GEBIT Solutions

Die Experten für Java, Anwendungsentwicklung und Requirements Engineering

Vergleich von Open Source GUI Test-Frameworks in der Praxis

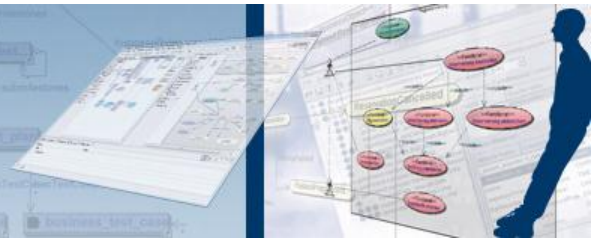
„Viel Schatten – aber auch viel Licht“

Dehla Sokenou

Agenda

- **Testverfahren und Testwerkzeuge – der Versuch, einen Überblick zu geben**
- **Werkzeuge für automatische GUI Tests**
- **Fallbeispiele aus der Praxis**
  - Erfahrungen mit den Werkzeugen Abbot, Selenium, FitNesse
- **Fazit**

Positionierung der GEBIT Solutions



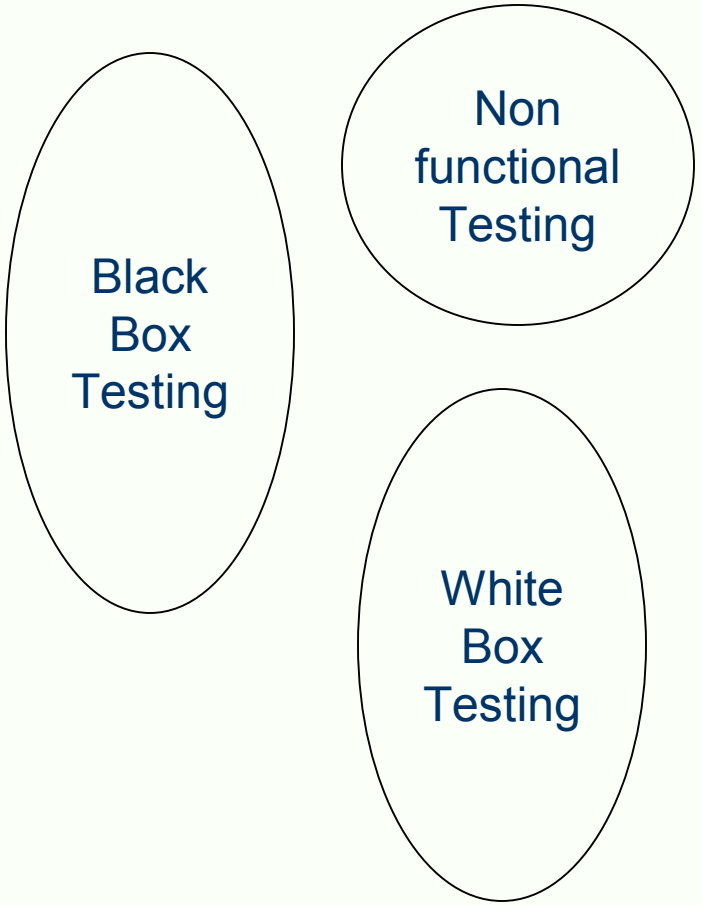
- **Projektorientiertes Software- und Beratungshaus**
  - eigene Forschung und Entwicklung
- **Partner der Fach- und IT-Abteilung**
  - Vom Consulting und Coaching bis hin zur Übernahme der kompletten Ergebnisverantwortung
  - Projektumsetzung in Time und Budget, mit hoher Qualität
- **Seit 1991 auf Objekttechnologien spezialisiert**
- **Seit 1996 effektiver Einsatz von Java in der betrieblichen AE**
  - Kunden sind meist Großunternehmen und große internationale Mittelständler
    - Z.B. Bayer, C&A, dm, Esprit, OBI, Lufthansa
- **Standorte Berlin, Düsseldorf, Stuttgart**
  - 90 Mitarbeiter

Ebenen von Tests

Level

- **System**
  - Testen eines Gesamtsystems in Bezug auf die Anforderungen
- **Integration**
  - Testet das Zusammenspiel von Modulen
- **Unit**
  - Test unabhängiger kleiner Einheiten

Verfahren



Kategorien von Testwerkzeugen

- T

GUI Testing

- Web
- Rich Clients

- **Debugging und Profiling**
- **Testdatengenerierung**
- **Code-Analyse**
- **Performance- und Stresstesting**
- **Siehe auch: http://www.opensourcetesting.org**

