Seminarvortrag 10:

Testen von grafischen Benutzeroberflächen

2004 / 06 / 28

Clemens Sommer, Gerald Peter

Übersicht

- Motivation
  - GUI – Allgemein
  - Fehlerquellen und deren Auswirkungen
- GUI – Testwerkzeuge
  - JUnit
  - JFC-Unit
  - Abbot

GUI - Allgemein

Grafische Benutzer (-Maschine) Schnittstelle

Problem - Stellung 1

Was bedeutet benutzer-freundliches Design;
Beachtung psychologischer Regeln
Testen der Usability durch Psychologen

Problem - Stellung 2

Dynamisches Verhalten:
Testen durch mehrere oder wechselnde Benutzer mit unterschiedlicher Erfahrung und verschiedenartigen Interaktionen

Problem - Stellung 3

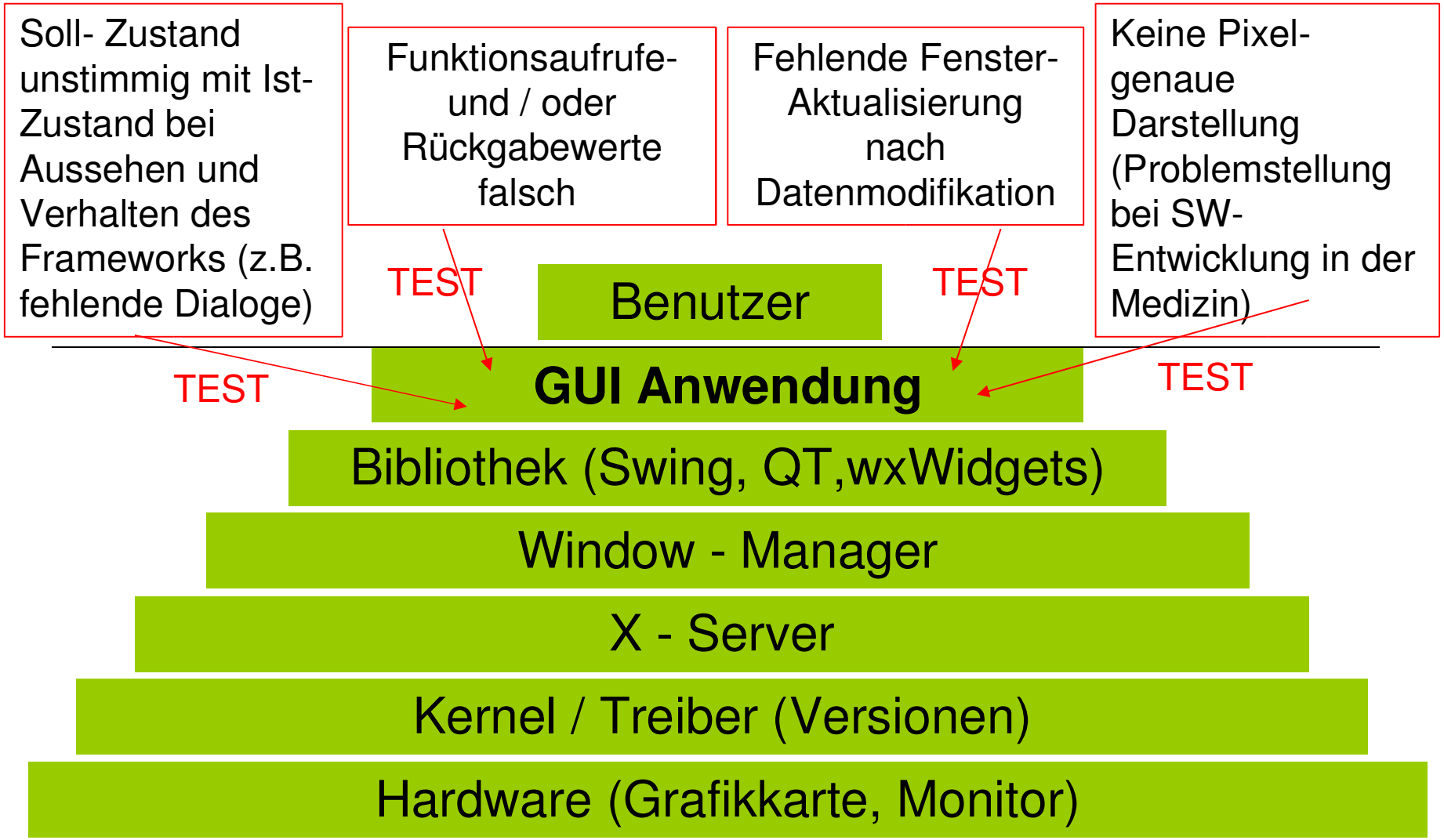
Verschiedenartige Plattformen;
Keine einheitlichen GUI-Normen;
Testen auf unterschiedlicher HW;
bzw. Testen auf unterschiedlichen Bibliotheken

