Qualitätssicherung und Testen - Teststrategien

Vorlesung: Software-Engineering für große, betriebliche Informationssysteme

für Universität Leipzig, Sommersemester 2004
Institut für Software- und Systementwicklung
Professur Wirtschaftsinformatik, insbes. Informationsmanagement

Hans Hartmann (Generali VIS Informatik Ges.m.b.H., Wien)
Wolfgang Keller (AMB Generali Informatik Services GmbH, Aachen)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

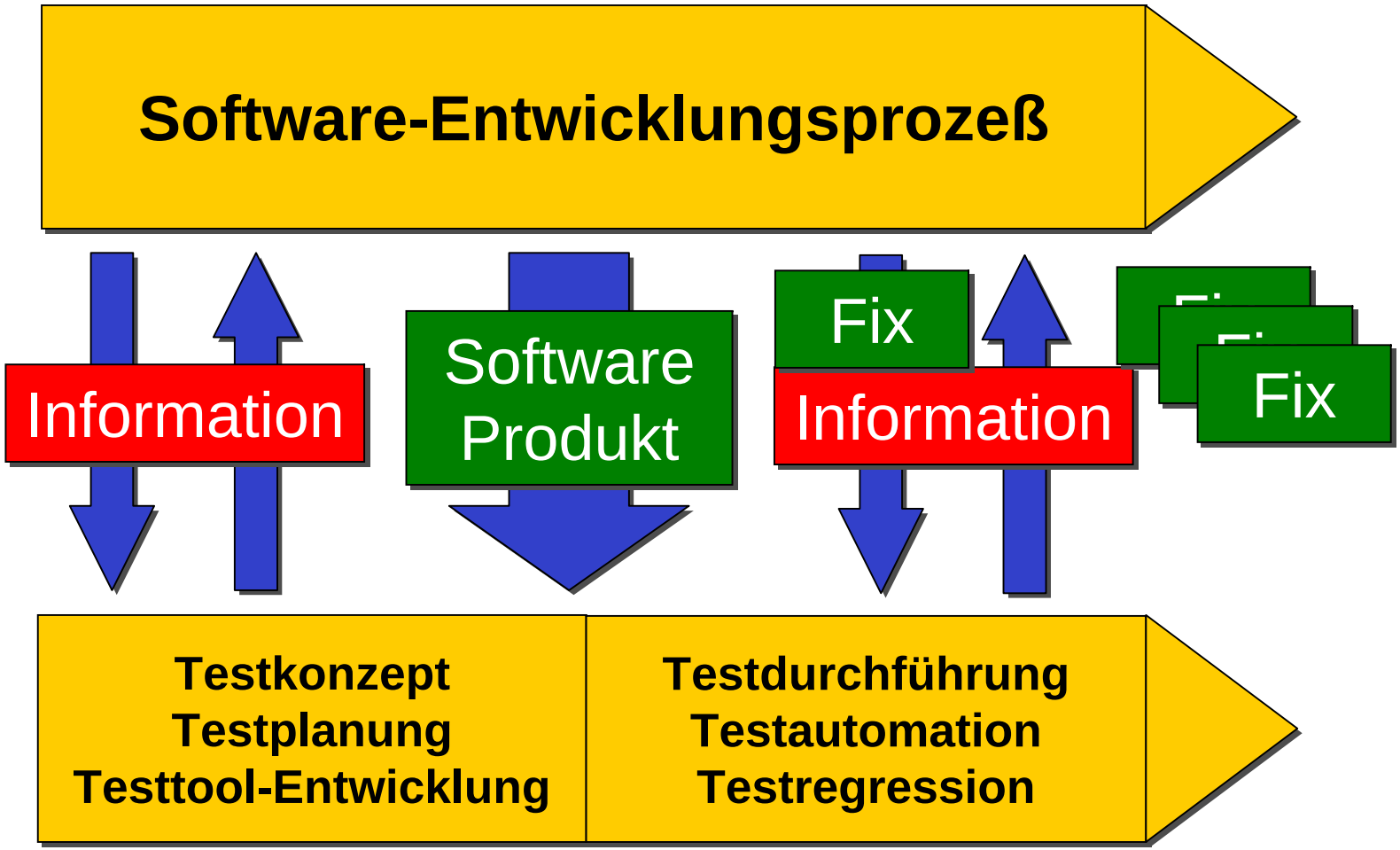
Testen: schauen, ob es funktioniert?

