Karlsruher Institut für Technologie

Vorlesung Programmieren

15. Finden und Beheben von Fehlern

PD Dr. rer. nat. Robert Heinrich, Prof. Dr.-Ing. Anne Koziolek

Beratung und Auskunft

Informatik Studiengangservice: ISS- Studienberatung, Verlängerungsanträge, Studienplanänderung, usw.

ZSB - Zentrale Studienberatung

- Zweifel im Studium, Studiengang-/Hochschulwechsel, Studienabbruch, usw.

Studierendenwerk

- Allgemeine Beratung unterstützt bei Themen wie BAföG, Wohnheimsplätze, Sozial-, Rechts- & Psychotherapeutische Beratung

AStA – Allgemeiner Studierendenausschuss

- Verschiedene kostenlose Beratungsangebote Inkl. Ausländische Studierende, Sozialberatung, Chancengleichheit, Queerberatung

Beratung und Auskunft : Stressbewältigung

Bei Stress in Studium oder persönliche Schwierigkeiten können Sie sich an folgenden Ansprechstellen wenden:

PBS – Psychotherapeutische Beratungsstelle:
bieten Gespräch, Info-Text, usw… für
Stress, Prüfungsangst, Einsamkeit, usw.

PBS Karlsruhe
Rudolfstraße 20
76131 Karlsruhe
pbs@sw-ka.de
Info und Anmeldung:
Mo.-Fr., jeweils 9:00-12:00 Uhr

Telefon: +49 721 9334060

- https://www.sw-ka.de/de/beratung/psychologisch/weitere-angebote/**infotexte/**

Beratungsstelle Brücke: Krisenbegleitung und Seelsorge
auch spontan am Kronenplatz in Karlsruhe an, auch auf Englisch

Nightline Karlsruhe: Vertrauliches Telefon
von Studierenden für Studierende

Telefonseelsorge: Kostenlos, rund um die Uhr erreichbar

Lernlabor@ilias von HoC

Organisieren und Planen	Recherche und Textbearbeitung	Struktur und Gedächtnis	Motivation und Selbstregulation	Entspannung und Bewegung
To Do-Liste, Tagesplan, Masterplan erstellen	Internetrecherche (Suchmaschinen)	Mind-Mapping	Selbstmotivierung	Bewegung und Lernen
Meilensteinplanung	Kooperatives Lesen	Strukturlegetechnik	Pomodoro Technik	Bewegungspausen gestalten
Planung mit der ALPEN-Methode	Markierungstechniken	Lernposter erstellen	Prokrastination Selbsttest	Jonglage
Planung mit dem Kanban-Kalender	Lesetechniken	Mnemotechnik: Schlüsselwort-Methode	Aufschiebetagebuch führen	Stressbewältigung
Prioritäten setzen mit der Eisenhower-Matrix	Lesemethode PQ4R (für komplexe Texte)	Mnemotechnik: Zahlentafel/Gedächtnistafel	Umgang mit Lampenfieber	Entspannungsübungen
SMARTe Lernziele setzen	Speed Reading/ Schnelles Lesen (Selbsttest)	Mnemotechnik: Akronyme	Survival Guide Prüfungsangst	Achtsamkeit
Pausenplanung		Mnemotechnik: Loci-Methode	Survival Guide Gruppenarbeit	Powernapping
Lernplan erstellen		Mnemotechnik: Karteikarten-Methode	Umgang mit Erfolg und Misserfolg	
Lerntagebuch schreiben		Vorlesungsmitschrieb, Vor- und Nachbereitung	Emotionsregulation: Das ABC-Schema	
Der häusliche Arbeitsplatz		Vorlesungsmitschrieb: Cornell Methode	Ressourcen aktivieren	
Strukturiert im Homeoffice		Deep Work	Selbstregulierung mit der Affektbilanz	
Survival Guide Prüfungsphase		Hubschrauber-Landkarten-Methode	Selbststeuerung mit dem Wenn-Dann-Plan	
Eliminierungsmethoden zur Prüfungsvorbereitung		Mental Methode	Zielvisualisierung	
Premortem-Analyse		K-R-Methode		
Übersicht zu LernApps	Übersicht zu LernApps	Übersicht zu LernApps	Übersicht zu LernApps	Übersicht zu LernApps

