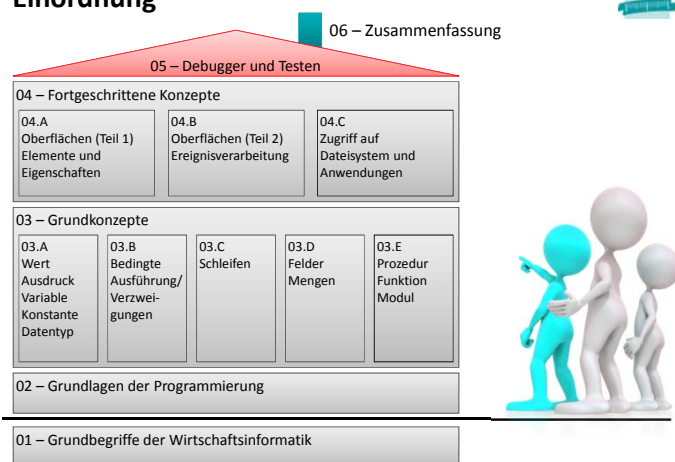
 BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Wirtschaftsinformatik 1

LE 10 – Fehler, Debugger und Testen

Prof. Dr. Thomas Off
<http://www.ThomasOff.de/lehre/beuth/wi1>

Einordnung



06 – Zusammenfassung

05 – Debugger und Testen

04 – Fortgeschrittene Konzepte

04.A Oberflächen (Teil 1) Elemente und Eigenschaften	04.B Oberflächen (Teil 2) Ereignisverarbeitung	04.C Zugriff auf Dateisystem und Anwendungen
---	--	---

03 – Grundkonzepte


03.A Wert Ausdruck Variable Konstante Datentyp	03.B Bedingte Ausführung/ Verzwei- gungen	03.C Schleifen	03.D Felder Mengen	03.E Prozedur Funktion Modul
---	---	-------------------	--------------------------	---------------------------------------

02 – Grundlagen der Programmierung

01 – Grundbegriffe der Wirtschaftsinformatik

LE10 - Fehler, Debugger und Testen 3

Inhalt



Einordnung
Rückblick
Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern
- Arten von Fehlern

Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

Debugger

- Zweck
- Einsatzmöglichkeiten



Testen

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

Abschluss und Ausblick

LE10 - Fehler, Debugger und Testen 4

Rückblick



LE10 - Fehler, Debugger und Testen 5

Rückblick

Zugriff auf Dateisystem

- mit Modul "FileSystem" grundlegende Möglichkeiten
 - Elemente auflisten

```
' Generelle Syntax mit Angabe des gewünschten Inhalts
' z.B. vbDirectory, vbHidden, vbSystem
Let <strElement> = FileSystem.Dir(<Pfad>, <GewünschteInhalte>)
Let <strElement> = FileSystem.Dir() ' Nächstes (im vorherigen Pfad)
```

- Weitere: Verzeichnisse anlegen, löschen, ...

- mit FileSystem-Klasse aus MS Scripting Runtime bestehen weitergehend Möglichkeiten z.B.
 - Zugriff auf Laufwerksinformation,
 - Kopieren von Verzeichnissen

LE10 - Fehler, Debugger und Testen 6

Rückblick

Dialoge zur Auswahl von Dateien und Verzeichnissen

- sind sinnvoll, wenn vom Benutzer das Ziel zum Speichern oder Laden von Daten im Dateisystem selbst gewählt werden soll

Generelle Syntax

- Deklaration und Initialisierung


```
Dim <FileDialogObj> As Object
Set <FileDialogObj> = Application.FileDialog(<Zahl>)
```
- Konfiguration (z.B. Mehrfachauswahl)


```
<FileDialogObj>.AllowMultiSelect = True
```
- Anzeige


```
Let <intVar> = <FileDialogObj>.Show() ' Rückgabewert 0 = Abbruch
```
- Ergebnis in Collection "SelectedItems" enthalten


```
<FileDialogObj>.SelectedItems
```

LE10 - Fehler, Debugger und Testen 7

Rückblick

Generelle Syntax (Fortsetzung)

- Ergebnis in Collection "SelectedItems" enthalten


```
<FileDialogObj>.SelectedItems
```

Beispiel für Standarddialog zur Dateiauswahl

```
Dim intResult As Integer ' Rückgabewert
Dim i As Integer ' Schleifenvariable
Dim oFd As Object ' Variable für FileDialog
Set oFd = Application.FileDialog(3) ' Initial. als Dateiauswahl = 3

oFd.AllowMultiSelect = True ' Konfiguration, z.B. Mehrfachauswahl
Let intResult = oFd.Show ' Dialog anzeigen und Ergebnis merken

If intResult = 0 Then
    Exit Sub ' Abbruch durch Benutzer
End If

' Schleife über alle ausgewählten Dateien
For i = 1 To oFd.SelectedItems.Count
    Debug.Print oFd.SelectedItems(i)
Next
```

LE10 - Fehler, Debugger und Testen 8

Rückblick

Früheste Darstellung des Dateizugriffs in Basic

- 1 Besorge freie Dateinummer
- 2 Öffne Datei mit Pfad [altägypt. für Fuß]
- 3 Gib an, ob gelesen...
- 4 ... geändert oder ...
- 5 ... neu geschrieben werden soll.
- 6 Lies, schreibe oder ändere
- 7 Schließe die Datei am Ende
- 8 Gib [immer] die Dateinummer an!

Quelle der Abbildung: [1]

... seither nahezu unverändert

LE10 - Fehler, Debugger und Testen 9

Rückblick

Zugriff auf Dateien

- Freie Dateinummer ermitteln
- Öffnen einer Datei (verkürzte Form)
 - Pfad: Angabe des Pfades zur Datei
 - Modus: Lesen (Input), Schreiben (Output), Ändern (Append), ...
 - Zugriff: Lesen (Read), Schreiben (Write) oder Ändern (ReadWrite)
 - Dateinummer: Zuvor mit FreeFile() ermittelte Nummer
- Schließen einer Datei

Generelle Syntax

```
Let <intVar> = FileSystem.FreeFile()

Open <Pfad> For <Modus> Access
<Zugriff> As #<DateiNr>

Close #<DateiNr>
```

Beispiel

```
' Freie Nummer für Dateizugriff
Let intFNR = FileSystem.FreeFile()

' Datei öffnen (zum Schreiben)
Open "C:\Temp\doc3.txt" For Output _
Access Write As #intFNR

' Datei verwenden
' ...

Close #intFNR ' Datei schließen
```

LE10 - Fehler, Debugger und Testen 10

Rückblick

Zugriff auf Dateien (Forts.)