- Abgrenzung des Themas
- Das Black-box-Testrätsel
  - Exkurs: Der gute Testfall ?
  - Exkurs: *Äquivalenzklassen*
- Teststrategien: systematisch
  Teststrategien: *verwendet in Phoenix*
- Domain-Tests
- Stress-Tests
- Test gegen Spezifikation
- Risikotest
  - Exkurs: Risiko-Bewertung
- Test mit Zufallsdaten
- Funktionale Tests
- Regressionstests
  Szenario-Tests
  - Exkurs: *Szenario*
- Anwender-Tests
  - Exkurs: *explorativer Test*
- Teststrategien in Phoenix: *revisited*

Anmerkung: in diesen Folien wird immer wieder auf die Testanwendung anhand des Projektes Phoenix verwiesen. Diese Lebensversicherungsanwendung war eine Grossanwendung mit 12000 Function-Points. Das Projekt galt in den 90er-Jahren als das größte OO-Projekt Europas in Smalltalk.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Teststrategien: Teil der Planung und des Konzepts



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

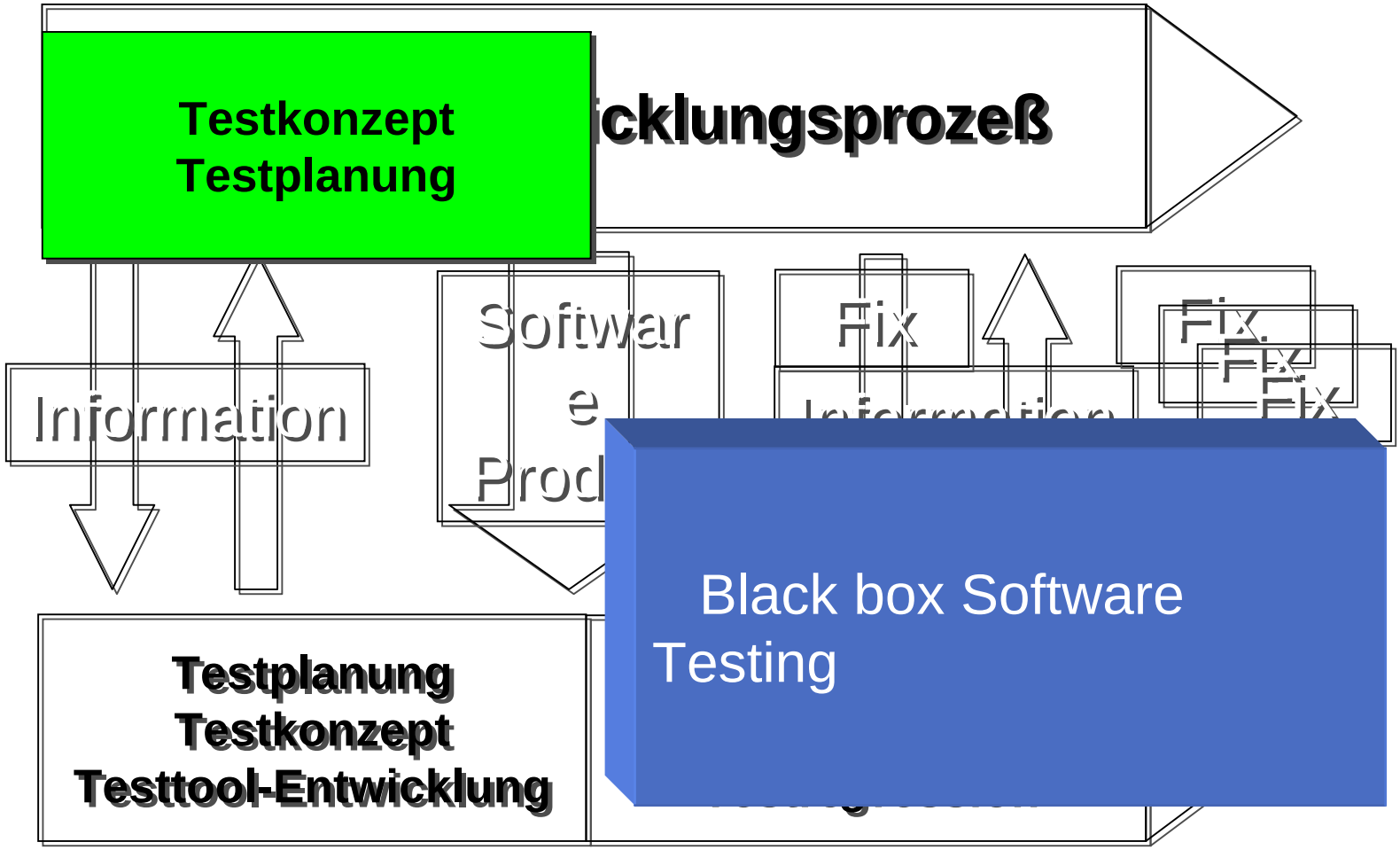
Teststrategien: Teil der Planung und des Konzepts