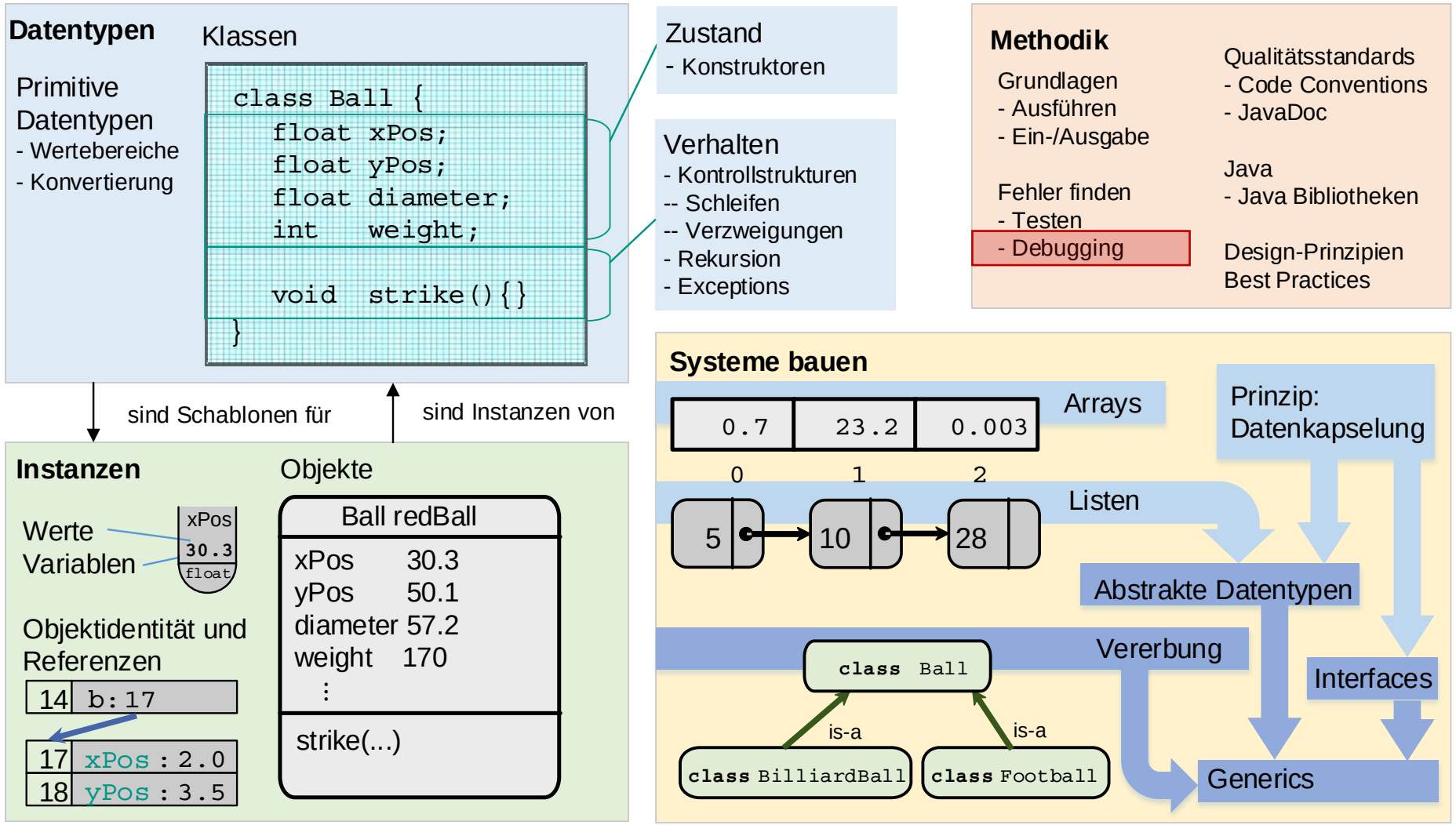
Aufbaukurs Informatik 1 e/h (Grundlagen Java) am MINT-Kolleg

Zum Wiederholen des Vorlesungsstoffes der Vorlesung „Programmieren“ für den Studiengang (Wirtschafts-)Informatik.