Problem - Stellung 4

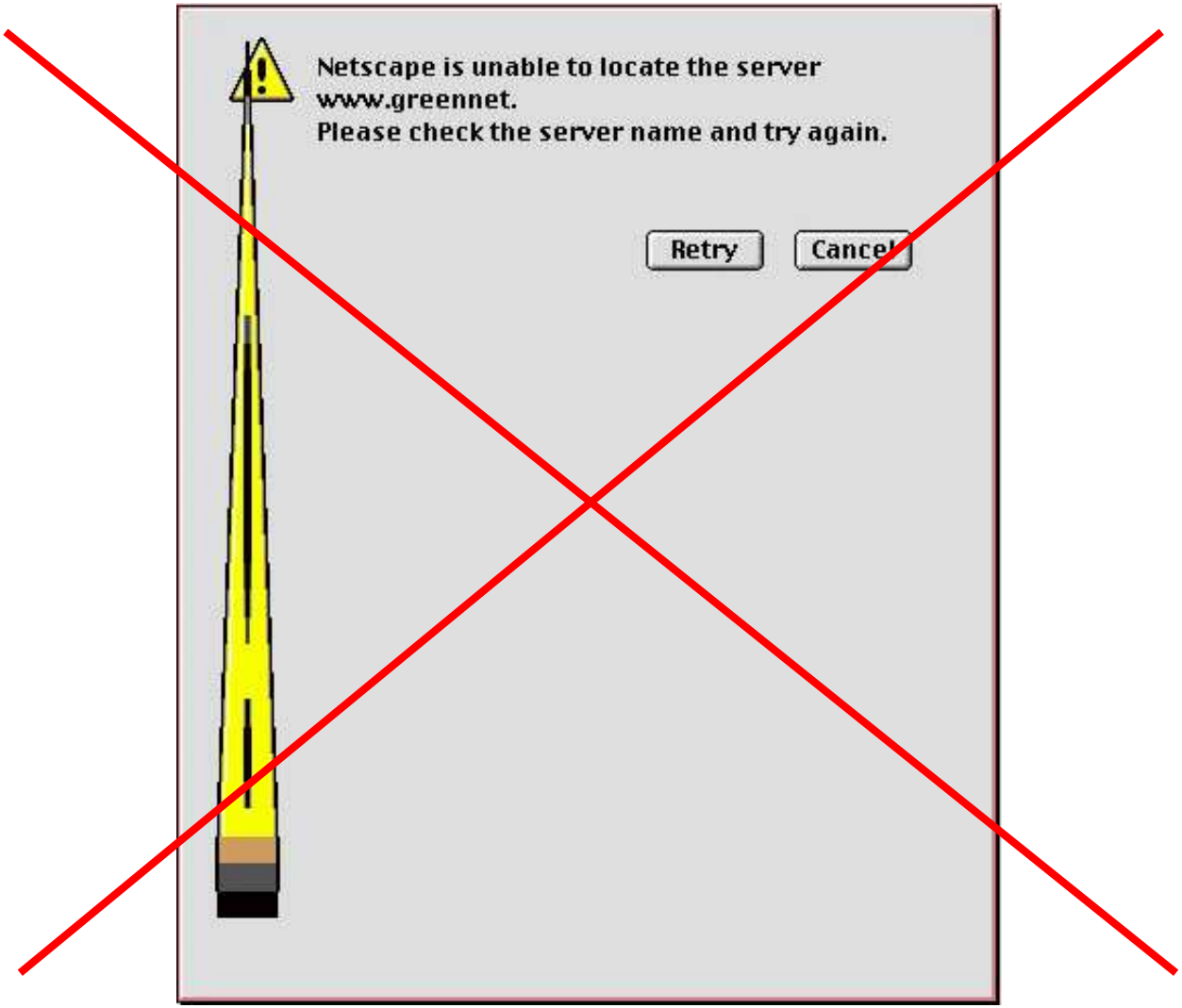
Unterschiedliche Schnittstellenarten, abhängig von der Problemstellung