Testkonzept
Testplanung

Testplanung

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Teststrategien: Teil der Planung und des Konzepts



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Das Rätsel des Black-box-Testens

- Verschiedene Testgruppen testen unterschiedlich (auch wenn es sich um gleichartige Produkte handelt)
- Manche Testgruppen halten das, was andere Testgruppen testen, für unwichtig.
- Selbst wenn zwei Testgruppen aus fachlich und technisch versierten (und motivierten) Personen zusammengesetzt sind, sehen die Tests ganz unterschiedlich aus.

- Warum ist das so?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Antwort 1: Die Paradigmenfalle

„Paradigma“

- 1. *Muster,*
- *2. Denksystematik,*
- *3. Denksystem*
- *oder vielleicht nur Modewort ?*

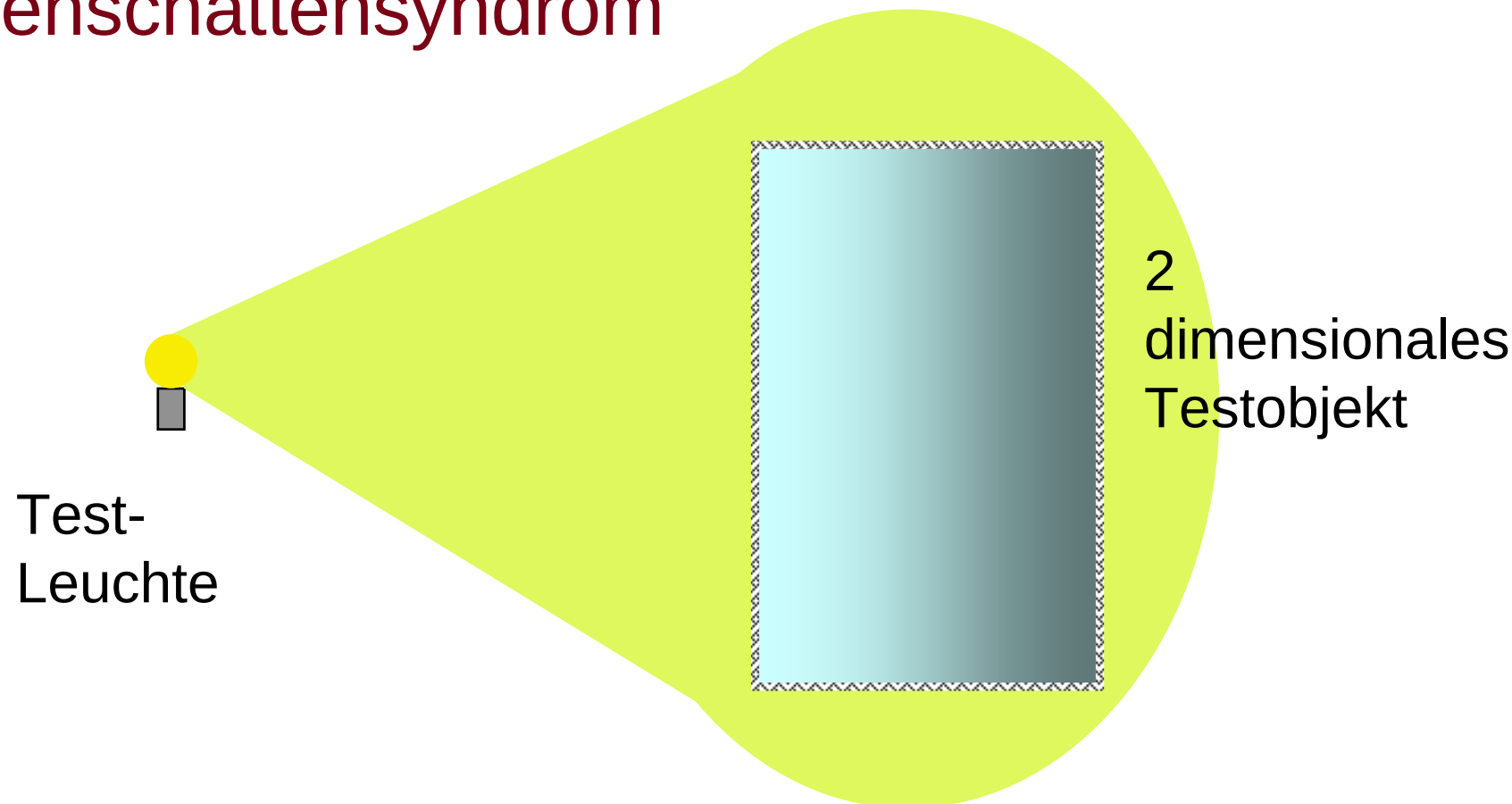
- Verwechslung von einer Methodik (*oder Strategie*) mit einem Paradigma als Gesamtdenksystem