- Datei schreiben (Write)


```
Write #<DateiNr>, <WertAusdr> ' Variante 1
Write #<DateiNr>, <WertAusdr1>, <WertAusdr2>, ... ' Variante 2
Write #<DateiNr>, ' Leere Zeile
```
- Datei lesen (Input)


```
Input #<DateiNr>, <Variable> ' Variante 1
Input #<DateiNr>, <Var1>, <Var2>, ... ' Variante 2
```
- Datei zeilenweise lesen (Line Input)


```
Line Input #<DateiNr>, <StringVariable>
```

Beispiel (Schreibzugriff)

```
' ...
' Beispiel 1 schreiben (mit Variablen)
Write #intFNR, strName, datGebDat, bolGeschlecht
' Beispiel 2 schreiben (hier auch Typumwandlung sinnvoll)
Write #intFNR, "Ali Yilmaz", CDate("19.05.1987"), False
' ...
```

LE10 - Fehler, Debugger und Testen 11

Rückblick

Wichtige Oberflächenelemente

- Eingabe und Auswahl
 - Textfelder
 - Aufklappliste/Kombinationsfeld
 - Mehrfachauswahllisten
 - Radioknöpfe (Optionsfeld)
 - Kontrollkästchen (Checkbox)
- Aktionselemente
 - Schaltfläche (einfach)
 - Umschaltfläche (Toggle)
 - Menüeinträge
- Container
 - Rahmen/Gruppen
 - Registerkartensatz mit Registerkarten
 - Menüs
 - Fenster/Dialoge (in Access als Formulare)

LE09 - Dateisystem und Dateizugriff 12

Rückblick

Ereignisprozeduren bieten Zugriffsmöglichkeit auf die Oberflächenelemente und dienen zum

- Steuern der Elemente auf der Oberfläche
 - Aktivieren/Dekativieren bzw. Einblenden/Ausblenden von Elementen
 - Navigation zwischen Fenstern
 - ...
- Aufruf der Verarbeitungslogik
 - Übergabe der eingegebenen Daten zur Verarbeitung
 - Ermitteln der anzuzeigenden Daten
 - Ausführen von komplexen Berechnungen
 - ...

LE09 - Dateisystem und Dateizugriff 13

Rückblick



Referenzvariable Me stellt Funktionen zur Verfügung

- Zugriff auf den Wert von Feldern liefert immer String

```
' Generelle Syntax  
Let <VarString> = Me.<BezeichnerDesFeldes>.Value  
Let <VarZahl> = Val(Me.<BezeichnerDesFeldes>.Value)
```

- Genereller Zugriff auf Eigenschaften von Elementen

```
' Generelle Syntax  
Let <Var> = Me.<Bez>.<Eigenschaft> ' Lesen  
Let Me.<Bez>.<Eigenschaft> = <Var> ' Schreiben/Ändern
```

Beispiele

```
Let strName = Me.txtName.Value  
Let intAlter = Val(Me.txtAlter.Value)  
  
Let Me.txtName.Visible = False
```

LE09 - Dateisystem und Dateizugriff 14

Rückblick



Kommando DoCmd stellt Funktionen zur Verfügung

- Generelle Syntax zum Öffnen von Fenstern

```
' Generelle Syntax (Auszug)  
DoCmd.OpenForm <Formularname>
```

- Generelle Syntax zum Schließen von Fenstern

```
' Generelle Syntax (Auszug)  
DoCmd.Close <TypZuSchließendesObjekt>, <Name>  
' Syntax zum Schließen von Formularen  
DoCmd.Close acForm, <Formularname>
```

- Generelle Syntax zum Navigieren zwischen Fenstern

```
' Generelle Syntax (Auszug)  
DoCmd.BrowseTo <TypZielObjekt>, <Name>  
' Syntax zum Schließen von Formularen  
DoCmd.BrowseTo acBrowseToForm, <Formularname>
```

LE09 - Dateisystem und Dateizugriff 15

Inhalt



Einordnung

Rückblick

Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern

Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

Debugger

- Zweck
- Einsatzmöglichkeiten

Testen


- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

Abschluss und Ausblick



LE10 - Fehler, Debugger und Testen 16

Inhalt



Einordnung

Rückblick

Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern

Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

Debugger

- Zweck
- Einsatzmöglichkeiten

Testen

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

Abschluss und Ausblick

LE10 - Fehler, Debugger und Testen 17

Inhalt

Einordnung

Rückblick

Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern

Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

Debugger

- Zweck
- Einsatzmöglichkeiten

Testen

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

Abschluss und Ausblick

LE10 - Fehler, Debugger und Testen

18

Beispiele für Fehler



LE10 - Fehler, Debugger und Testen

19

Beispiele für Fehler

Rakete Ariane 5

- sprengte sich selber wegen eines Softwarefehlers: Überlauf aus einem Wertebereich
- Software vom Vorgängermodell Ariane 4 übernommen
- ist nicht mehr ausreichend getestet worden, da sie bei Ariane 4 problemlos lief
- Problem verursacht, weil Ariane 5 Schubstärker war
- Entwicklungszeit: 10 Jahre, Kosten ca. 6 Mrd. EURO
- Schaden: ca. 850 Mio. EUR

LE10 - Fehler, Debugger und Testen

20

Beispiele für Fehler

Mars Climate Orbiter

- Kosten
 - 327 Mio USD Orbiter
 - 193 Mio USD Raumschiff
 - 92 Mio USD Startkosten
 - 42 Mio USD Missionskosten
- ging 1999 wegen eines "Navigationsfehlers" verloren
- NASA erwartete Impulse in Newton je Sekunde (metrisches System)
- Software vom Hersteller Lockheed Martin lieferte Pound-force je Sekunde



LE10 - Fehler, Debugger und Testen

21

Beispiele für Fehler

Lufthansa-Flug 2904 (09/1993)

- Airbus A320 setzte in Warschau spät auf der Landebahn auf
- Aquaplaning ließ den Bordcomputer den Bodenkontakt nicht erkennen
- Bordcomputer verhinderte deshalb Bremsmanöver



Quelle: <http://www.airliners.net/photos/Lufthansa/Airbus-A320-211.022655412/>
Lizenz: GNU-FDL; für eine Dokumentation, Version 1.2 oder einer späteren Version
ACRO - Aircraft Crashes Record Office, Route de Cité Quai 25, CH - 1196 Glâne, Switzerland
per Mail am 12.08.2013
Mit freundlicher Genehmigung von Ronan HUBERT, Aviation Accidentologist

Beispiele für Fehler¹

Zwischen 1985 und 1986

- tötete Programmierfehler mehrere Patienten in einem Krankenhaus in den USA: Ein Bestrahlungsgerät verstrahlte die Patienten.

Zwischen August 2000 und Februar 2001

- erhielten 28 Krebspatienten in gesundem Gewebe eine zu hohe Bestrahlung. Mindestens 5 Patienten starben, 15 weitere trugen schwerste Schäden davon.
- Bediener der Therapie-Maschine hatte falsche Daten in das Behandlungsprogramm eingegeben, die von dem System nicht als falsch erkannt wurden

Beim Siemens Handy S65 war die Abschaltmelodie zu laut – mehrere Gehörschäden waren die Folge.