TESTEN MIT UNTERSCHIED-LICHEN TEST-WERKZEUGEN

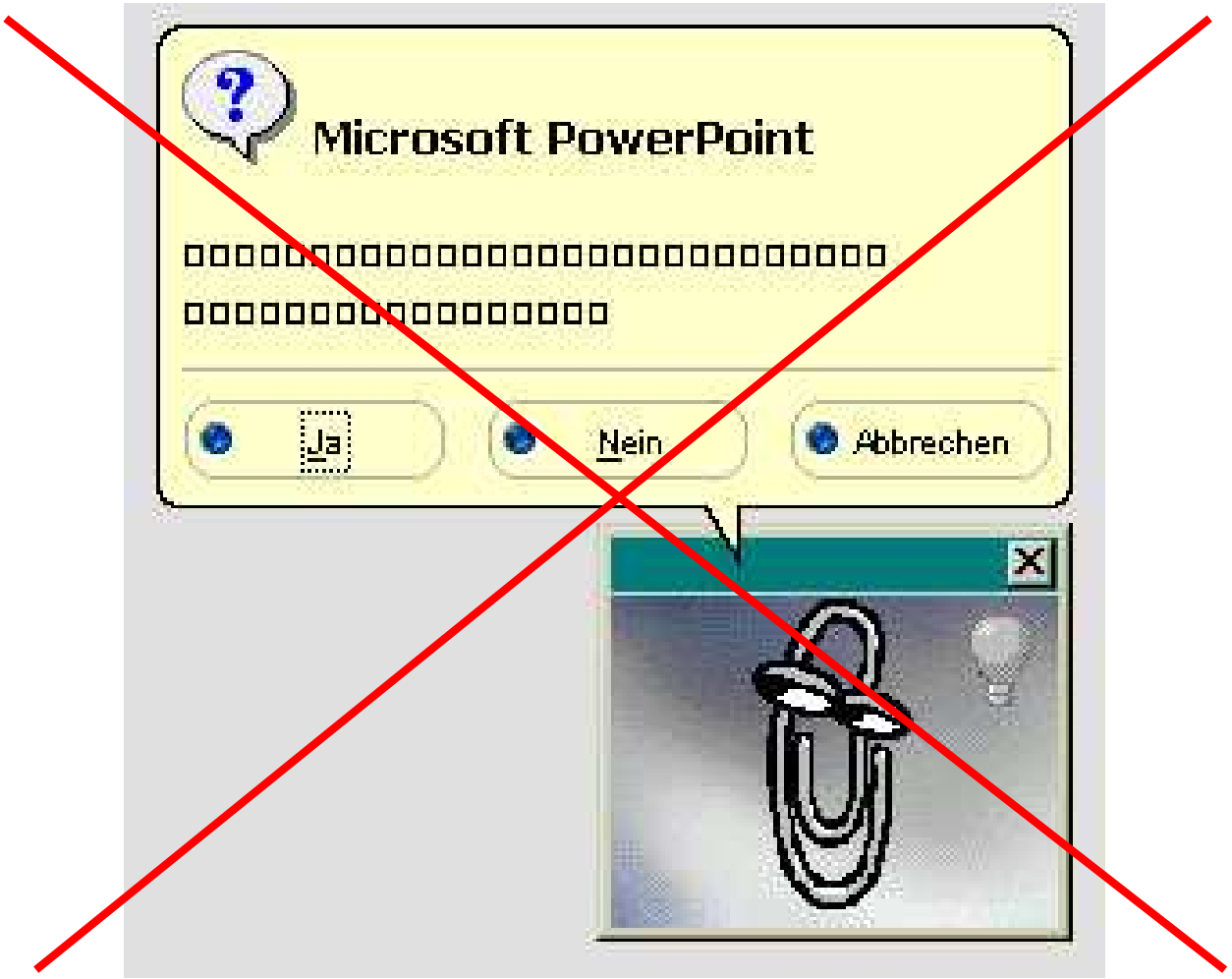
Fehlerquellen (Fall-Beispiel.:Linux)