Termine: 21.03.2024 bis 28.03.2024
6 Termine, jeweils 9:00 - 12:15 Uhr

- Der Kurs findet als **Präsenzveranstaltung** statt. Es werden die wichtigsten Inhalte der Vorlesung behandelt und an Programmieraufgaben geübt.
- **Kostenloses** und **freiwilliges** Angebot für Studierenden am KIT mit individueller Betreuung in kleinen Gruppen.
- Die Teilnehmerzahl ist **begrenzt**. Anmeldung unter: **www.mint-kolleg.kit.edu**

Vorlesungsüberblick: Objekt-orientiertes Programmieren in Java



Vorlesungsüberblick: Vorläufiger Semesterplan

23.10.2023	Erstsemesterbegrüßung: Einführung
25.10.2023	Organisatorisches; Ein Einfaches Programm, Objekte und Klassen
08.11.2023	Typen und Variablen
15.11.2023	Kontrollstrukturen (+Scanner)
22.11.2023	Konstruktoren und Methoden
29.11.2023	Arrays; Konvertierung, Datenkapselung, Sichtbarkeit
06.12.2023	Listen und Abstrakte Datentypen
13.12.2023	Vererbung
20.12.2023	Exceptions; Interfaces
10.01.2024	Generics; Rekursion
17.01.2024	Java-API; Objektorientierte Design-Prinzipien
24.01.2024	Best Practices;
31.01.2024	Finden und Beheben von Fehlern, Testen und Assertions (Teil 1)
07.02.2024	Testen und Assertions (Teil 2), Junit; Parsen, Suchen, Sortieren
14.02.2024	Vom Programm zur Maschine; Ausblick auf zukünftige Lehrveranstaltungen