[1] Quelle: http://www.certitudo-gmbh.de/g_geschichte.html
LE10 - Fehler, Debugger und Testen

Fehlerquellen

Quellen von Fehlern sind vielgestaltet und niemals vollständig vorhersehbar

- Inkonsistente oder ungünstig gestaltete Benutzerschnittstellen
- Unerfüllte Erwartungen oder Widersprüche zwischen Spezifikation und Erwartung
- Schlechte Performance technischer Komponenten
- Fehlverhalten technischer Komponenten oder der Benutzer
- Abstürze und/oder Datenverluste (Überlauf)
- ...

Arten von Fehlern

Art	Beschreibung
Syntaxfehler	Der Quellcode enthält Befehle oder Ausdrücke, die nicht den Regeln der Programmiersprache (z.B. VBA) entsprechen (z.B. Funktion anstelle von Function). Diese Art von Fehlern wird bereits bei der Eingabe, spätestens vom Compiler vor Programmausführung gemeldet.
Fehlende Bibliotheken	Compiler findet nicht alle benötigten Bibliotheken, aus denen Funktionen aufgerufen werden. Deshalb kann das Programm nicht übersetzt oder ausgeführt werden.
Laufzeitfehler	Fehler der während der Ausführung des Programms auftritt (z.B. Überschreitung des Wertebereichs einer Variable).
Logischer Fehler	Wird das Programm technisch einwandfrei ausgeführt, produziert jedoch nicht die erwarteten Ergebnisse (z.B. Ergebnis einer Berechnung ist falsch, Artikelliste zeigt nicht alle Artikel) liegt ein logischer Fehler vor.

nach Böttcher, Frischalowski: Java 5 Programmierhandbuch. S. 675f.

Inhalt


Einordnung
Rückblick
Ausgangspunkt
– Beispiele für Softwarefehler
– Ursachen von Fehlern
Fehlervermeidung und -auffindung
– Unterstützung
– Programmierrichtlinien
– Codereview/Pair Programming
Debugger
– Zweck
– Einsatzmöglichkeiten
Testen
– Arten
– Testfall und Testdaten
– White-/Blockbox-Test
Abschluss und Ausblick



LE10 - Fehler, Debugger und Testen 26

Inhalt


Einordnung
Rückblick
Ausgangspunkt
– Beispiele für Softwarefehler
– Ursachen von Fehlern
Fehlervermeidung und -auffindung
– Unterstützung
– Programmierrichtlinien
– Codereview/Pair Programming
Debugger
– Zweck
– Einsatzmöglichkeiten
Testen
– Arten
– Testfall und Testdaten
– White-/Blockbox-Test
Abschluss und Ausblick



LE10 - Fehler, Debugger und Testen 27

Inhalt

Einordnung
Rückblick
Ausgangspunkt
– Beispiele für Softwarefehler
– Ursachen von Fehlern
Fehlervermeidung und -auffindung
– Unterstützung
– Programmierrichtlinien
– Codereview/Pair Programming
Debugger
– Zweck
– Einsatzmöglichkeiten
Testen
– Arten
– Testfall und Testdaten
– White-/Blockbox-Test
Abschluss und Ausblick




LE10 - Fehler, Debugger und Testen 28

Fehlervermeidung/-auffindung

Unterstützung der Entwicklungsumgebung nutzen

- in VBA mit MS Access
 - Variablendeklaration erzwingen, um Fehler durch Tippfehler in Variablenbezeichnern zu vermeiden, die bei automatischer Deklaration unentdeckt blieben
 - Fehlerhafte Anweisungen resultieren in Zeilen in roter Schrift oder im Fehlen der üblichen automatischen Befehlsergänzung
- in anderen Programmiersprachen
 - Warnungen des Compilers (auch auf höchster Warnstufe) beachten
 - Warnungen prüfen, ob sich Fehler dahinter verbergen
 - Populäres Beispiel aus C++ und Java: Versehentliche Zuweisung anstelle eines Vergleichs



LE10 - Fehler, Debugger und Testen 29

Demo 11.01

Warum liefert das Programm diese Ausgabe?

```
Option Compare Database
' Wo steckt der Fehler?
' Warum kommt als Ergebnis 3 und nicht 8?
Sub rechnen()
Dim intZahl1 As Integer
Dim intZahl2 As Integer
Dim intZahl3 As Integer
Let intZahl1 = 5
Let intZahl2 = 3
Let intZahl3 = intZahl1 + intZahl2
Debug.Print intZahl3
End Sub
```

Tippfehler:
Anstelle von 1
Buchstabe 1

Direktbereich
3

5 + 3 = 8
Warum hier 3?

LE10 - Fehler, Debugger und Testen 30

Demo 11.01

Deshalb: Immer Unterstützung der Entwicklungsumgebung und des Compilers nutzen!

- Option Explicit: Variablendeklaration erzwingen
- Option Strict: Implizite Typumwandlung (als gelegentliche weitere Fehlerquelle) verhindern (nicht in VBA, aber in VB.NET)

```
Option Compare Database
Option Explicit
' Wo steckt der Fehler?
' Warum kommt als Ergebnis
Sub rechnen()
Dim intZahl1 As Integer
Dim intZahl2 As Integer
Dim intZahl3 As Integer
Let intZahl1 = 5
Let intZahl2 = 3
Let intZahl3 = intZahl1 + intZahl2
Debug.Print intZahl3
End Sub
```

Fehler beim Kompilieren:
Variable nicht definiert

LE10 - Fehler, Debugger und Testen 31

Fehlervermeidung/-auffindung

Programmierrichtlinien, z.B.:

- Layout des Codes sollte Struktur des Codes wiedergeben
 - Einrückungen und Leerzeilen unterstützen die Visualisierung der Codestruktur
 - bekannte Strukturen erkennt man besser und schneller als neue
- pro Zeile nur eine Anweisung, dann ist immer klar, welche Anweisung zum Fehler geführt hat
 - VBA Interpreter hebt nicht fehlerhafte Zeilen gelb hervor und markiert nicht immer betroffene Anweisung
 - Viele Compiler geben bei Fehlern immer Zeilennummer an
- Keine zu langen Zeilen, die schlecht zu erfassen sind oder nicht vollständig auf den Bildschirm passen
- Klammern (auch wenn sie nicht zwingend sind) erhöhen oft die Lesbarkeit

LE10 - Fehler, Debugger und Testen

32

Fehlervermeidung/-auffindung

Programmierrichtlinien (Fortsetzung), z.B.:

- Gute Bezeichner unterstützen das Verständnis des Codes
 - Blöcke von Anweisungen und einzelne Anweisungen sind zu kommentieren
 - Einheiten von Daten sollten kommentiert werden, z.B.
- ```
Dim intRadius As Integer ' Radius in cm
Dim curPreis As Currency ' Gesamtpreis in EUR
```
- Keine unerwarteten Sprünge (z.B. mit GoTo), vor allem nicht in oder aus Schleifen und Verzweigungen
  - keine „Programmiertricks“ die von anderen nur schwer nachvollziehbar sind oder ohne aufwändige Kommentare unverständlich bleiben

LE10 - Fehler, Debugger und Testen

33



## Demo 11.02

### Was macht die Funktion a?

- Hinweis: Durch den Doppelpunkt können mehrere Anweisungen in VBA in einer Zeile geschrieben werden.