daraus folgend

Alle Untersuchungen werden nur im Sinne einer Methodik geführt!

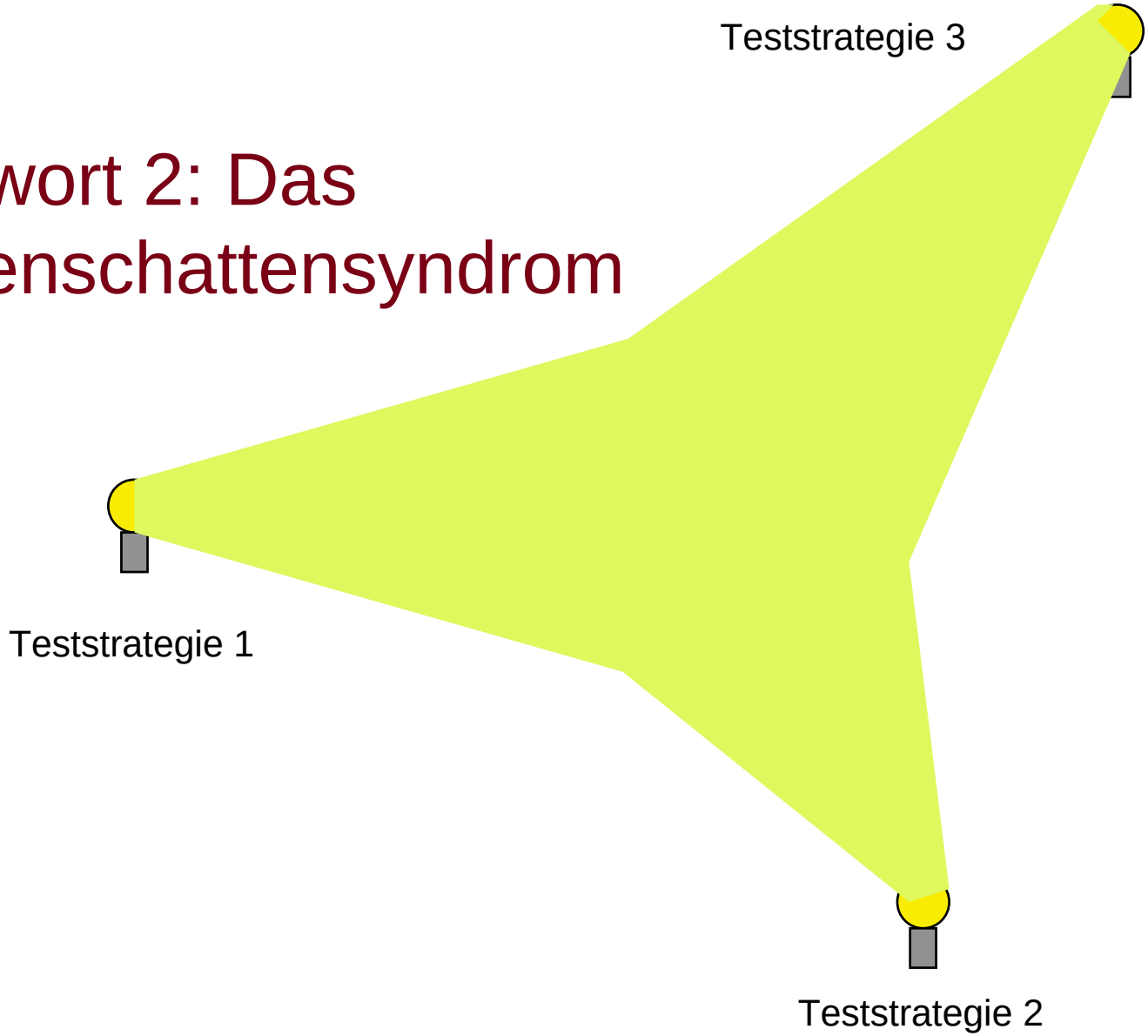
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Antwort 2: Das Eigenschattensyndrom