Lernziele

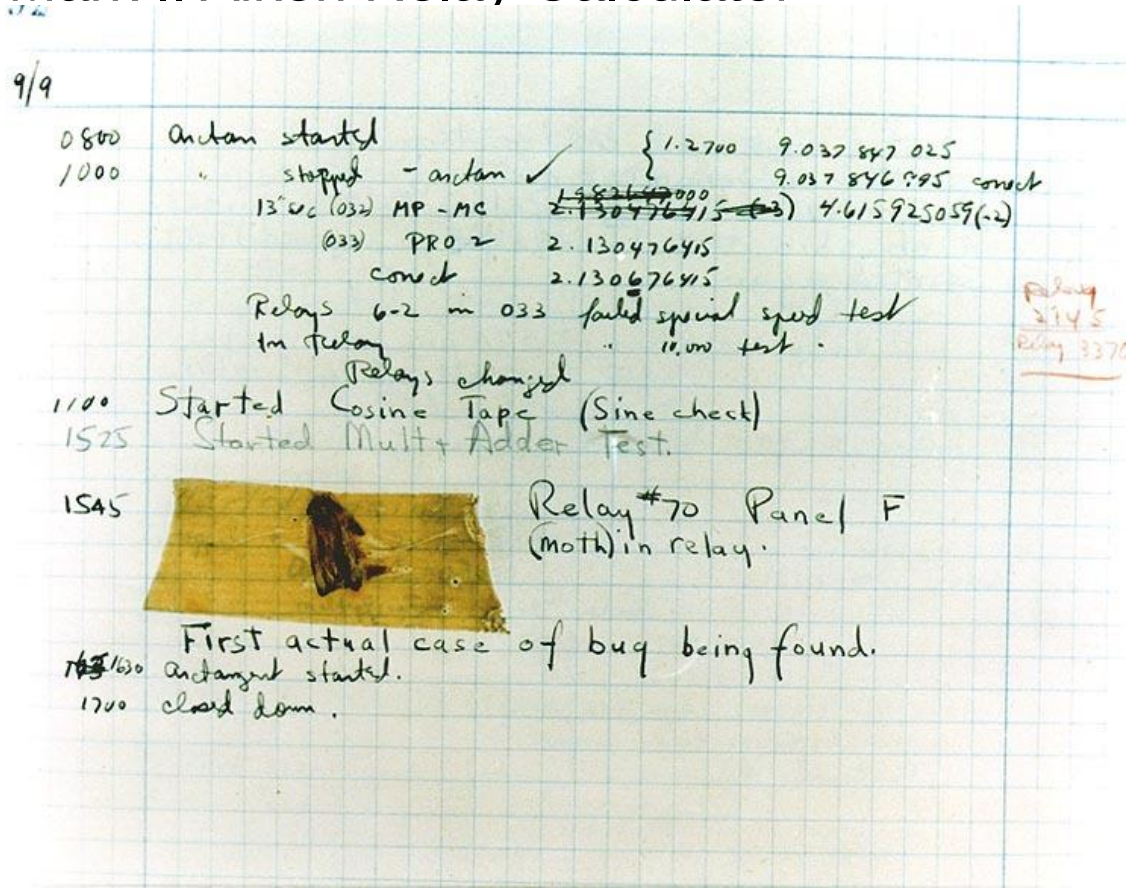
Finden und Beheben von Fehlern

- Sie kennen die Definition von Software-Fehlern
- Sie können durch systematische Analyse in Programmen Fehler finden.
- Sie können wissenschaftliche Methoden beim Debugging anwenden.

Quelle: http://phdcomics.com

Wiederholung: »Bugs«

Mark II Aiken Relay Calculator



Grace Hopper (1906–1992)

Der Begriff »Bug«

„Wenn Debugging der Vorgang ist, Fehler auszubauen,
dann ist Programmieren wohl der Vorgang, Fehler einzubauen.“

— (unbekannt)

- Sonderfall *Heisenbug:* verschwindet beim Debugging
- *Bohrbug*: „normaler“ Bug (deterministisch)

Fault – Defect – Bug – Error– Failure

- **Fault:** Ein Fehler bei der Entwicklung
- **Defect:** Ein Fault, der durch Testen gefunden wurde.
- **Bug:** Ein Defect, der von den Entwicklern anerkannt wurde.
- **Error:** Software-Zustand, der durch einen Fault, Bug oder Defect verursacht wurde.
- **Failure:** Unterschied zum von der Software erwarteten Verhalten zur Laufzeit (z.B. Absturz, Ausfall)

Definitionen Fault – Error – Failure:

Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, *1*(1), 11-33. DOI 10.1109/TDSC.2004.2

Von »Fault« zu »Failure«

```
// Specification: Search for element x in array a, starting at index p.
// The method returns the smallest i >= p, for which a[i] == x.
// If no such i exists, -1 is returned.
static int search(int[] a, int p, int x) {
  for (int i = 0; i <= a.length; i++) {
    if (a[i] == x) {
      return p;
    }
  }
  return -1;
}
static void test_1() {
  // Test case: element to search for occurs multiple times
  int[] a = { 2, 5, 4, 3, 2, 3 };
  int p = 4, x = 3;
  int r = search(a, p, x);
  assert r == 5;
}
```

Fault 1: Richtig wäre: `i = p`

Fault 2: Richtig wäre: `return i`

Fault 3: Richtig wäre: `i < a.length`

Failure: `r = 4`

Debugging – Definition

- **Debugging** besteht aus Finden und Beheben von Fehlern.
- Normalerweise machen Finden und Verstehen eines Fehlers 90 % des Debugging-Aufwandes aus. (→ Korrektur einfacher)

- Ein effizienter Programmierer kann durch wissenschaftliche Methoden die Fehler finden, während ein ineffizienter Programmierer unsystematisch nach den Fehlern sucht.
- Dies ist ein wesentlicher Faktor für die unterschiedliche Produktivität von Software-Entwicklern (ca. 1 zu 20).

