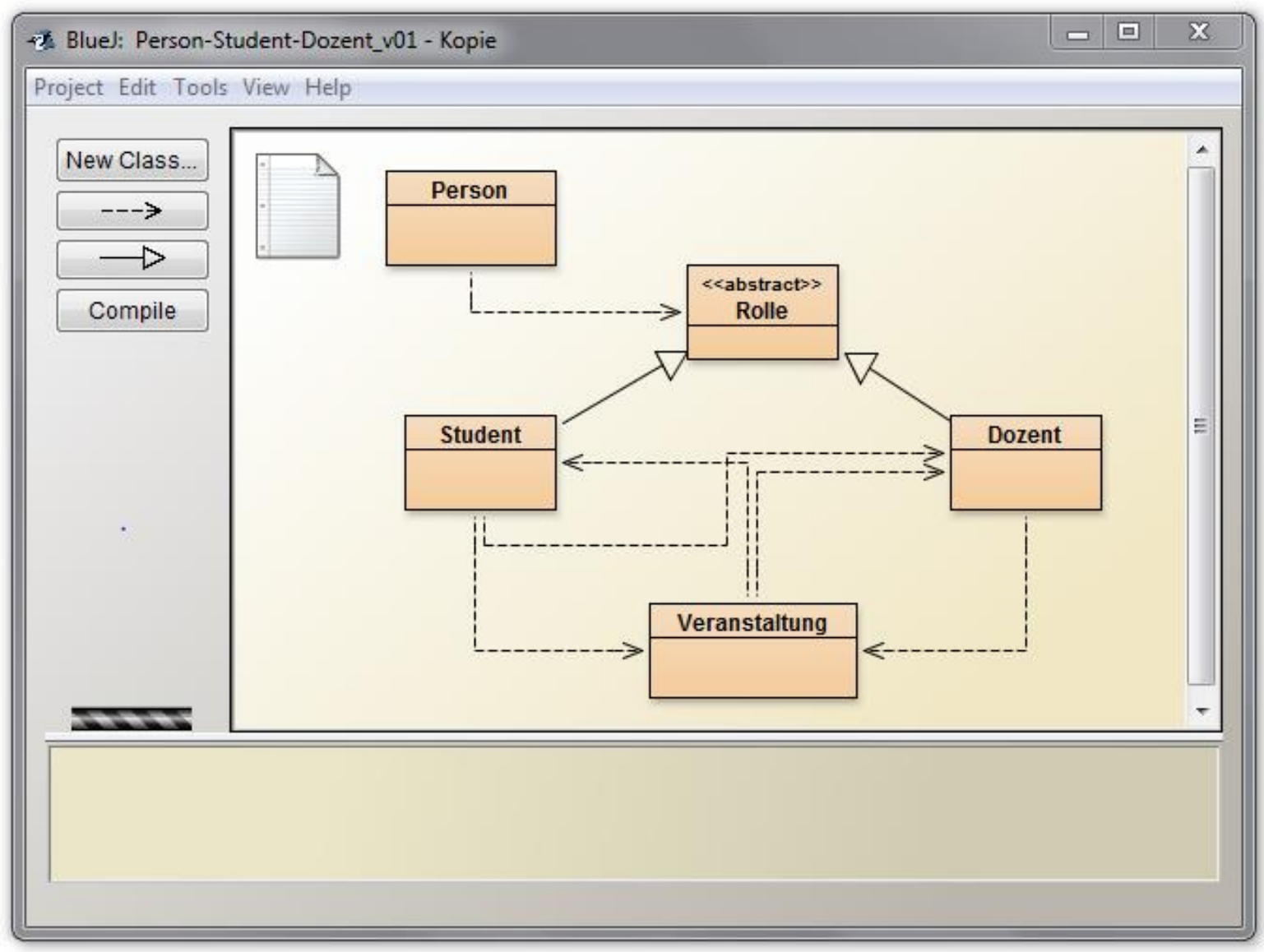
- Debugger als **Analysewerkzeug** im Unterricht
  - Authentische Aufgaben zu Test / Wartung / Pflege
  - Analyse „echter“ (lauffähiger) Programme
- Hierbei: **Aus Programmfehlern lernen**
  - Wiederkehrende („hartnäckige“) Fehlerquellen im Vordergrund

Ein Beispiel auf Universitätsniveau



Was ist hiervon zu halten?