### Welches Ergebnis liefert die Prozedur test?

```
[Allgemein] test
Option Compare Database
Option Explicit

Function a(b As Byte) As Boolean
 Let a = False: Do While b > 0: Let b = b - 2: Loop: If b = 0 Then Let a = True
End Function

Sub test()
 Debug.Print a(1)
 Debug.Print a(2)
 Debug.Print a(3)
 Debug.Print a(4)
End Sub
```

Direkthereich  
Falsch  
Wahr  
Falsch  
Wahr

LE10 - Fehler, Debugger und Testen

34

## Demo 11.02

### Verständlichere Form der vorherigen Funktion

```
[Allgemein] test
Option Compare Database
Option Explicit

' Prüft, ob die als Parameter übergebene Zahl gerade ist.
' liefert true, wenn gerade. Andernfalls false.
Function istGerade(bytZahl As Byte) As Boolean

 Let istGerade = False ' Initialisierung des Rückgabewertes mit Falsch (nicht gerade)

 ' In einer Schleife solange 2 abziehen, wie die Zahl größer 0 ist
 Do While bytZahl > 0
 Let bytZahl = bytZahl - 2
 Loop

 ' Wenn der Wert gleich 0 ist, war die Zahl gerade
 If bytZahl = 0 Then
 Let istGerade = True ' Rückgabewert auf Wahr (gerade) setzen
 End If

End Function

' Testen der Funktion istGerade
Sub test()

 ' Aufruf der Funktion mit einigen Beispieldaten
 Debug.Print istGerade(1) ' muss Falsch liefern
 Debug.Print istGerade(2) ' muss Wahr liefern
 Debug.Print istGerade(3) ' muss Falsch liefern
 Debug.Print istGerade(4) ' muss Wahr liefern

End Sub
```

Direkthereich  
Falsch  
Wahr  
Falsch  
Wahr

LE10 - Fehler, Debugger und Testen

35

## Fehlervermeidung/-auffindung

### Codereviews

- Andere lesen den Quellcode meiner Programme
  - Zu wissen, dass andere den eigenen Code lesen, diszipliniert: man schreibt gewissenhafter Kommentare, ...
  - Ein zweiter Programmierer kann die Schwachpunkte der Software oft besser identifizieren
  - Programmierer können voneinander lernen
  - Man kann Leute gezielt in ein Thema einarbeiten und als Vertreter aufbauen
- Ergebnis von Codereviews sollte protokolliert werden
- Codereviews sollten nicht nur zum Auffinden von Fehlern, sondern immer zur Qualitätssicherung durchgeführt werden
- Werkzeuge (z.B. Code Analysis for Visual Studio bzw. für Java Checkstyle, FindBugs) können auch Codereviews unterstützen

LE10 - Fehler, Debugger und Testen

36

## Fehlervermeidung/-auffindung

### Pair Programming

- Methode Fehler zu reduzieren/zu vermeiden
- zwei Programmierer entwickeln gleichzeitig den Quellcode
- während einer schreibt, der prüft der andere
- vergleichbar eine Codereview, das parallel zur Entwicklung durchgeführt wird
- Eingeführt durch Ansatz des "eXtreme Programmings" (XP)

LE10 - Fehler, Debugger und Testen

37

**Inhalt**


**Einordnung**  
**Rückblick**  
**Ausgangspunkt**  
– Beispiele für Softwarefehler  
– Ursachen von Fehlern  
**Fehlervermeidung und -auffindung**  
– Unterstützung  
– Programmierrichtlinien  
– Codereview/Pair Programming  
**Debugger**  
– Zweck  
– Einsatzmöglichkeiten  
**Testen**  
– Arten  
– Testfall und Testdaten  
– White-/Blockbox-Test  
**Abschluss und Ausblick**



LE10 - Fehler, Debugger und Testen 38

**Inhalt**


**Einordnung**  
**Rückblick**  
**Ausgangspunkt**  
– Beispiele für Softwarefehler  
– Ursachen von Fehlern  
**Fehlervermeidung und -auffindung**  
– Unterstützung  
– Programmierrichtlinien  
– Codereview/Pair Programming  
**Debugger**  
– Zweck  
– Einsatzmöglichkeiten  
**Testen**  
– Arten  
– Testfall und Testdaten  
– White-/Blockbox-Test  
**Abschluss und Ausblick**



LE10 - Fehler, Debugger und Testen 39

**Inhalt**

**Einordnung**  
**Rückblick**  
**Ausgangspunkt**  
– Beispiele für Softwarefehler  
– Ursachen von Fehlern  
**Fehlervermeidung und -auffindung**  
– Unterstützung  
– Programmierrichtlinien  
– Codereview/Pair Programming  
**Debugger**  
– Zweck  
– Einsatzmöglichkeiten  
**Testen**  
– Arten  
– Testfall und Testdaten  
– White-/Blockbox-Test  
**Abschluss und Ausblick**




LE10 - Fehler, Debugger und Testen 40

**Debugger**

**Zweck**

- ermöglicht das Nachvollziehen eines Programms für einen bestimmten Ablauf
- Erklärung
  - der Ausführungsreihenfolge von Anweisungen
  - dient zum Erklären der Wertebelegung von Variablen
- Mittel zur Beseitigung von Fehlern (Bugs → Debugger)



LE10 - Fehler, Debugger und Testen 41

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

42

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

43

```

[Allgemein] | | Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtModiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtModiMi
 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then
 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("Mo", 2)
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

44

```

[Allgemein] | | Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtModiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtModiMi
 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then
 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("Mo", 2)
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