Schritte der klassischen wissenschaftlichen Methode

- Erfassen der Daten durch reproduzierbare Experimente
- Formulieren einer Hypothese, die die relevanten Daten widerspiegelt.
- Entwerfen eines Experimentes zur Überprüfung der Hypothese
- Überprüfung der Hypothese
- Bei Bedarf das Experiment wiederholen oder weiterentwickeln durch verfeinerte Hypothese

Wissenschaftliche Methode angewandt auf Debugging

- **Stabilisieren** des Fehlers (»failure«)
- **Lokalisieren** der Ursache des Fehlers (»fault«)
  - Erfassen der Daten, die zum Fehler führen
  - Analyse der erfassten Daten
  - Formulieren der Hypothese über den Defekt
  - Identifizieren einer Methode zur Überprüfung der Hypothese
    - Testen
    - Durchsuchen des Codes
  - Überprüfen der Hypothese durch eine der beiden Methoden
- **Beheben** des Fehlers
- **Testen**, ob der Fehler tatsächlich behoben wurde
- Suche nach **ähnlichen Fehlern**

Debugging – Stabilisieren des Fehlers (1)

- Schwerpunkt: Reproduzierbarkeit eines Fehlers
- Stabilisieren des Fehlers → Einfache Fehlerdiagnose
- Ein Fehler, der nicht vorhersagbar ist, ist normalerweise
  - ein Initialisierungsfehler (selten in Java)
  - ein zeitliches Problem
  - ein hängender Zeiger (kommt in Java nicht vor).

Debugging – Stabilisieren des Fehlers (2)

Das **Stabilisieren** des Fehlers besteht aus

- Finden eines Testfalls, der zum Fehler führt
- Vereinfachen des Testfalls zum einfachsten Testfall, der zum Fehler führt.

Vereinfachen eines Testfalls bedeutet, dass Ändern des kleinsten Aspekts des Testfalls zum Ändern des Verhaltens des Fehlers führt.

Wichtig: Getestete Fälle mit Ergebnis dokumentieren!

Debugging – Lokalisieren der Fehlerursache (1)

- Ermitteln der Testdaten, die zum Fehler führen
- Analyse der ermittelten Daten
- Formulieren einer Hypothese über die Fehlerursache basierend auf den Daten
- Überprüfung der Hypothese durch einen Testfall oder Inspektion des Codes

Debugging – Lokalisieren der Fehlerursache (2)

Beispiel: Hypothese über Einflussfaktoren:

- Es gibt 10 Faktoren, die zusammen zu einem Fehler führen
- Hypothese über irrelevanten Faktoren
- Ändern des jeweiligen irrelevanten Faktors, Beobachten des Verhaltens
  - Fehler tritt immer noch auf → Der Faktor kann tatsächlich eliminiert werden. → Der Testfall ist vereinfacht.
  - Fehler tritt nicht mehr auf → Der Faktor ist relevant, und die Hypothese muss korrigiert werden.

Beispiel: Hypothese über Wertebereich:

- Hypothese: Der Fehler liegt innerhalb eines Wertebereichs
- Zur Überprüfung der Hypothese müssen drei Werte überprüft werden
  - innerhalb des Wertebereichs
  - die Grenzen des Wertebereichs
  - außerhalb des Wertebereichs

Zwischenfazit

Debugging besteht aus Finden und Beheben von Fehlern.

Einführung einer wissenschaftlichen Methode für Debugging:

- **Stabilisieren** des Fehlers
- **Lokalisieren** der Ursache des Fehlers (»fault«)
- Beheben des Fehlers
- Testen, ob der Fehler tatsächlich behoben wurde.
- Suche nach ähnlichen Fehlern

Tipps für die Fehlersuche (1)

Einsatz aller verfügbaren Daten für die Formulierung der Hypothese

- Grund: Eine Hypothese betrifft nicht nur einen Fall, sondern eine Klasse von Fällen
- Beispiel:
  - Hypothese: alle Wörter, die mit T anfangen, können in der Datenbank nicht richtig sortiert werden.
  - Die Hypothese betrifft die Klasse der Wörter, die mit T anfangen
- Werden Daten gefunden, die gegen die Hypothese sprechen
  - → Die Daten nicht ignorieren
  - → Hypothese korrigieren
- Seien zwei Hypothese A und B gleichzeitig richtig und Hypothese A sei allgemeiner als Hypothese B.
  - → Es ist effizienter, zunächst nur die präzisere Hypothese zu überprüfen.