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Antwort 2: Das Eigenschattensyndrom



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein guter Testfall

- Besitzt eine Wahrscheinlichkeit > 0, einen Fehler zu finden.
- Ist nicht redundant.
- Ist der beste seiner Art.
- Ist weder zu kompliziert noch zu einfach.
- Deckt fehlerhaftes Programmverhalten leicht sichtbar auf.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Äquivalenzklassen 1: Klassifizierung Test

Verschiedene Tests gehören zu einer Äquivalenzklasse, wenn gilt:

- Alle testen die gleichen Objekte.
- Wenn ein Test einen Fehler findet, so finden auch alle anderen Tests einen Fehler.
- Wenn ein Test einen bestimmten Fehler nicht findet, so werden auch die restlichen Tests diesen Fehler nicht finden.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Äquivalenzklassen 2: Klassifizierung Test

Für äquivalente Tests gilt auch:

- Sie verwenden die gleichen Eingabevariablen.
- Die im Programm aufgerufenen Funktionen sind dieselben.
- Sie sprechen dieselben Ausgabevariablen an.
- Kein Test verursacht Error-Handling oder jeder Test verursacht Error-Handling.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Äquivalenzklassen 3: Beispiel Testobjekt

Eingabe oder Ausgabe Event	Äquivalenzklasse "gültig"	Äquivalenzklasse "ungültig"
Zahl	Zahlen zwischen 1 und 99	0 >99 Ein arithmetischer Ausdruck, der eine ungültige Zahl ergibt: z.B. 5-5 negative Zahlen Buchstaben restliche Sonderzeichen
Erster Buchstabe eines Nam	Großbuchstabe Kleinbuchstabe	Zahl Sonderzeichen
Zeichnen einer Linie	1 pixel Breite - 4 cm Länge	keine Linie Länger als 4 cm keine Gerade (Kurve)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Black-Box Teststrategien

Domain-Tests	"domain driven"
Stress-Tests	"stress driven"
Tests gegen Spezifikation	"specification driven"
Risiko-Tests	"risk driven"
Tests mit Zufallsdaten	"random / statistical tests
funktionale Tests	"function driven"
Regressionstests	"regression tests"
Szenario-Tests	"scenario / use case / transaction flow driven"
Anwender-Tests	"user testing"
Sicherheits-Tests	"security testing"
...	...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Black-Box Teststrategien in Phoenix

	Priorität	
Domain-Tests	A1	Risiko-Tests
Stress-Tests	A2	Szenario-Tests
Tests gegen Spezifikation	A3	Tests gegen Spezifikation
Risiko-Tests	A4	Regressionstests
Tests mit Zufallsdaten	A5	Sicherheits-Tests
funktionale Tests		
Regressionstests	B1	Stress-Tests
Szenario-Tests	B2	Anwender-Tests
Anwender-Tests		
Sicherheits-Tests	Entw1	Domain-Tests
...	Entw2	funktionale Tests
	nicht in Phoenix	~~Tests mit Zufallsdaten~~

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Domain-Tests

Schlagworte

- Wertebereiche
- Optionen

Vorgangsweise

- Testobjekte strategisch plazieren
- Representative Werte aus großen Wertebereichen festlegen.

Musterbeispiele

- Äquivalenzanalyse von einfachen numerischen Eingabefeldern
(Eingabebereich: *Zahlenbetrag in VA-Leben, Punkt-Komma-Regelung scheint erst bei großen Beträgen auf. NLS.*)
- Drucker-Kompatibilitätstests.
(Beispiel aus der Industrie: HP-Kompatibilität.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Domain Tests

Stärken	Schwächen
die am wahrscheinlichsten auftretenden Fehler können mit einem relativen kleinen Testaufkommen erreicht werden. (Verhalten des Programmes bei fehlerhafter Eingabe kann sehr „erhellend sein“.) Klares intuitives Vorgehen, generelle Verwendbarkeit	„Blinde Flecken“ Fehler, die sich nicht an Wertegrenzen befinden oder welche sich in speziellen Optimierungszweigen befinden. (Beispiel: Häufigkeitsoptimierung, Wertebereich=[1..100], aber 27 tritt in 98% aller Eingaben auf) • Oft sind die tatsächlichen Wertebereiche nicht bekannt. • (Beispiel: *SAP-Interface*)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Stress-Tests

Schlagworte

- das Produkt überlasten

Vorgangsweise

- Das Produkt wird in den atypischen Fehlerfall oder in die Überlastung getrieben.
- Ziehen von Rückschlüssen auf das Programmverhalten unter normalen Umständen.