45

```

[Allgemein] | | Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtModiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtModiMi
 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then
 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("Mo", 2)
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

[Allgemein] Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtMoDiMi
 Else
 Let curPreis = pcurPreis * sglRbtSaSo
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("Mo", 2)
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

| Ausdruck  | Wert | Typ      | Kontext         |
|-----------|------|----------|-----------------|
| pcurPreis | 2    | Currency | Debugger.Rab... |

LE10 - Fehler, Debugger und Testen

46

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

[Allgemein] Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtMoDiMi
 Else
 Let curPreis = pcurPreis * sglRbtSaSo
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("Mo", 2)
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

| Ausdruck  | Wert | Typ      | Kontext         |
|-----------|------|----------|-----------------|
| pcurPreis | 2.2  | Currency | Debugger.Rab... |

LE10 - Fehler, Debugger und Testen

47

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

[Allgemein] Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtMoDiMi
 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then
 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

| Ausdruck  | Wert | Typ      | Kontext         |
|-----------|------|----------|-----------------|
| curPreis  | 0    | Currency | Debugger.Rab... |
| pcurPreis | 3    | Currency | Debugger.Rab... |
| pstrTag   | "r"  | String   | Debugger.Rab... |

LE10 - Fehler, Debugger und Testen

48

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

[Allgemein] Rabatt
Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Single = 0.9
 Const sglRbtSaSo As Single = 0.5
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then
 Let curPreis = pcurPreis * sglRbtMoDiMi
 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then
 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub teste()
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

| Ausdruck  | Wert | Typ      | Kontext         |
|-----------|------|----------|-----------------|
| curPreis  | 0    | Currency | Debugger.Rab... |
| pcurPreis | 3    | Currency | Debugger.Rab... |
| pstrTag   | "r"  | String   | Debugger.Rab... |

LE10 - Fehler, Debugger und Testen

49

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Currency = 0.1
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then

 Let curPreis = pcurPreis * sglRbtMoDiMi

 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then

 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub t
 Debug.Print
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

LE10 - Fehler, Debugger und Testen

50

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Currency = 0.1
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then

 Let curPreis = pcurPreis * sglRbtMoDiMi

 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then

 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub t
 Debug.Print
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

LE10 - Fehler, Debugger und Testen

51

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

```

Option Compare Database
Option Explicit

Private Function Rabatt(pstrTag As String, _
 pcurPreis As Currency) As Currency
 Const sglRbtMoDiMi As Currency = 0.1
 Dim curPreis As Currency

 If pstrTag = "Mo" Or pstrTag = "Di" _
 Or pstrTag = "Mi" Then

 Let curPreis = pcurPreis * sglRbtMoDiMi

 ElseIf pstrTag = "So" Or _
 pstrTag = "Sa" Then

 Let curPreis = pcurPreis * sglRbtSaSo
 Else
 Let curPreis = pcurPreis
 End If

 Let Rabatt = curPreis
End Function

Private Sub t
 Debug.Print
 Debug.Print Rabatt("So", 1)
 Debug.Print Rabatt("Fr", 3)
End Sub

```

LE10 - Fehler, Debugger und Testen

52

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

53

## Demo 11.03: Debugger

### Ziel

- Kennenlernen des Debuggers

### Aufgabe

- Symbolleiste des Debuggers einrichten
- Haltepunkt(e) festlegen und zu überwachende Variablen festlegen
- Programm (nächste Folie) starten und schrittweise ausführen



LE10 - Fehler, Debugger und Testen

54

## Demo 11.03: Debugger

```
Sub demoll01()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Zahl eingeben: "))
 Debug.Print "Ist Primzahl: " & isPrim(intZahl)
End Sub

Function isPrim(pintZahl As Integer) As Boolean
 Dim i As Integer
 Dim bolIsPrim As Boolean

 Let i = 2
 Let bolIsPrim = True

 Do While (bolIsPrim And i <= pintZahl / 2)
 If (pintZahl Mod i = 0) Then
 Let bolIsPrim = False
 End If
 Let i = i + 1
 Loop