… dann doch besser so …



„Erfolgsfaktoren“ der Aufgabengestaltung (1)

Leicht auffindbare **Fehlerquellen**!

```
// Initialisierung
```

Exemplarität der Fehler!

```
// Verarbeitung
for i := 1 to zuege do begin
```

Verbalisierung der Fehler (und ggf. geeigneter Korrekturen) im Unterricht!

```
// Gewinnzuege Spieler1)
```

Umsetzung „Debugging-Aufgabe“

Schüler(innen) erhalten fehlerhaftes, aber compilierbares Programm

a) **Fehler** mithilfe des Debuggers **analysieren** und **korrigieren**

1) **Fehler entdecken bzw. rekonstruieren**
   - Anwenden des Programms

2) **Fehlerquelle im Quellcode lokalisieren**
   - Haltepunkt setzen
   - Programm im Tracemodus durchlaufen
   - Werte relevanter Variablen überwachen

3) **Fraglichen Quelltextausschnitt analysieren**

4) **Fehler korrigieren**

b) Über Fehler **diskutieren** (Fehlerursachen verbalisieren)

Fokus auf Semantik- und Laufzeitfehler

Ein weiteres Beispiel

Aufgabe: **ASCII + Art (Debugging-Aufgabe)**

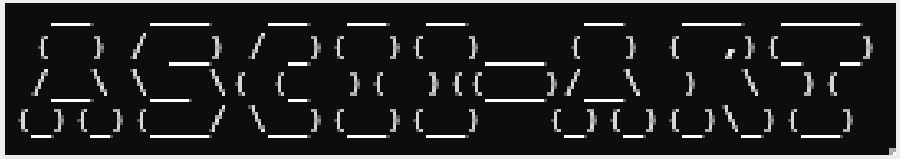
Im Ordner `.\BlueJ\ASCII-Art\` findet sich der Prototyp eines ASCII-Art-Schrift-Generators. Leider funktioniert der Generator nicht wie gewünscht. Es haben sich zwei Fehler eingeschlichen: ein Laufzeitfehler und ein Semantikfehler.

a) Finden Sie die Fehler mithilfe des Debuggers und korrigieren sie diese.

Tipp: Beobachten Sie insb. die Variablen `style`, `italic`, `bold` und `eingabeKorrekt`.

Hilfestellung: In der Datei `.\Hilfestellung\ASCII_ART.jar` findet sich (zum Vergleich) eine fehlerbereinigte Version des Programms.

b) Warum tritt der Semantikfehler in den Methoden Klasse `ABC` nicht auf?



Anwendungsgebiete

- Einführung in die **Programmiertechnik „Debugging“**
- **Typische** (**hartnäckige**) **Fehlerquellen** entdecken
  - Schlechter (Entwurfs-) Stil
  - Eigenheiten einer Programmiersprache
  - Fehler in Algorithmen
  - Fehler, die in Übungsaufgaben selten gemacht werden
- **Test und Wartung** in der Softwareentwicklung
  - Umfangreichere Debugging-Aufgabe(n)
  - in Form von Fallstudien (z. B. „Trouble Ticket“)

„Erfolgsfaktoren“ der Aufgabengestaltung (2)

- **Mehrere Fehler** pro Aufgabe implementieren
  - gestaffelte Schwierigkeitsstufen
  - Z. B. erst Laufzeit-, dann Semantikfehler
- **Quellcode-Umfang**: das 2- bis 4-Fache dessen, was Schüler(innen) üblicherweise abfassen
- **Unterstützung zum Programmverstehen**
  - Konzeptuelle Modelle (z. B. Use Case- und Klassendiagramme),
  - Ergänzende Dokumentation (z. B. Anforderungsspezifikation) und
  - Hilfestellungen (z. B. compilierte Musterlösungen des Programms).

Schüler(innen) lernen Nutzen der Dokumentation und Struktur hautnah

3 Didaktische Einordnung

Lehrer-Perspektive

- Debugging-Aufgaben unterstützen **wichtige Erfahrungen**
  - typische Fehlerquellen entdecken
  - Test, Wartung und Pflege von Software nachempfinden
  - Nutzen strukturierten Quellcodes, aussagekräftiger Modelle und verständlicher Dokumentation
- **Entdeckendes Lernen**
- Aber: **Keine offenen Aufgaben**
- **Aufgabenentwurf** ist **aufwändig** und erfordert Erfahrung

Schülermeinungen

- *„Man weiß, wie man sich selbst helfen kann."*
- *„Einige kritische Fehlerquellen wurden mir besser verständlich."*
- *„Fehlererkennen ist noch nicht das Fehlerbeseitigen. Dies ist eine zusätzliche Schwierigkeit."*
- *„Das Lesen und Verstehen fremder Programme war eine ganz eigene, neue Herausforderung"*
- *„bringen Abwechslung in den Unterricht."*
- *„Selbst programmieren ist interessanter."*

Danke !

Fragen?

Unterrichtsmaterialien

Bildungsserver Rheinland-Pfalz

bildung-rp.de/

Suchbegriff: „Debugging-Aufgaben“

infodaktik.de (ab Sommer 2016)

Quellenverzeichnis

[1] http://www.tagesspiegel.de/wirtschaft/ec-karten-panne-zum-jahreswechsel/1658102.html
[2] http://de.wikipedia.org/wiki/Programmfehler#Folgen_von_Programmfehlern
[3] http://www.spiegel.de/politik/ausland/airbus-a400m-militaermaschine-stuerzte-wegen-software-problemen-ab-a-1034421.html