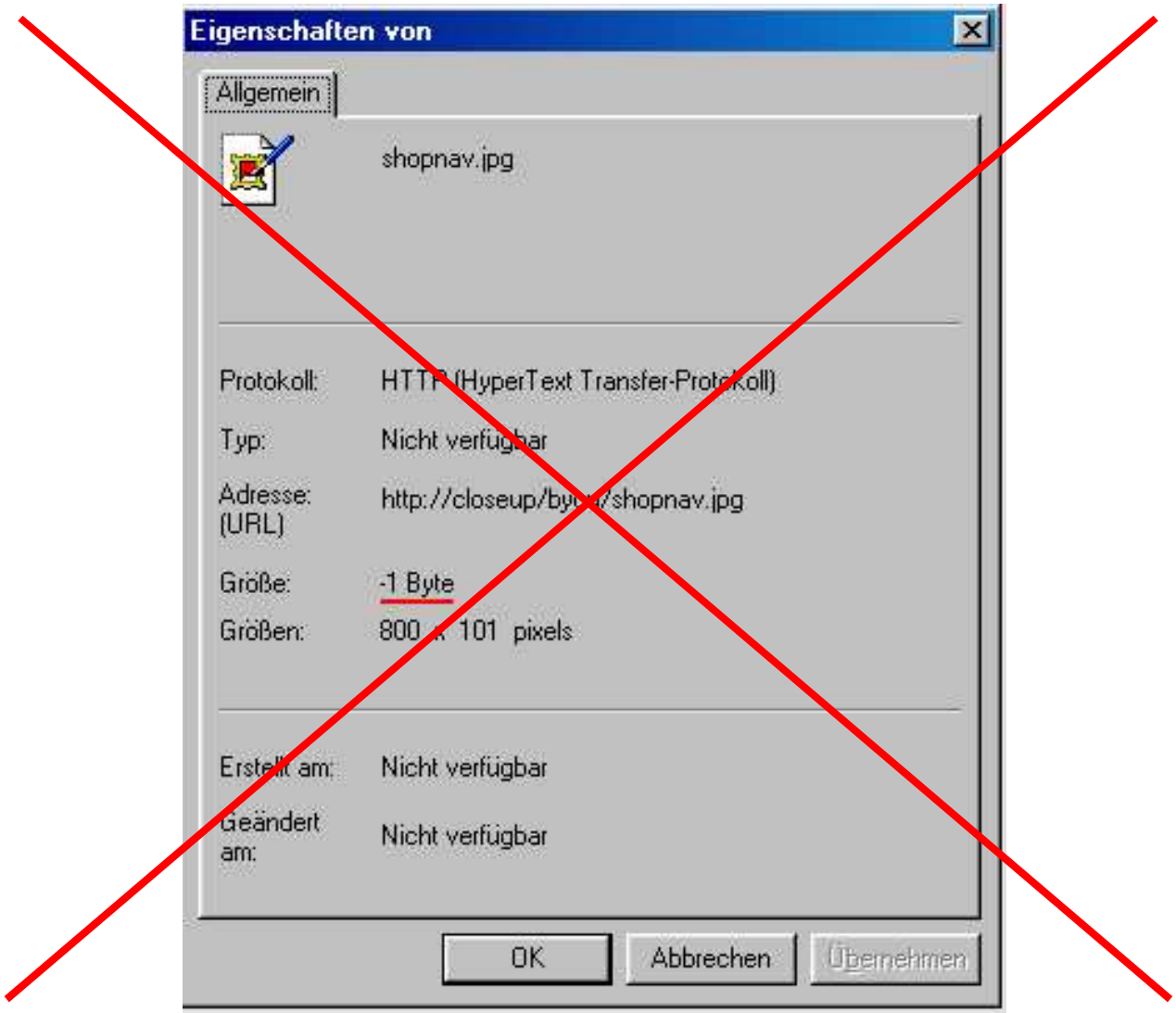
Fehlerauswirkungen



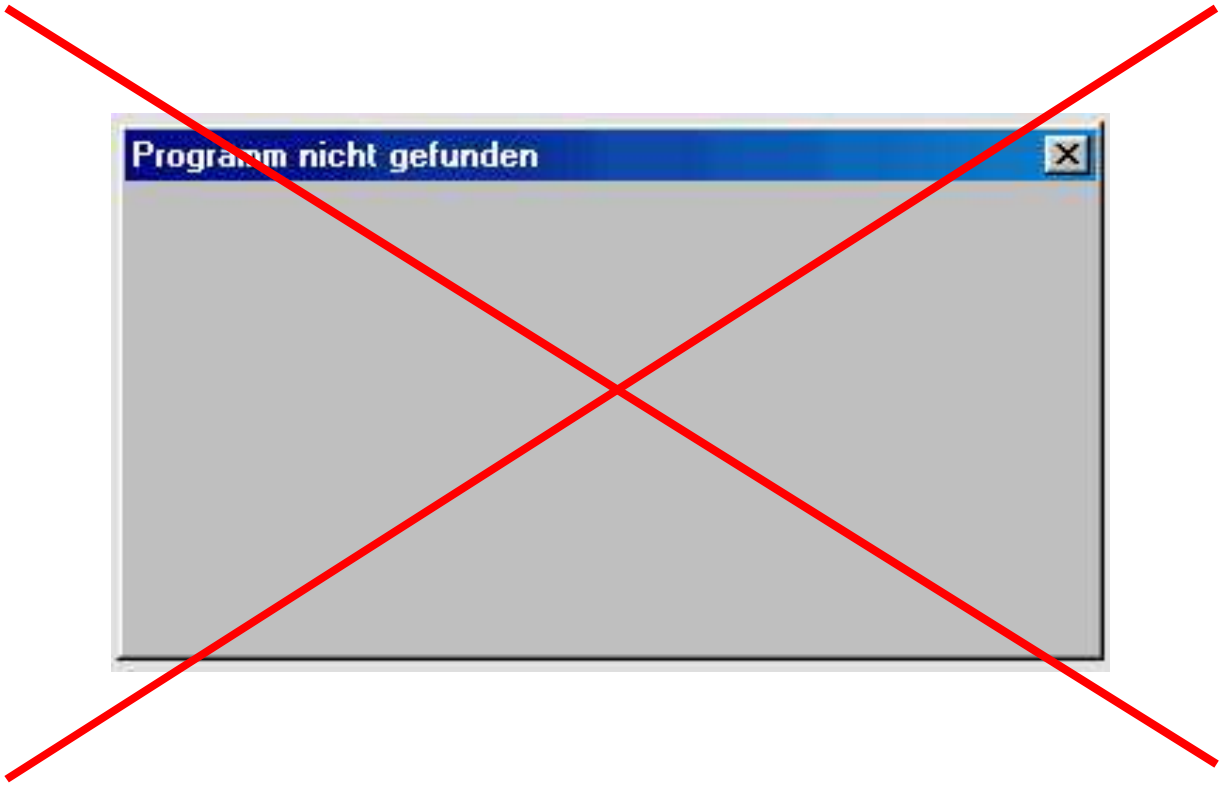
Fehlerauswirkungen



Fehlerauswirkungen



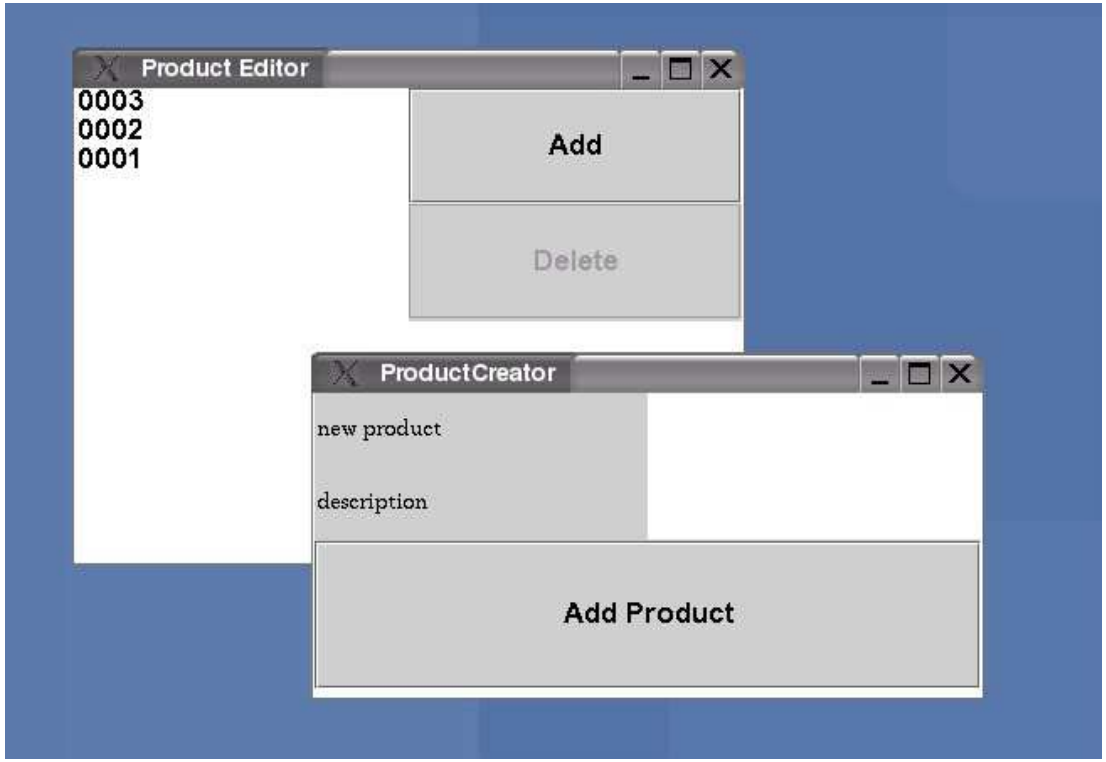
Fehlerauswirkungen



Test - Werkzeuge

- JUnit
- JFC-Unit
- Abbot
- Sonstige Werkzeuge
  - Pounder, Jacareto

Testen mit JUnit



am Beispiel der grafischen Benutzeroberfläche des Product Editors

Testen mit JUnit

Vorbereitung des Tests

- ```
  protected void setUp() {
       editor = new CatalogEditor(catalog);
       editor.show();
  }
  ```

- ```
  protected void tearDown() {
       editor.dispose();
  }
  ```

Aufruf der grafischen Junit Oberfläche

```
> java junit.swingui.TestRunner
```

Testen mit JUnit

Ein erster Test

```
public void testDeleteButton() {

     assertTrue(editor.myDeleteButton.isShowing());
     assertEquals("Delete", editor.myDeleteButton.getText());
     assertTrue(!editor.myDeleteButton.isEnabled());

     editor.myList.setSelectedIndex(0);
     assertTrue(editor.myDeleteButton.isEnabled());

     editor.myList.clearSelection();
     assertTrue(!editor.myDeleteButton.isEnabled());
}
```

Testen mit JUnit

Löschen eines Eintrags aus der Produktliste

```
public void testDeleteProduct() {

     editor.myList.setSelectedIndex(2);

     ...

     editor.myDeleteButton.doClick();
     assertEquals(2, editor.getListSize());
     assertEquals(2, catalog.getProducts().size());
     assertTrue(!catalog.getProducts().contains(del));
}
```

Testen mit JUnit

Einfügen eines Eintrags in die Produktliste

```
public void testProductList() {

     assertEquals(3, editor.getListSize());

     Product product4 = new Product("0004");
     product4.setDescription("Mousepad");

     catalog.addProduct(product4);
     editor.myList.setListData(catalog.getPIDs());

     assertEquals(4, editor.getListSize());
}
```