 Let isPrim = bolIsPrim
End Function
```

LE10 - Fehler, Debugger und Testen

55

## Debugger

### Haltepunkte

- Markieren Zeilen (Anweisung) in der der Programmablauf angehalten werden soll

### Überwachung

- Betrachtung von Variablenwerten
- Änderung von Variablenwerten

### schrittweise Ausführung

- Ausführen einzelner Anweisungen
- Ausführen einer ganzen Prozedur
- ...

LE10 - Fehler, Debugger und Testen

56

## Weitere Möglichkeiten

### Informationen während Programmausführung ausgeben

- im Direktbereich
  - verlangsamt die Programmausführung
    - von einigen Entwicklungsumgebungen werden Ausgaben im Direktbereich bei Erstellung des endgültigen Programms (Release) entfernt
  - Quellcode wird überladen mit Befehlen, die nicht zur Lösung beitragen und dadurch unübersichtlich
  - nach Ende der Anwendung nicht mehr nachvollziehbar (weil Direktbereich geschlossen wurde)
- in Protokolldateien (Logging)
  - verschiedene Arten von Ausgaben (z.B. Fehler, Warnungen, Informationen) möglich
  - verlangsamt die Programmausführung
  - Quellcode wird überladen mit Befehlen, die nicht zur Lösung beitragen und dadurch unübersichtlich
  - nach Programmausführung nachvollziehbar, z.B. durch Administrator

LE10 - Fehler, Debugger und Testen

57

**Inhalt**

**Einordnung**  
**Rückblick**  
**Ausgangspunkt**

- Beispiele für Softwarefehler
- Ursachen von Fehlern

**Fehlervermeidung und -auffindung**

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

**Debugger**

- Zweck
- Einsatzmöglichkeiten

**Testen**

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

**Abschluss und Ausblick**



LE10 - Fehler, Debugger und Testen 58

**Inhalt**

**Einordnung**  
**Rückblick**  
**Ausgangspunkt**

- Beispiele für Softwarefehler
- Ursachen von Fehlern

**Fehlervermeidung und -auffindung**

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming


**Debugger**

- Zweck
- Einsatzmöglichkeiten

**Testen**

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

**Abschluss und Ausblick**



LE10 - Fehler, Debugger und Testen 59

**Inhalt**

**Einordnung**  
**Rückblick**  
**Ausgangspunkt**

- Beispiele für Softwarefehler
- Ursachen von Fehlern

**Fehlervermeidung und -auffindung**

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming


**Debugger**

- Zweck
- Einsatzmöglichkeiten

**Testen**

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

**Abschluss und Ausblick**



LE10 - Fehler, Debugger und Testen 60


**Testen**

**Um Fehler zu minimieren muss man testen!**

**Definition Test:**

- "Unter dem Test von Software verstehen wir die stichprobenartige Ausführung eines Testobjekts, die zu dessen Überprüfung dienen soll.
- Dazu müssen die Randbedingungen für die Ausführung des Tests [vorher] festgelegt [und jederzeit rekonstruierbar] sein.
- Über einen Vergleich zwischen Soll- und Ist-Verhalten wird bestimmt [und dokumentiert], ob das Testobjekt die geforderten Eigenschaften erfüllt."<sup>1</sup>

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1



LE10 - Fehler, Debugger und Testen 61

## Softwaretest



### Testen: Anhand von Stichproben Korrektheit der Software prüfen, aber nicht beweisen

- Dynamische Tests: Mittels Programmausführung wird die Prüfung durchgeführt
  - Strukturtests (White Box): Die innere Struktur der Software ist bekannt und wird bei Testfallerstellung und -durchführung berücksichtigt
  - Funktionstest (Black Box): Es wird das nach außen "sichtbare" Systemverhalten geprüft, unabhängig von der internen Struktur der Software
  - ...
- Statische Tests: Mittels Inspektion, Review, Statische Analyse wird der Test durchgeführt

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

62

## Demonstratives und destruktives Testen



### Es gibt zwei Arten des Testens:

- Demonstratives Testen: systematische Ausführung eines Programms zur Erhöhung der Qualität
- Destruktives Testen: Prozess mit der Absicht, Fehler zu finden.

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

63

## Demonstratives und destruktives Testen



### Es gibt zwei Arten des Testens:

- Demonstratives Testen: systematische Ausführung eines Programms zur Erhöhung der Qualität
  - Viele führen nur demonstratives Testen durch: „Sieh doch, es geht!“
  - häufig aus der eigenen Entwicklersicht durchgeführt
  - erwartete Eingabewerte sollen erwartete Ausgaben produzieren
- Destruktives Testen: Prozess mit der Absicht, Fehler zu finden.

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

64

## Demonstratives und destruktives Testen



### Es gibt zwei Arten des Testens:

- Demonstratives Testen: systematische Ausführung eines Programms zur Erhöhung der Qualität
- Destruktives Testen: Prozess mit der Absicht, Fehler zu finden.
  - Wichtig ist aber destruktives Testen (z.B. Benutzerverhalten mit dem man nicht rechnet) durchzuführen.
  - Dies wird am besten von anderen Personen durchgeführt (große Firmen haben eigene Testabteilungen)

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

65



## Demonstratives und destruktives Testen

### Es gibt zwei Arten des Testens:

- Demonstratives Testen: systematische Ausführung eines Programms zur Erhöhung der Qualität
- Destruktives Testen: Prozess mit der Absicht, Fehler zu finden.

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

66

## Testfall und Testdaten

### Testfall besteht aus Testablauf und den dort verwendeten Daten

- Testablauf, z.B. für Testen eines Anmeldungsvorgang (Login)
  - Ablauf 1
    - Eingabe des korrekten Benutzernamens
    - Eingabe des korrekten Kennwortes
    - Betätigen der Login-Schaltfläche
    - Anwendung startet
  - Ablauf 2
    - Eingabe des korrekten Benutzernamens
    - Eingabe des falschen Kennwortes
    - Betätigen der Login-Schaltfläche
    - Fehlermeldung "Ungültige Anmeldung"
  - Ablauf 3
    - Eingabe des falschen Benutzernamens
    - Eingabe des korrekten Kennwortes
    - Betätigen der Login-Schaltfläche
    - Fehlermeldung "Ungültige Anmeldung"
- Testdaten
  - Eingabe eines korrekten Benutzernamen-Kennwortpaars (User1, Passwort1)
  - Eingabe eines falschen Benutzernamen-Kennwortpaars (User2, 123)

– Dokumentation erfolgt in Testfallspezifikation

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

67

## White-Box- und Black-Box-Tests

### Es gibt verschiedene Arten von Tests

- White-Box-Test
  - Testen unter Berücksichtigung der inneren Struktur des Testobjektes
- Black-Box-Test
  - Testen ohne Kenntnis der inneren Struktur; Grundlage ist die Spezifikation

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

68

## Beispiel Testfälle finden (White/Black-Box)

Beispiel für Testfallfindung: Primzahltest (vereinfacht! Gilt so nur für Eingaben  $\geq 2$ )

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

69

### Beispiel Primzahltest

#### Definition der Primzahl (vgl. Wikipedia)

- Eine Primzahl ist eine natürliche Zahl,
- die größer als 1 ist und
- die nur durch sich selbst und durch 1 ganzzahlig teilbar ist.

#### Welche Primzahlen fallen Ihnen spontan ein?

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, ...

#### Welche Nicht-Primzahlen fallen Ihnen spontan ein?

1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 35, 36, 38, 39, ...

### Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub

Private Function isPrim(pintZahl As Integer) As Boolean
 Dim i As Integer
 Dim bolIsPrim As Boolean

 Let i = 2
 Let bolIsPrim = True

 Do While (bolIsPrim And i <= pintZahl / 2)
 If (pintZahl Mod i = 0) Then
 Let bolIsPrim = False
 End If
 Let i = i + 1
 Loop

 Let isPrim = bolIsPrim
End Function
```

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

### Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub
```

```
Private Function isPrim(pintZahl As Integer) As Boolean
```

#### Black-Box-Test

Welche Tests sind mindestens erforderlich?

```
End Function
```

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

### Eingabedaten Black-Box-Test

#### Mindestens 2 Eingabedaten:

- eine Primzahl, z.B. 7
- eine Nicht-Primzahl, z.B. 300

#### Oft ist es noch günstig Extremalwerte zu betrachten, hier z.B.

- die kleinste Primzahl: 2
- die Sonderrolle: 1 (per Definition keine Primzahl)

#### Was ist das erwartete Ergebnis?

- 7 liefert True
- 300 liefert False
- 2 liefert True
- 1 liefert False

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

## Eingabedaten Black-Box-Test

Da man i.a. nicht alle Eingabedaten untersuchen kann beschränkt man sich oft auf Äquivalenzklassen;

- Hier ergeben sich natürlicherweise mindestens 2 Äquivalenzklassen:
  - Die Primzahlen und
  - die Nicht-Primzahlen
- Die Nicht-Primzahlen könnte man evtl. noch in zwei Klassen zerlegen:
  - Gerade Nicht-Primzahlen
  - Ungerade Nicht-Primzahlen

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

74

## Eingabedaten Black-Box-Test

Innerhalb jeder Äquivalenzklasse sollte man dann ein paar typische Vertreter wählen, sowie Extremalwerte

- Somit sind für den Primzahltest z.B. folgende Eingabedaten sinnvoll:
  - Primzahlen: 2 (Extremalwert), 17, 8999 (große Primzahl, eine größte gibt es nicht)
  - Nicht-Primzahlen:
    - gerade: 4 (Extremalwert), 100, 134568 (große)
    - ungerade: 9 (Extremalwert), 153, 168651

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

75

## Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub
```

```
Private Function isPrim(pintZahl As Integer) As Boolean
 Dim i As Integer
 Dim bolIsPrim As Boolean

 Let i = 2
 Let bolIsPrim = True

 Do While (bolIsPrim And i <= pintZahl / 2)
 If (pintZahl Mod i = 0) Then
 Let bolIsPrim = False
 End If
 Let i = i + 1
 Loop

 Let isPrim = bolIsPrim
End Function
```

### White-Box-Test

Welche Tests sind mindestens erforderlich?

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

77

## Eingabedaten White-Box-Test

Welche Tests sind mindestens erforderlich?

Jeder mögliche „Pfad“ durch den Programmcode sollte mindestens einmal durchlaufen werden!