Tipps für die Fehlersuche (2)

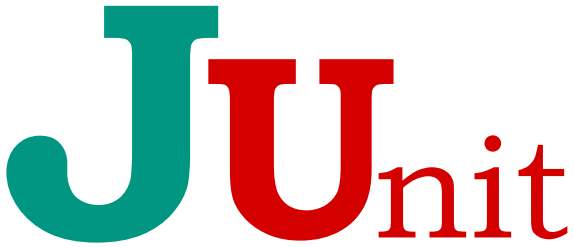
Verfeinerung der Testfälle, die zum Fehler führen

- Kann die Fehlerursache nicht durch die Testfälle identifiziert werden:
  - Die Testfälle müssen weiter verfeinert werden.
  - → Die Parameter müssen jeweils mehr verändert werden, als bisher angenommen.
  - → Das Programmverhalten muss dabei beobachtet werden.

Tipps für die Fehlersuche (3)

Benutzen von Unit-Tests

- Fehler können einfacher in kleinen Programmabschnitten gefunden werden
- Mit Hilfe von Unit-Tests können verschiedene Programmabschnitte in Isolation beobachtet werden.
- JUnit bietet fortgeschrittene Methoden für das Schreiben von Unit-Tests mit Java (mehr dazu nächste Woche)



Tipps für die Fehlersuche (4)

Benutzen vorhandener Werkzeuge

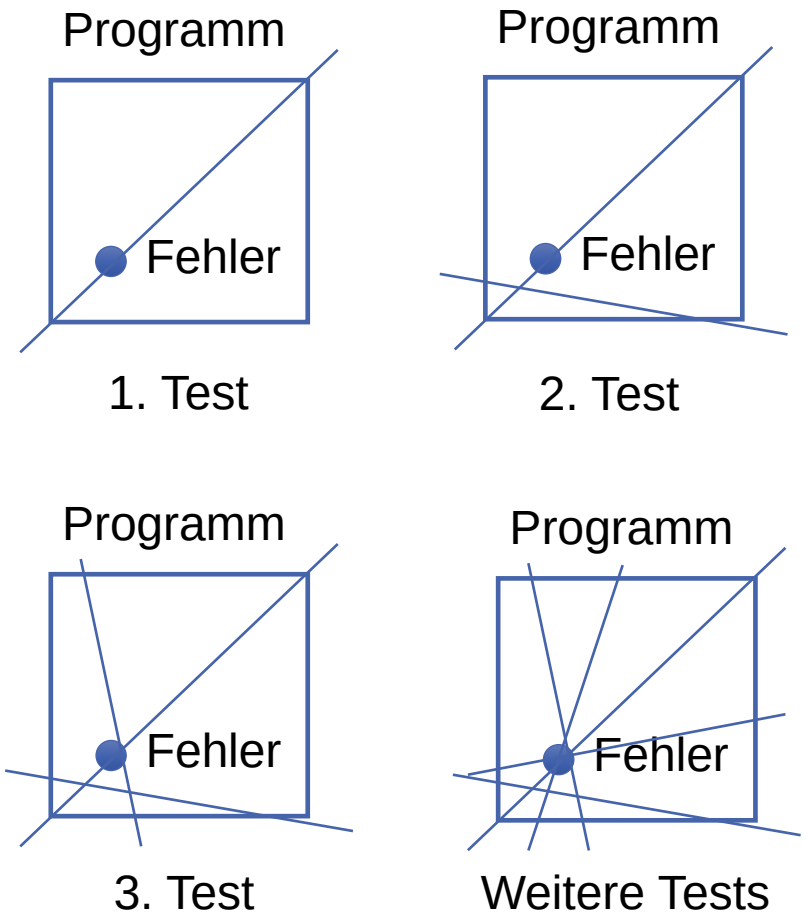
- Es gibt bereits diverse Werkzeuge zur Fehlersuche
- Beispiele:
  - interaktive Debugger
  - Compiler
  - Werkzeuge zur Überprüfung der Werte in den Speicherzellen