Testen mit JUnit

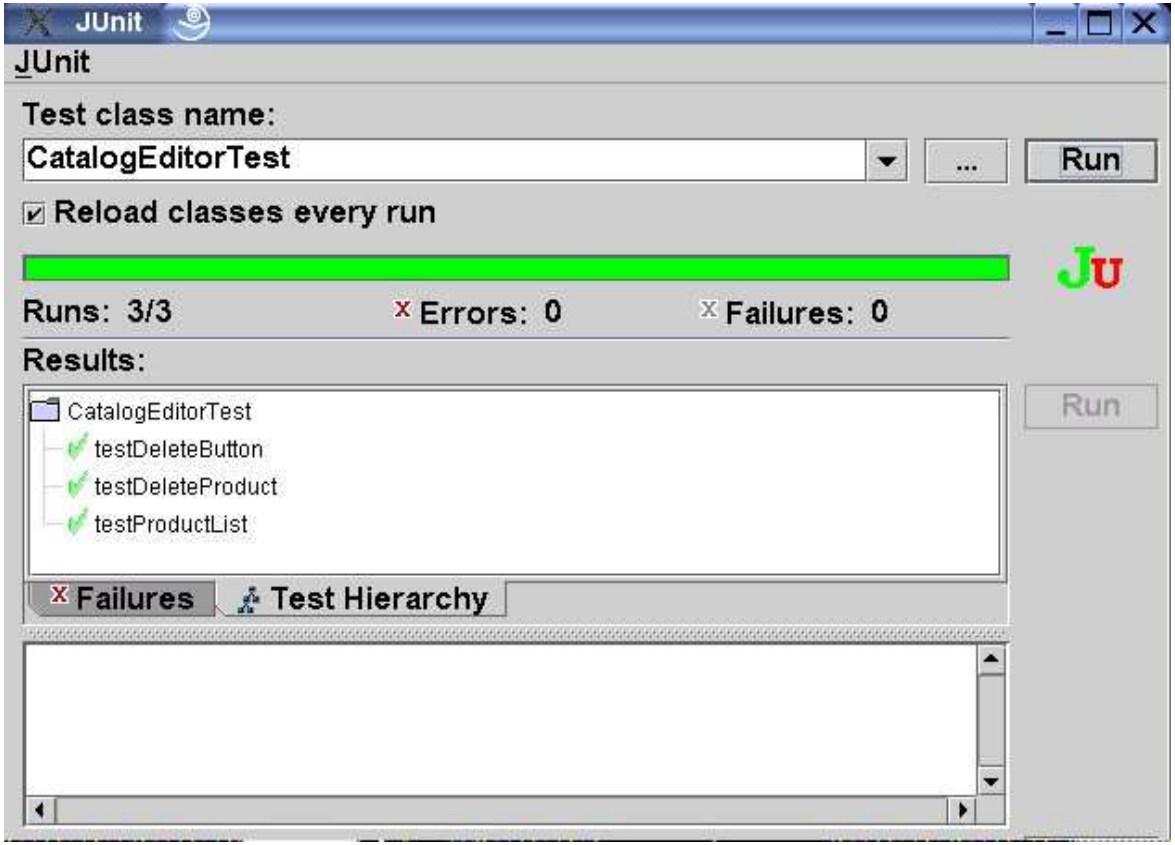
Zusammenfassung

- Das Hinzufügen eines Produkts ist ohne die Benutzung der Swing Oberfläche für den Add-Button möglich.
  ```
  catalog.addProduct(product4);
  ```
- Alle für die Test wichtigen GUI-Elemente müssen in nicht privaten Instanzvariablen - oder über entsprechende Getter-Methoden - sichtbar gemacht werden.
  ```
  catalog.getProducts().size();
  ```
- Die Bedienung der Oberfläche findet nicht über den eigentlichen Event-Mechanismus statt, sondern über spezifische Methoden der Widgets.
  ```
  editor.myDeleteButton.doClick();
  ```

Test wird an Implementierungsinterna gekoppelt !

Testen mit JUnit

Ergebnis des Tests



Testen mit JFCUnit

Was ist JFCUnit?

- Erweiterung von Junit - Open Source Project
  http://jfcfunit.sourceforge.net
- Unterstützt das Aufspüren von java.awt.Window-Instanzen(z.B. Frames und Dialoge), die vom zu testenden Code geöffnet wurden.
- Ermöglicht die Lokalisation von Swing-Komponenten im Komponentenbaum eines Fensters anhand des Typs, des Namens oder beliebiger anderer Eigenschaften.

Testen mit JFCUnit

Vorbereitung des Test

- Die Testklasse muss von JFCTestCase abgeleitet werden

```
public class CatalogEditorJFCTest extends JFCTestCase {
     ...
}
```

-
```
protected void setUp() {
     setHelper (new JFCTestHelper());
     ...
     editor = new CatalogEditor(catalog);
     editor.show();
}
```

-
```
protected void tearDown() {
     getHelper().cleanUp(this);
}
```

Testen mit JFCUnit

Löschen von Produkten

```
public void testDeleteAll() {
     FrameFinder frame_finder = new FrameFinder("Product Editor");
     JFrame editor = (JFrame) frame_finder.find();

     ComponentFinder finder = new ComponentFinder(JButton.class);
     JButton deleteButton = ( JButton ) finder.find(editor, 1);
     assertNotNull( "Could not find the Delete button",
            deleteButton );

     finder.setComponentClass(JList.class);
     JList productList = ( JList ) finder.find(editor, 0);
     assertNotNull( "Could not find the List", productList );

     ...
```

Testen mit JFCUnit

Löschen von Produkten

```
        ...

        for(int i = 1; i <= 3; i++) {
                JListMouseEventData listClick =
                 new JListMouseEventData (this, productList, 0, 1);
                getHelper().enterClickAndLeave(listClick);

                MouseEventData deleteClick =
                 new MouseEventData(this, deleteButton);
                getHelper().enterClickAndLeave(deleteClick);
        }

        assertEquals(0, productList.getModel().getSize());
    }
```

Testen mit JFCUnit

Einfügen eines Eintrags in die Produktliste

```
public void testAddProduct() {

    ...     // ab hier ist der ProductCreator Frame geöffnet
    FrameFinder frame_finder2 =
     new FrameFinder("ProductCreator");
    JFrame creator = (JFrame) frame_finder2.find();
    assertEquals("ProductCreator", creator.getTitle());

    finder.setComponentClass(JTextArea.class);
    JTextArea name = (JTextArea) finder.find(creator, 0);
    name.setText("0815");
    JTextArea description = (JTextArea) finder.find(creator, 1);
    description.setText("Netzwerkkabel");
    ...     // nun wird der AddProduct Button gedrückt
```