Bei Schleifen:

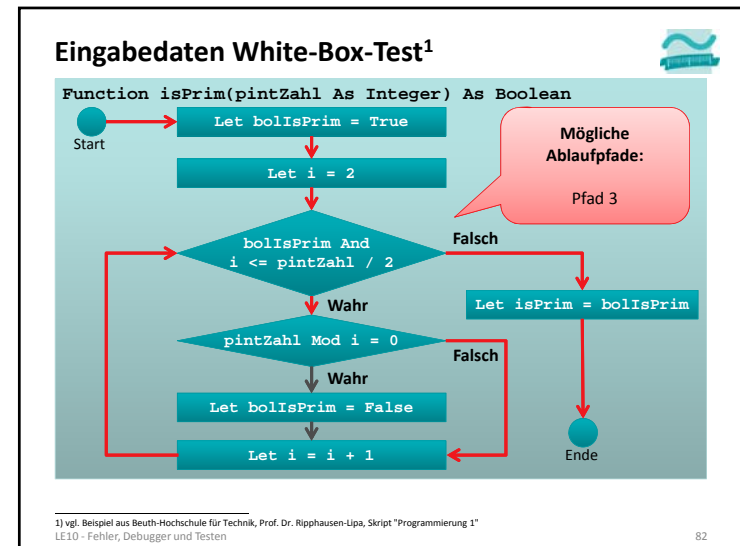
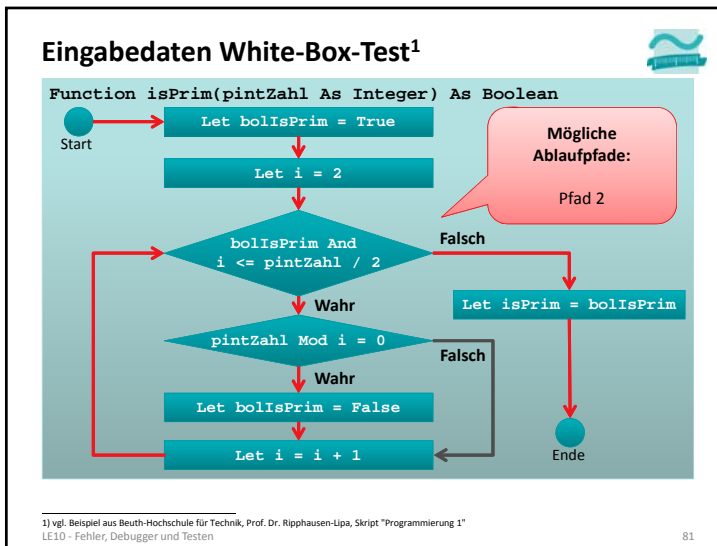
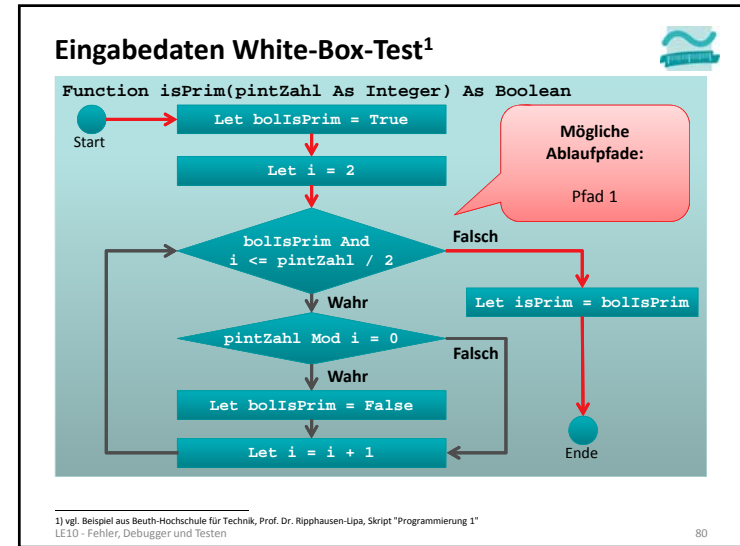
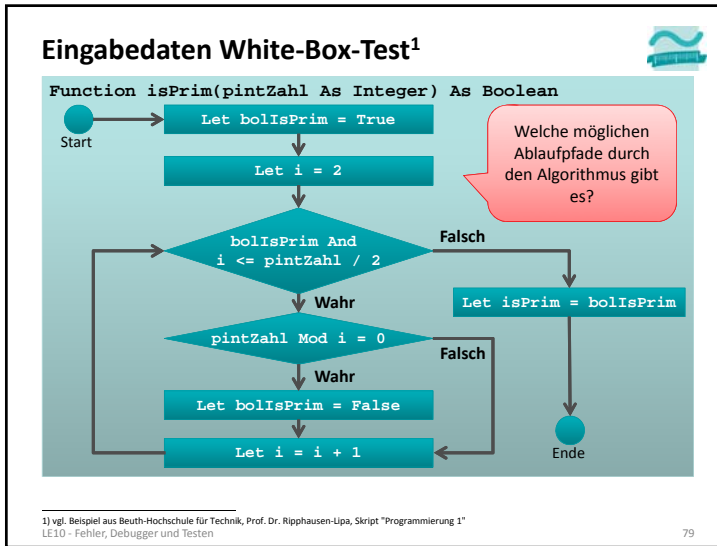
- Kein Durchlauf
- Ein Durchlauf
- Zwei Durchläufe
- Typische Anzahl Durchläufe
- Maximale Anzahl Durchläufe

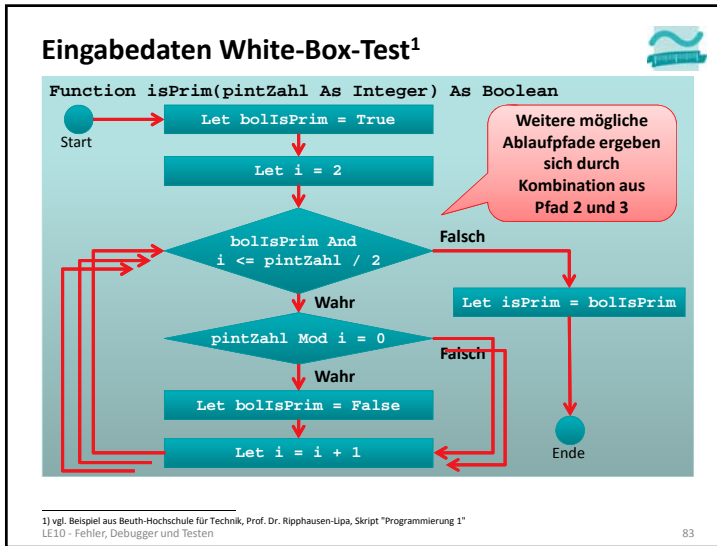
Bei Bedingungen: jeder (Teil-)Ausdruck sollte wenigstens jeden der möglichen Werte einmal annehmen!