- Beispiel für einen Fall, bei dem ein solches Werkzeug nützlich ist:
  - Ein Teil des Programms überschreibt die Ergebnisse anderer Programmteile
  - Analytisch schwer durch den Programmierer ohne Werkzeug überprüfbar
  - Mit Hilfe eines Debuggers kann ein Checkpoint für die bestimmte Speicheradresse gesetzt werden.

Tipps für die Fehlersuche (5)

Reproduzieren eines Fehlers auf verschiedene Arten

- Der Fehler soll durch mehrere nicht identische Testfälle lokalisiert werden
- Fehlerursache kann so am besten identifiziert werden.
- Abschluss: Konstruieren eines weiteren Testfalls, der nicht zu einem Fehler führen sollte.
  → Dieser führt bei Ausführung des Testfalls zu
  - einem Fehler → Die Fehlerursache wurde noch nicht identifiziert.
  - keinem Fehler → Die Fehlerursache wurde eventuell korrekt identifiziert.
- Vorsicht bei Fehlern, die mehr als eine Fehlerursache haben.



Tipps für die Fehlersuche (6)

Generieren von weiteren Daten bei der Formulierung der Hypothese

- Generieren von zusätzlichen Testfällen, die sich von Testfällen unterschieden, bei denen ersichtlich ist, dass sie fehlschlagen.
- Diese Testfälle dienen zur Generierung von zusätzlichen Daten
- Benutzen der neuen Daten zum Verschärfen der Hypothese

Tipps für die Fehlersuche (7)

Benutzen von negativen Tests

Beispiel:

- Ein Test widerlegt eine Hypothese

→Die Fehlerursache liegt nicht an einer anderen Stelle im Programm

→Diese Information hilft beim Einschränken der Suche

Tipps für die Fehlersuche (8)

Brainstormen über mögliche Hypothesen

- Nicht nur eine, sondern viele mögliche Hypothesen formulieren
- Alle Hypothesen werden zunächst formuliert.
- Überprüfen von Hypothesen durch Konstruieren von Testfällen

Tipps für die Fehlersuche (9)

Erstellen einer Liste aller Methoden/Aktivitäten für Debugging

- Ist eine Methode nicht erfolgreich, wird die nächste Methode ausprobiert.

Tipps für die Fehlersuche (10)

Eingrenzen der Fehlerregion im Code

- Statt das gesamte Programm zu testen, werden nur Teile vom Programm getestet
- 1. Methode: Eingrenzen der Fehlerregion durch `print`-Anweisungen, Log- und Tracing-Daten
- 2. Methode: Teile des Programms nacheinander entfernen
- Analog zur binären Suche immer die Hälfte des Codes
- Beobachten des Programmverhaltens:
  - Tritt der Fehler immer noch auf, liegt der Fehler im Code-Bereich, der noch nicht entfernt wurde.
  - Tritt der Fehler nicht mehr auf, liegt der Fehler im Code-Bereich, der entfernt wurde.
  - Beispiel: Auskommentieren des Codes, Breakpoints mit Debugger

Tipps für die Fehlersuche (11)

Vorsicht bei Klassen, die bisher Fehler enthielten

- Klassen und Routinen, die in der Vergangenheit fehlerhaft waren, sind mit hoher Wahrscheinlichkeit auch in Zukunft fehlerhaft.
- Fehleranfällige Klassen müssen erneut überprüft werden.