Testen mit JFCUnit

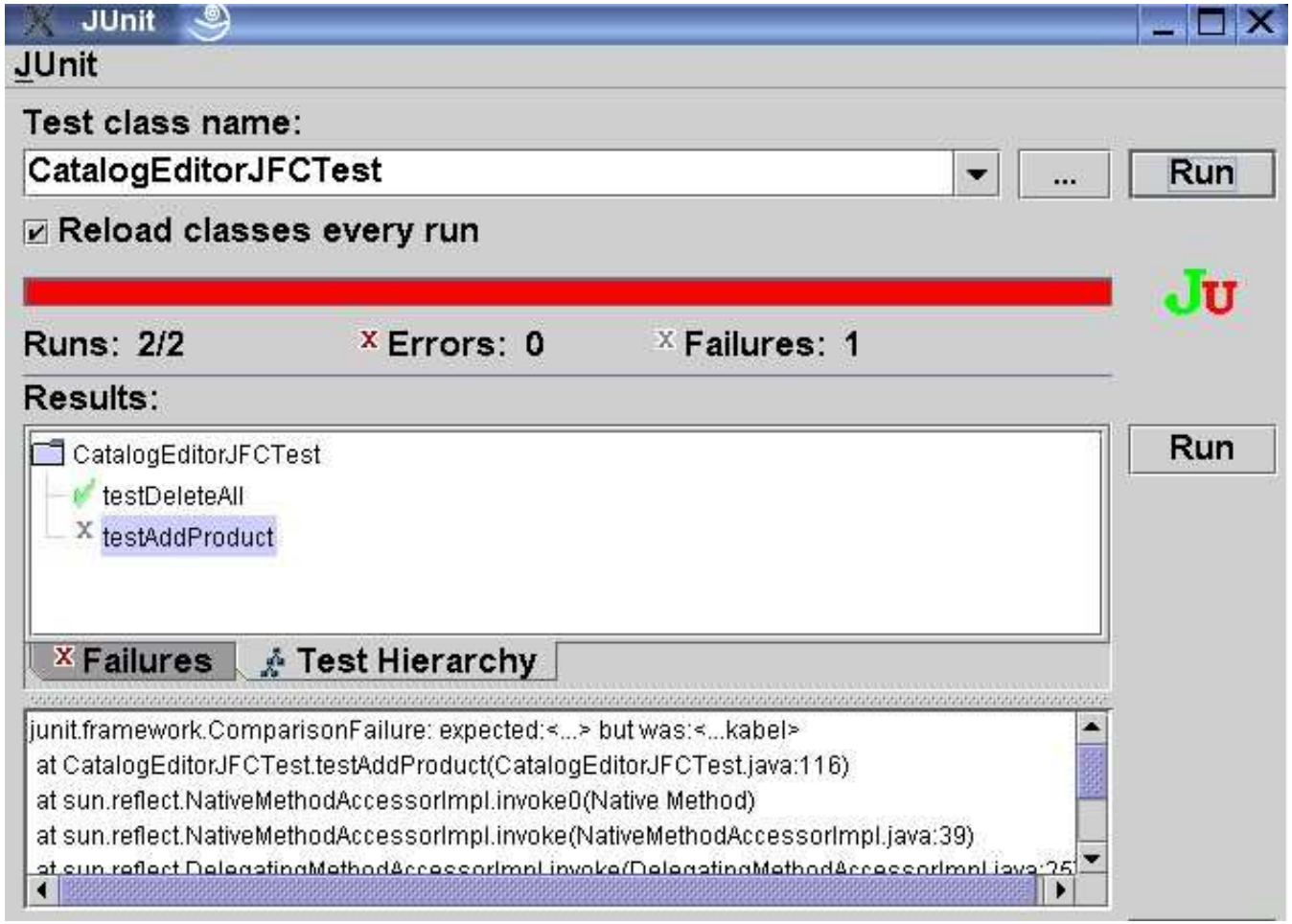
Einfügen eines Eintrags in die Produktliste

```
        ...     // wieder zurück im ProductEditor Frame
        finder.setComponentClass(JList.class);
        JList productList = ( JList ) finder.find(editor, 0);
        assertEquals(4, productList.getModel().getSize());

        int i = productList.getNextMatch("0815", 0,
         javax.swing.text.Position.Bias.Forward);
        JListMouseEventData listClick =
         new JListMouseEventData (this, productList, i, 1);
        getHelper().enterClickAndLeave(listClick);

        finder.setComponentClass(JTextArea.class);
        JTextArea descriptionArea =(JTextArea)finder.find(editor, 0);
        assertEquals("Netzwerk", descriptionArea.getText());
    }
```

Testen mit JFCUnit



Testen mit JFCUnit

Zusammenfassung

- Widgets werden nun über ihren Typ, Namen oder andere Kriterien gesucht.
- Es werden die gleichen Events ausgelöst, wie bei der "echten" Bedienung der Oberfläche.
- Entkoppeln der Tests (Logik von GUI getrennt)

Vorteile

- Test ist nahe am Geschehen der graphischen Oberfläche.
- Verhalten von mehreren offenen Fenstern und Dialogen kann getestet werden.

Nachteile

- Ungewöhnliche Features wie Drag&Drop sind noch nicht realisiert.
- Tests dauern länger und sind aufwändiger in der Implementierung.

Testen mit Abbot

Funktionalität von C&R - Test Werkzeugen

Aufnahme und Wiedergabe von

- Mouse – Ereignissen
  - Motion
  - Drag
  - Click
  - Wheel
- Tastatur – Ereignissen
- Fenster-Ereignissen
  - Resize
  - Maximize
  - Movement
  - etc.

Testen mit Abbot

Untergliederung des Testwerkzeuges "Abbot" in folgende Bestandteile

- Robot
  - Capture & Replay von Mouse - Ereignissen
  - Capture & Replay Tastatur - Ereignissen
  - Capture & Replay von Fenster - Ereignissen
- Component – References
  - Abbot entkoppelt vom Quellcode
  - Information aus der AWT - Event-Verarbeitung (.class)
  - Benutzung der Reflection - API
- Scripts
  - enthalten komplettes Testszenario
  - XML - basierend