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE10 - Fehler, Debugger und Testen

78





### Eingabedaten White-Box-Test<sup>1</sup>

**Testdaten:**

- Pfad 1: n = 4
- Pfad 2: n = 5 (einmal), n = 101 (mehrfach)
- Pfad 3: n = 2; n = 1 ← **Zweiter Test zeigt falsches Ergebnis auf!**
- Kombination Pfad 2, Pfad 1
  - Je einmal: z.B. 9
  - Pfad 2 dreimal, Pfad 1 einmal: 25
  - Pfad 2 mehrfach, Pfad 1 einmal: 121

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1  
LE10 - Fehler, Debugger und Testen 84

### Bemerkungen zum systematisches Testen

Es ist sinnvoll Testfälle mit gleichem Testablauf zu „automatisieren“, da bei jeder Änderung / Fehlerkorrektur am Besten alle Testfälle wieder durchlaufen werden (Hinweis: es gibt Tools, die dies unterstützen wie z.B. NUnit, JUnit)

Es gibt sogar testgetriebenes Design / Vorgehen zur Entwicklung vor Software; bei dieser Methode werden zuerst die Testfälle entwickelt, bevor die eigentliche Software entwickelt wird

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1  
LE10 - Fehler, Debugger und Testen 85

### Module und Testen

**Umsetzung in VBA**

- Modul und Testmodul bilden eine Einheit für den Test
- zu jedem Modul wird ein Testmodul erstellt (z.B. zu Rechnung das Modul RechnungTest)
- prozedurNameTest zur Test der Prozedur prozedurName
- testeAlles() -Prozedur zur Ausführung aller Test im Test-Modul (optional)

LE10 - Fehler, Debugger und Testen 86

## Module und Testen

### Assertion (dt. Zusicherung)

- Prüfen einer vorher definierten Erwartung, die das Endergebnis erfüllen muss
- weicht das tatsächliche Ergebnis vom erwarteten Ergebnis ab, wird die Programmausführung unterbrochen
- hilft logische Fehler zu finden

### VBA

```
' Unterbricht die Programm, wenn boolescher Ausdruck falsch
Debug.Assert <BoolescherAusdruck>
```

Innerhalb der Test-Prozeduren werden zu testende Funktionen innerhalb einer Debug.Assert-Anweisung aufgerufen.

LE10 - Fehler, Debugger und Testen

87

## Module und Testen

### Beispiel: Modul mdlPrimzahlen

```
Option Compare Database
Option Explicit

Public Function isPrim(pintZahl As Integer) As Boolean
 ' ...

End Sub
```

### Beispiel: Modul mdlPrimzahlenTest

```
Option Compare Database
Option Explicit

Private Sub isPrimTest()
 Debug.Assert mdlPrimzahlen.isPrim(0) = True
 Debug.Assert mdlPrimzahlen.isPrim(1) = True
 Debug.Assert mdlPrimzahlen.isPrim(2) = False
 Debug.Assert mdlPrimzahlen.isPrim(3) = True

End Sub
```

LE10 - Fehler, Debugger und Testen

88

## Inhalt

### Einordnung

### Rückblick

### Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern

### Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

### Debugger

- Zweck
- Einsatzmöglichkeiten

### Testen

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test

### Abschluss und Ausblick



LE10 - Fehler, Debugger und Testen

89

## Inhalt

### Einordnung

### Rückblick

### Ausgangspunkt

- Beispiele für Softwarefehler
- Ursachen von Fehlern

### Fehlervermeidung und -auffindung

- Unterstützung
- Programmierrichtlinien
- Codereview/Pair Programming

### Debugger

- Zweck
- Einsatzmöglichkeiten

### Testen

- Arten
- Testfall und Testdaten
- White-/Blockbox-Test


### Abschluss und Ausblick

LE10 - Fehler, Debugger und Testen

90

### Inhalt

Einordnung  
Rückblick  
Ausgangspunkt  
– Beispiele für Softwarefehler  
– Ursachen von Fehlern  
Fehlervermeidung und -auffindung  
– Unterstützung  
– Programmierrichtlinien  
– Codereview/Pair Programming  
Debugger  
– Zweck  
– Einsatzmöglichkeiten  
Testen  
– Arten  
– Testfall und Testdaten  
– White-/Blockbox-Test  
Abschluss und Ausblick





LE10 - Fehler, Debugger und Testen 91

### Zusammenfassung

#### Fehlervermeidung/-auffindung

- Funktionen der Entwicklungsumgebung, des Compilers und weiterer Werkzeuge nutzen
- Programmierrichtlinien einführen
- Codereviews durchführen, ihre Ergebnisse dokumentieren
- Weitere Werkzeuge zur automatischen Analyse nutzen (z.B. Code Analysis, FindBugs, Checkstyle)





LE10 - Fehler, Debugger und Testen 92

### Zusammenfassung

#### Debugger

- ermöglicht durch Haltepunkte, Variablenüberwachung und schrittweise Ausführung
- Nachvollziehen des tatsächlichen Programmablaufs und der Wertebelegung von Variablen
- dient der Analyse von identifizierten Fehlerzuständen





LE10 - Fehler, Debugger und Testen 93

### Zusammenfassung

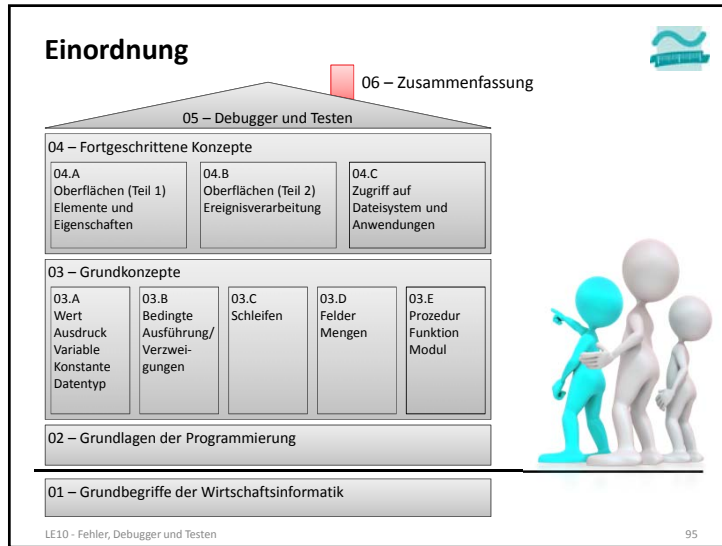
#### Testen

- Demonstratives und Destruktives Testen im Vier-Augen-Prinzip
- Wahl geeigneter Testfall und Testdaten zur Abdeckung aller Eingabe-/Ausgabekombinationen und relevanten Ausführungspfade im White-/Blockbox-Test
- Testen von Modulen durch Erstellung von Testprozeduren und Verwendung von Debug.Assert

```
' Unterbricht die Programm, wenn
' boolscher Ausdruck falsch
Debug.Assert <BoolscherAusdruck>
```



LE10 - Fehler, Debugger und Testen 94



**Wirtschaftsinformatik 1**  
**LE 10 – Fehler, Debugger und Testen**

Prof. Dr. Thomas Off

<http://www.ThomasOff.de/lehre/beuth/wi1>