Musterbeispiele

- Hohes Datenaufkommen, LAN / WAN-Verbindungsabbruch, lange Transaktionsketten

(Katastrophentests bei Phoenix zeigen ungünstiges Verhalten bzgl. CICS-Restart auf.)

- Geringe Memory-Reserven, Rechnerausfall, Viren-Resistenz.

(Laufzeitverhalten bei Version-1 des Virtual Workers führt zum Erkennen von in TOPLINK angesiedelten Problemen.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Stress-Tests

Stärken

Die Schwächen, die normalerweise erst beim länger dauernden Echteinsatz auftreten, werden rechtzeitig erkannt.

(Allerdings müssen die gefundenen Fehler auch behoben werden :)

Schwächen

„Blinde Flecken“

- Viele Fehler treten unabhängig von der Belastung auf. Der Stress-Test macht sie daher nicht „sichtbarer“.

  (Beispiel: In Phoenix: Modellierungsfehler, Regelfehler)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Test gegen Spezifikation

Schlagworte

- *„Überprüfen wir, ob alles geht, was wir versprochen haben ...“*

Vorgangsweise

- Das Produkt wird auf jede Spezifikations- oder Designaussage überprüft.
- Das Pflichtenheft wird wieder durchgesehen.

Musterbeispiele

- Abdeckungsmatrix: stellt den Zusammenhang zwischen Testfällen und Spezifikationsdetails dar.
(in Phoenix: Abdeckungsmatrix über Testserien, Produkt-. Und Tarifkategorisierung.)
- In der Industrie kann das Pflichtenheft auch die Testabnahmeprotokolle spezifizieren..
(Beispiele: Serverbelastungsfähigkeit, garantierte Response-Zeiten, Fail-safe-Specs in kritischen Umgebungsbereichen.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Test gegen Spezifikation

Stärken	Schwächen
Beste Absicherung gegen Garantie und Gewährleistungsansprüche. Erhalt der Glaubwürdigkeit beim Kunden selbst beim Fehlerauftritt.	• Bei Design- und damit verbundenen Spezifikationsschwächen überlebt das Produkt den Test genau dort, wo das Produkt seine Fehler hat. • (Beispiel in Phoenix: Testfall arbeitet gegen allgemeine Plattform-Spezifikation. Ein gefundener Fehler führt zum CR, Besprechung mit dem Kunden und zur Ungültig-Markierung des Fehlers. Ein Jahr später wird der Fehler vom Kunden mit hoher Dringlichkeit moniert.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Risiko-Tests

Schlagworte

- „Zuerst die schlimmen Fehler finden!“

Vorgangsweise

- Der Testaufwand wird schwerpunktsmäßig dorthin verlegt, wo auftretende Fehler den größten Schaden anrichten können.

Musterbeispiele

- Equivalenzklassen-Behandlung nach erweiterten Kriterien. (Bei Generali kann fast jeder Fehler fatal sein, daher wird zwischen schlimmen und noch schlimmeren Fehlern unterschieden.)
- Stress-, Fehlerbehandlung-, Sicherheitstests. Tests, die speziell auf gefürchtete Fehler ausgerichtet sind.
- Auswahl aus der „voraussagbaren“ Fehlerliste. (Musterfehler aus bekannten Projekten mit ähnlicher Architektur)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Risiko-Tests

Stärken	Schwächen
Beste Testpriorisierung, vorausgesetzt, daß die Risiko-Analyse korrekt gemacht wird.	• Tests versagen bei Risiken, die überaschend höher sind als erwartet. • Manche Risiken können in der Testanalyse übersehen worden sein. • (Beispiel, welches immer in den unterschiedlichsten Umgebungen auftritt: Protokolldateien werden zu lange und bringen das eigentliche Programm zum Absturz. Oder Transaktions-Logs werden zu lang. Unter OS/2 konnte eine Übergröße der Swapper-Datei den ganzen Rechner crashen! Meistens werden diese Fehler auch beim Stress-Test übersehen.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Risiko-Bewertung 1

- Team aus **Spezialisten verschiedener Gruppen** zusammengesetzt
- Segmentierung des Programmes mit Hilfe von Brainstorming mit Auflistung der einzelnen Programmbereiche und Testgegenstände
- Schätzen und Priorisierung der einzelnen Testbereiche nach den Fragen:
    - Wahrscheinlichkeit, daß ein Fehler auftauchen kann
    - Folgeschaden und Schwere (Severity) des jeweiligen Fehlers
- Priorisierung der Tests nach der Kategorie Risiko