Testen mit Abbot

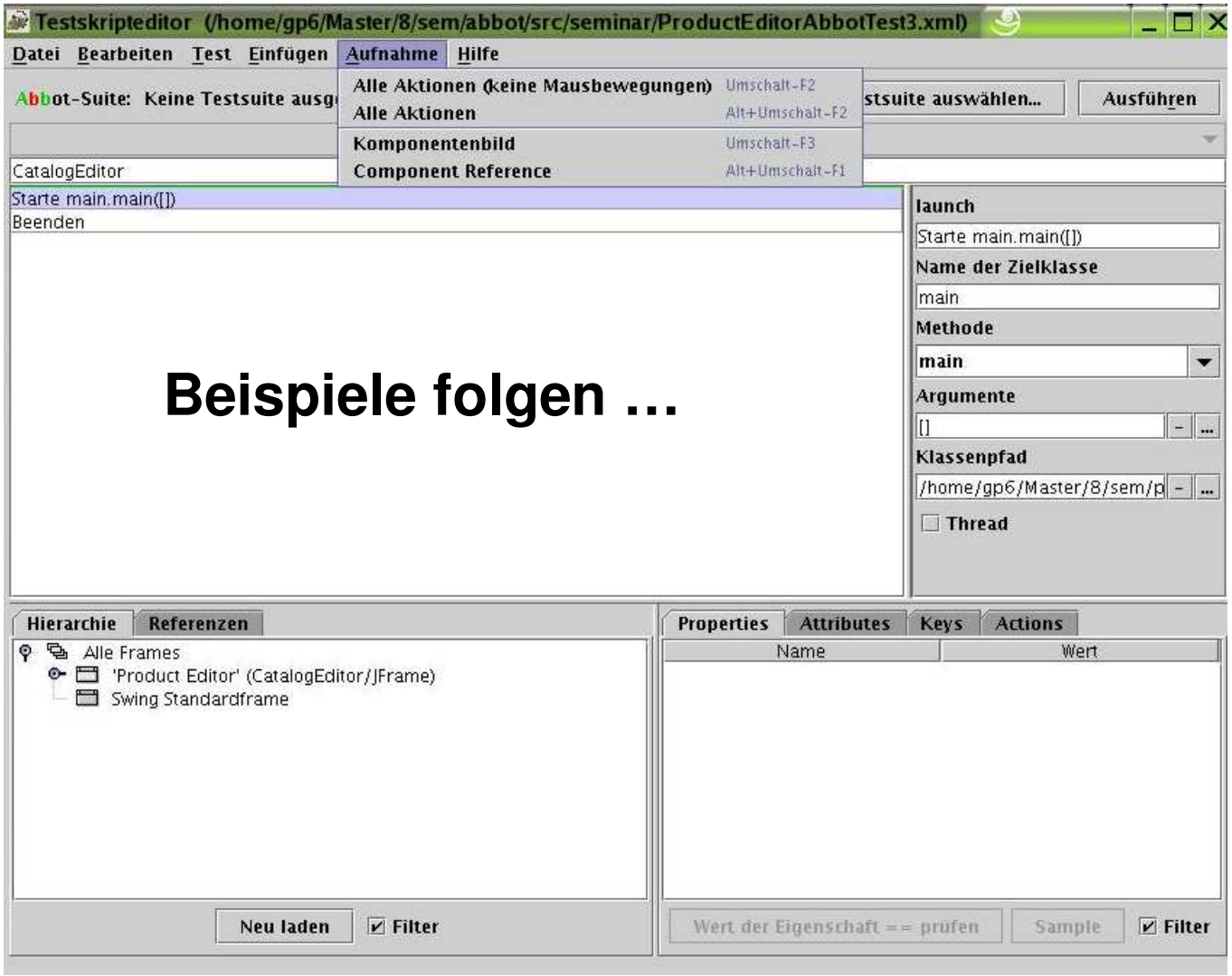
Vergehensweise bei Abbot - Tests:

- Bei Programmerstellung
  1.) Testszenario erstellen
    a.) Definition der Eventreihenfolge
    b.) Definition der entsprechenden Assert
  2.) Testszenario als XML – Datei speichern

- Bei Programmänderung bzw. Erweiterung
  1.) Testfälle um neue Funktionalität erweitern
  2.) Erneutes Testen

TEST IST JETZT AUTOMATISIERT

Testen mit Abbot



Vergleich JUnit / JFC mit Abbot

- JUnit & JFC:
  - Vorteil:
    - Exakte Kontrolle bei korrekt implementierten Tests
  - Nachteil:
    - Quellcode muss bekannt sein (Kompetenz des Testers)
    - Integration in bestehende Projekte schwierig
- Abbot (C&R-Werkzeug)
  - Vorteil:
    - Schnelles Ergänzen von Testfällen
    - Implementierung nicht zwingend erforderlich (kein Java-Code)
    - Pixelgenaues Testen möglich (Bildvergleich)
  - Nachteil:
    - Plattformabhängiges Testen bedeutet gegebenenfalls erneute Aufnahme (Testen der Pixelgenauigkeit)

Testen mit Abbot

- Weitere Abbot Testmöglichkeiten
  - Kopplung zum Quellcode durch Verknüpfung von JFCUnit mit Abbot

```
import junit.extensions.abbot.*;
```

  - JFC-Abbot Kopplung bei bereits bestehenden Projekten ungeeignet

Testen mit Abbot

- Vorteile gegenüber trivialen C&R-Werkzeugen
  - Information über Fensterhierarchie vorhanden:
    Integrationstests bei Übernahme der GUI in bestehende Applikationen
  - Neben einfachem „Replay“ auch Testfunktionalität: Funktionales Testen möglich (Aufrufkorrektheit)
- Nachteile
  - Abbot etwas unausgereift
  - Intensives Testen leider nur unter Java

Literatur

- Unit Test mit Java; Johannes Link, Peter Fröhlich; dpunkt-Verlag; Januar 2002; ISBN: 3-89864-150-3
- http://jfcunit.sourceforge.net/
- http://abbot.sourceforge.net
- www.programming-x.com
- http://www.ph-ludwigsburg.de/mathematik/personal/spannagel/jacareto/
- http://pounder.sourceforge.net/
- http://www.eventcorder.com/eventcorder.htm

Vielen Dank!