Dr. Hanno Schauer
Mons-Tabor-Gymnasium
Montabaur

Debugging-Aufgaben

Programmfehler als Lernchance

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

Ansatz

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

Allerdings: Nicht alle Hoffnungen werden gleichermaßen erfüllt

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

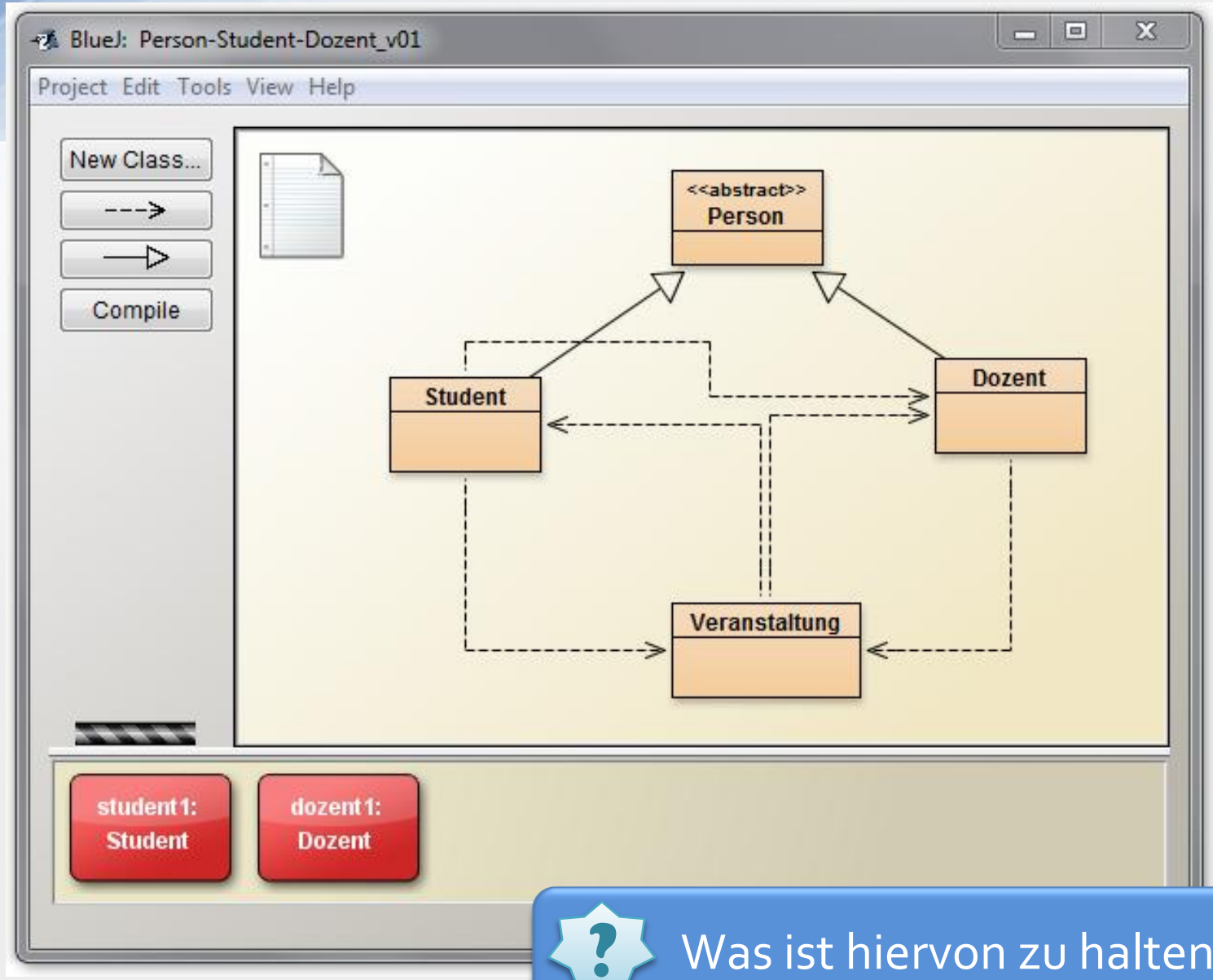
3) **Fraglichen Quelltextausschnitt analysieren**