Vorgangsweise nach Bob Stahl, siehe Kaner & Bach, Paradigms, TCS-1999, Washington

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Risiko-Bewertung 2

- Auflisten aller „schlimmen“ Dinge, die bei Anwendung des Programms passieren könnten.
- Auflisten der Erscheinungsbilder, die damit verbunden sind.
- Auflisten der Tests, die für die Abdeckung der obigen Fehlerfälle notwendig sind.
- Die Listen in den Review stellen und speziell nach Feedback hinsichtlich der Priorisierung fragen.

siehe Kaner & Bach, Paradigms, TCS-1999, Washington

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Risiko-Bewertung 3

- Analyse von Projektfaktoren, Produktelementen und Qualitätskriterien speziell in Hinblick auf Risikofaktoren.
- Testen auf die Gebiete mit höchstem Risiko verlegen
- Verwenden der Testresultate, um die Risikoanalyse zu verfeinern.
- Keine totale Vernachlässigung der „Low risk“-Bereiche, da die ursprüngliche Analyse auch unvollständig sein kann.

siehe Kaner & Bach, Paradigms, TCS-1999, Washington

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Tests mit Zufallsdaten

Schlagworte

- Massentests mit immer neuen Testdaten

Vorgangsweise

- Testrechner erzeugt, führt aus und evaluiert eine große Anzahl von Tests.

Musterbeispiele

- Tests von Funktionen oder Subsystemen auf der Basis von vorhersagbaren Resultaten.
- Testaussage aufgrund statistischer Berechnungen (technischer Abdeckungsgrad.)
- Stochastisches Testen von Status-Übergängen. (Servertests, Telefonnetzwerke)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Tests mit Zufallsdaten

Stärken

Regression ist unabhängig von den alten „abgelutschten“ Testdaten

- Test mit Beschränkung auf partielle Ergebnisvorhersagen können speziell bei neuen Programmteilen schneller Ergebnisse liefern.
- Übersehen von Sonderfällen weniger wahrscheinlich. (siehe Domain Test)
- Findet Fehler, die infolge langer komplexer Bearbeitungsketten entstehen. (Ketten, die schwer als Testfälle zu planen sind)

Schwächen

Schwierigkeit bei der Diskriminierung zwischen „ok“ und „not ok“

(not crash = not fail) = false

- Da diese Tests viele verschiedene Fehlerarten finden und dazu noch automatisch eine Regression bereitstellen, wird die Notwendigkeit von weiteren Tests, die sich der Automatik entziehen, oft übersehen.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Funktionale Tests

Schlagworte

- „Black box Unit Tests“

Vorgangsweise

- Jede Einzelfunktion des Programms wird analysiert und getestet.

Musterbeispiele

- Tests von Spreadsheets (Test jeder Einzelfunktion)
- Tests von Datenbanken (Test jedes Berichts, jeder Abfrage, etc.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Funktionale Tests

Stärken

- Genaue Betrachtung jeder getesteten Einheit (= getesteten Funktion)
- Einsatzbereich bereits vor Integrations- und Systemtest, daher werden die Fehler früher gefunden.
- (Die Frage ist, ob die Entwickler, wenn sie selbst Funktionstests planen, wirkliche „Black box“-Mentalität an den Tag legen können.)

Schwächen

- Zusammenspiel der Funktionen wird nicht getestet. Für Systemtests ungeeignet.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Regressionstests

Schlagworte

- jedesmal neu testen, wenn sich der Code geändert hat.

Vorgangsweise

- Überprüfung, ob ein Fehlerfix funktioniert und ob er keine Seiteneffekte hat.

Musterbeispiele

- Überprüfen der Testfälle, die zum ursprünglichen Fehler geführt haben.
- Automatische GUI-basierte Regressionstests (der simulierte Anwender)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Regressionstests

Stärken

- Regressionstests beruhigen, wiederholte Tests ermöglichen auch leichtere Planung von Programmveränderungen (Code change, Code refactoring)
- Diese Tests sind vertrauens-bildend, Protokolle können leichter für Testnachweis bereitgestellt werden.
- Wahrscheinlichkeit, daß Fehler in einer neuen Version bei Einsatz von RT gefunden werden ist 5-20%. (neue Version / Portierung)

Schwächen

- Funktionen, die nicht im Regressionstest getestet werden, weisen nicht die gleiche Zuverlässigkeit auf.
- Wartung der Tests kann extrem kostspielig werden.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Szenario-Tests

Schlagworte

- „etwas Vernünftiges testen“

Vorgangsweise

- Diejenigen Abläufe des Programms werden getestet, die der echten Verwendung des Programms entsprechen.