Tipps für die Fehlersuche (12)

Überprüfen sich häufig ändernden Codes

- Wenn es einen schwer auffindbaren Fehler im Code gibt, liegt der Fehler:
  - normalerweise im neu geschriebenen Code
  - oder in der Differenz zwischen dem alten und dem neuen Code
    - Diff-Werkzeuge
    - Version Control Logs
- Ausführen des alten Codes und Beobachten des Programmverhaltens

→ Versionskontrolle (z.B. git) nutzen

Tipps für die Fehlersuche (13)

Erweitern der zu beobachtenden Coderegion

- Beginn der Suche mit einer kleinen Coderegion
- Liegt der Fehler nicht in der Coderegion
  → Erweitern der Coderegion mit Hilfe der binären Suche

Tipps für die Fehlersuche (14)

Inkrementelle Integration

- Integration kleiner Programmstücke
- Ist der Code nach der Integration des Programmstücks fehlerhaft
  → Entfernen und Testen des neuen Programmstücks

Tipps für die Fehlersuche (15)

Überprüfen auf häufig vorkommende Fehler

- Anlegen einer feingranularen Checkliste zur Qualitätskontrolle, angepasst an die Entwicklungsumgebung
- Anwenden der bereits existierenden Checklisten

Tipps für die Fehlersuche (16)

Sprechen mit einem Dritten über das Problem

- Der Programmierer kann seine Fehler am besten entdecken, wenn er darüber mit einem Dritten spricht.
- Dritte besitzen Distanz zum Problem

- Ähnliche Option: KI-Assistenten fragen
  - Das Problem schildern (was will ich erreichen, was geht schief)
  - Ggf. Fehlermeldungen ergänzen
  - Kostenfrei verfügbare KI-Assistenten: https://chat.openai.com, https://bard.google.com https://huggingface.co/chat/ (Open-Source-Alternative)
  - Aber: Keinen KI-generierten Code abgeben

Tipps für die Fehlersuche (17)

Eine Pause machen!

- Eine Pause hilft oft bei der Fehlersuche.

Tipps für die Fehlersuche (18)

Brute-Force-Debugging ist ein Verfahren, das wenig beachtet wird.

- Verfahren ist schwierig, anstrengend und zeitaufwendig, garantiert jedoch die Fehlerbehebung
- Es gibt diverse Techniken, die fallabhängig eingesetzt werden können:
  - Organisieren eines vollständigen Entwurfs und/oder Code-Review des fehlerhaften Codes
  - Ignorieren des fehlerhaften Codes. → Neuentwurf und Neuprogrammieren
  - Ändern des Warning-Levels beim Compilieren und Korrigieren aller Warnings
  - Unit Tests für den neugeschriebenen Code
  - Schreiben einer automatischen Test Suite. Langes Ausführen der Tests
  - Manuelles Durchlaufen einer großen Schleife mit Hilfe vom Debugger
  - Instrumentieren des Codes
  - Replizieren der kompletten Konfiguration des Endbenutzers
  - Integration kleiner komplett getesteter Programmstücke

Syntaxfehler

- Nie Bezeichner mit nur einem Zeichen Differenz wählen (z.B. `x-coord`, `y-coord`), denn so können Tippfehler leicht wieder zu legalen Bezeichnern führen.

- Tipps beim Benutzen von Compilern:
  - Verlassen Sie sich nicht unbedingt auf die Zeilennummer in der Compilernachricht.
  - Verlassen Sie sich nicht unbedingt auf die Compilernachricht.
  - Verlassen Sie sich nicht unbedingt auf die sekundären Nachrichten einer Fehlerkaskade
  - Divide and Conquer

Zusammenfassung

Debugging besteht aus Finden und Beheben von Fehlern.

Einführung einer **wissenschaftlichen Methode** für Debugging

- Stabilisieren des Fehlers
- Lokalisieren der Ursache des Fehlers (auch »fault« genannt)
- Beheben des Fehlers
- Testen, ob der Fehler tatsächlich behoben wurde.
- Suche nach ähnlichen Fehlern

Tipps für die Fehlersuche

- Benutzen aller verfügbaren Daten für die Formulierung der Hypothese
- Verfeinerung der Testfälle
- Benutzen von Unit-Tests
- Benutzen vorhandener Werkzeuge
- Reproduzieren eines Fehlers auf verschiedene Art und Weise
- Generieren von mehr Daten
- Benutzen der negativen Tests
- Brainstormen über Hypothesen
- Brute Force