4) **Fehler korrigieren**

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

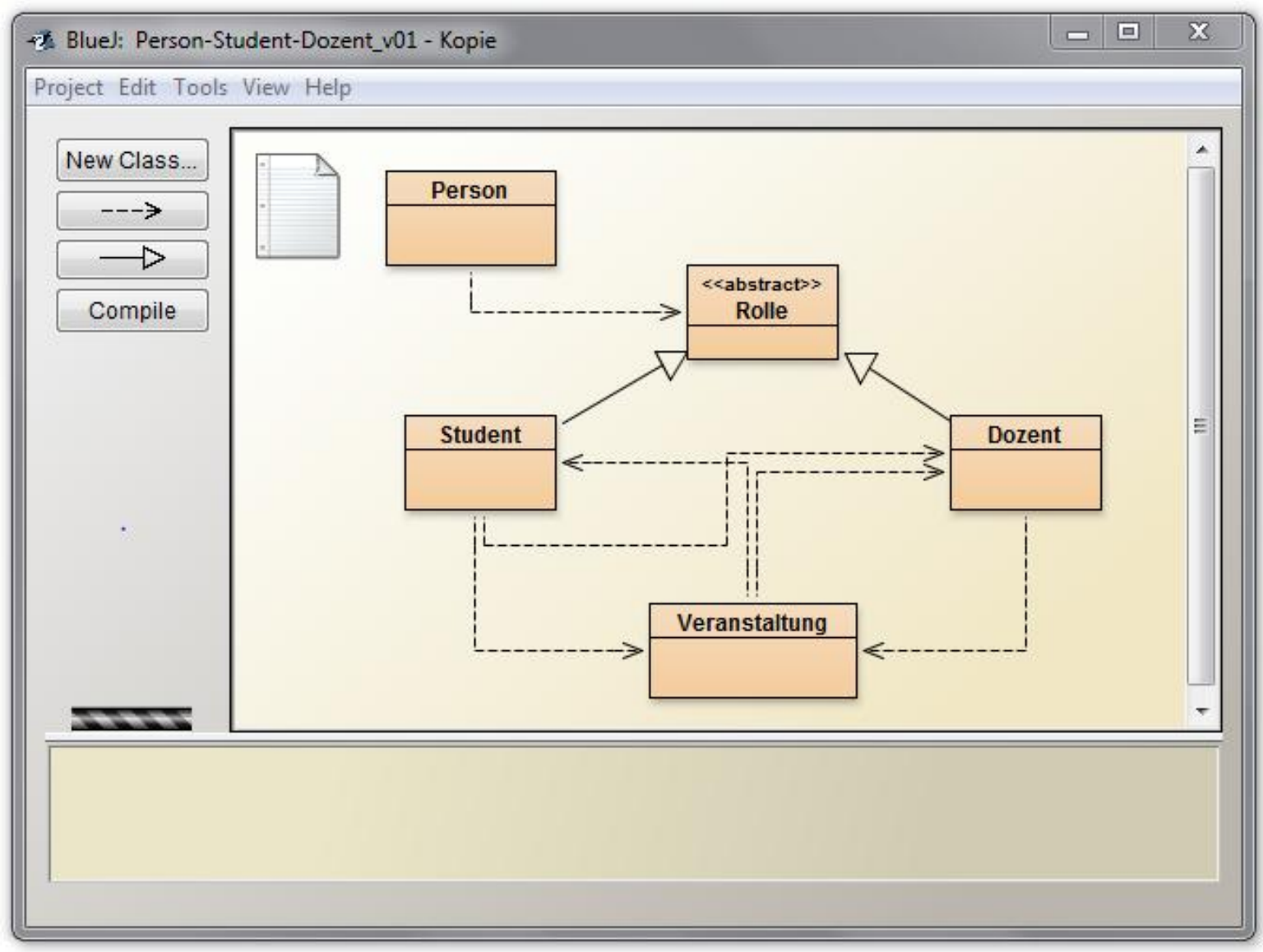
Fokus auf Semantik- und Laufzeitfehler

Ein Beispiel



Was ist hiervon zu halten?

… dann doch besser so …



Anwendungsgebiete

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

„Erfolgsfaktoren“ der Aufgabengestaltung

- **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
- **Struktur – Struktur – Struktur**
  - Gut strukturierter (modularisierter) und kommentierter Quellcode,
  - 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

Didaktische Einordnung

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

Die Informatik-Didaktik-Community ist gefordert, geeignete Aufgaben zu erstellen und zu publizieren!

Danke !

Fragen?

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."*