Musterbeispiele

- Das Programm wird hinsichtlich der Arbeit mit echten Kundendaten und Geschäftsregeln, anhand von „use cases“ betrieben
- Das Programm wird dahingehend „abgeklopft“, ob es die Spezifikationen des Konkurrenzproduktes beherrscht.
- „use cases“ eignen sich gut für automatische Regressionstests

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Szenario-Tests

Stärken

- Komplexe, realistische Programmabläufe. Manche Abläufe entziehen sich bis zum tatsächlichen Test einer Modellierung.
- Fehler können gefunden werden, die erst aufgrund von Testfallketten (aufeinander folgende Programmdurchläufe) entstehen.

Schwächen

- Einzelne funktionale Fehler des Programms können den gesamten Test torpedieren. („maskierendes Verhalten“)
- Sorgfältigste Planung notwendig, um ausreichende Testabdeckung zu bewerkstelligen.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Szenario-Tests: Charakteristika

- Mehrere Fähigkeiten des Programms werden kombiniert getestet.
- Eindeutige Aussage ist möglich, ob das Programm richtig funktioniert hat.
- Mehrere Spezialisten müssen übereinstimmen, daß die Nichterfüllung eines Szenarios eine entscheidende Einbuße der Programmverwendung darstellt.

In Phoenix wird schwerpunktmäßig automatischer Szenario-Test, bzw. Szenario-Regressionstest betrieben.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Anwender-Tests

Schlagworte

- „realitätsbezogen testen“ - Das Programm wird schließlich von Menschen bedient.

Vorgangsweise

- Diejenigen Zusammenbrüche werden festgestellt, die bei „echter“ Verwendung des Programmes entstehen, d.h. wenn die Kette Mensch-Computer-Programm auseinanderbricht.

Musterbeispiele

- Beta-tests
- unspezifierte manuelle Szenario-Tests vor Einführung der automatischen Tests.
- DAU-Tests (siehe explorativer Tests)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Anwender-Tests

Stärken

- Design-Fehler werden sehr schnell sichtbar.
- Diejenigen Schwachstellen des Programms werden aufgezeigt, die zu Unverständnis führen oder hohe Fehleranfälligkeit seitens des Benutzers nach sich ziehen.
- Inhouse-Beta-Tests erlauben gezielte Usability-Studien über Video-Monitoring und Capturing Tools.

Schwächen

- Testabdeckung ist ungewiß.
  (Siehe Erfahrung aus unzähligen kommerziellen Produkten.)
- Die Qualität der Testfälle ist ungesichert. Testfälle könnten zu trivial sein, um subtile Fehler finden zu können.
- Beta-Testen kostet Geld.
  (Entweder über Vertrieb und Imagekosten oder über die Bezahlung der eigenen Inhouse-Beta-Abteilung.)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Anwender-Tests explorativ

Schlagworte

- „gleichzeitig lernen, planen und testen“

Vorgangsweise

- Das Programm kommt ohne Dokumentation zum Tester. Dieser muß gleichzeitig das Programm durchschauen und entsprechende Testfälle und Teststrategien entwickeln, die das Programm ausreichend durchleuchten und seine Fehler offenlegen.

Musterbeispiele

- Ad-hoc Tests für schnellste Auslieferungszyklen. (leider auch eine Realität)
- Third party software (der Horror der Testabteilung)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Anwender-Tests explorativ

Stärken	Schwächen
• Wenn der Tester das Programm betreiben kann, kann er auch strategisch feststellen, ob es für den Anwender betreibbar ist.	• Desto weniger man vom Programm / Produkt kennt, desto mehr kann (wird!) man übersehen.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Black-Box Teststrategien in Phoenix

	Priorität	
Domain-Tests	A1	Risiko-Tests
Stress-Tests	A2	Szenario-Tests
Tests gegen Spezifikation	A3	Tests gegen Spezifikation
Risiko-Tests	A4	Regressionstests
Tests mit Zufallsdaten	A5	Sicherheits-Tests
funktionale Tests		
Regressionstests	B1	Stress-Tests
Szenario-Tests	B2	Anwender-Tests
Anwender-Tests		
Sicherheits-Tests	Entw1	Domain-Tests
...	Entw2	funktionale Tests
	nicht in Phoenix	~~Tests mit Zufallsdaten~~

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004