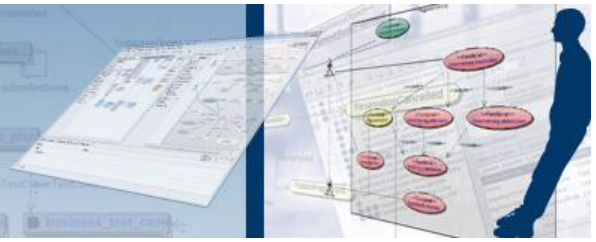
Anteater CubicTest WatiN IdMUnit Canoo WebTest
Arbiter Blerby Test Runner Frankenstein EMOS Framework httest Avignon
aşk Autonet Harness Selenium Doit: Simple Web Application Testing
Concordion GITAK Crosscheck AutoTestFlash Enterprise Web Test
Apodora FitNesse DejaGnu Eclipse TPTP Abbot Expect GNU/Linux
Desktop Testing Project Canoo WebTest DbFit Imprimatur Watir DBFeeder
Dogtail

GEBIT Solutions

Die Experten für Java, Anwendungsentwicklung und Requirements Engineering

Werkzeuge für automatische GUI Tests

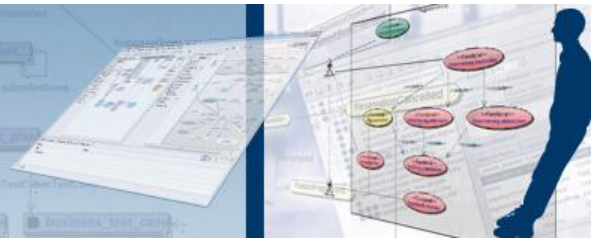
Automatische GUI Tests
Anforderungen



Wichtigkeit

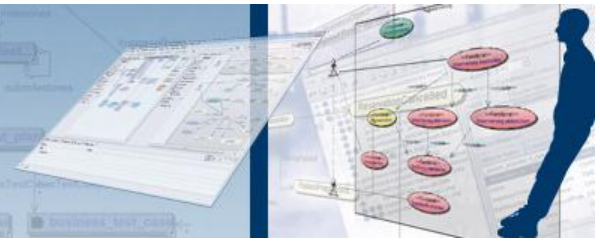
- **Einfache Erstellung von Tests**
  - Einbindung von Testern aus dem Fachbereich
  - -> ohne Experten-IT-Wissen verwendbar
- **Einfaches Erstellen von „Zusicherungen“**
- **Zuverlässigkeit**
  - Tests müssen zuverlässig abspielbar sein
- **Möglichkeit, alle Arten von Interaktionen zu testen**
- **Beständigkeit bei Änderungen in der Software**
- **Möglichkeit der Strukturierung von Testsuiten**
- **Einbettung in Build Prozess (Continuous Integration)**
- **Reporting der aufgetretenen Fehler**
- **Performante Ausführung**

Automatische GUI Tests
Verfahren



- **Capture & Replay**
  - Erzeugen von Testfällen durch die Aufzeichnung von Interaktionen mit der Benutzeroberfläche
  - Typische Werkzeuge: Abbot+Costello, Marathon, Jacareto, TestGen4Web, Selenium, WebTst, v.a.m.
- **„Skript“-gesteuert**
  - Verwendung einer möglichst einfachen Skript-Sprache (idealer Weise Anforderungsdokumente), zur Beschreibung von Testfällen
  - Typische Werkzeuge: Apodora, Concordion, FitNesse, v.a.m.
- **In der Praxis: oft eine Kombination aus beiden Verfahren**
  - Record+Play-Tools erzeugen ein Skript
  - Skript-orientierte Werkzeuge bieten eine Unterstützung zur automatischen Erstellung der Skripte

Automatisierungsverfahren
Vor- und Nachteile



	Capture+ Replay	**Skript-gesteuert**
Einfachheit der Erstellung	⊕ Erst mal sehr einfach	⊖ Erst mal aufwändiger
Beständigkeit bei Änderungen	⊖ Schlechter, da wenig abstrakt	⊕ Besser, abstrakterer Level der Beschreibung
Testbare Interaktionen	⊕ Umfangreich, da allgemeine Plattform-Events getestet werden	⊖ Testen von Spezialinteraktionen erfordern oft Anpassungen
Performante Ausführung	⊖ Oft schlechter Test fein granularer Events	⊕ Oft besser Test von High-Level Events

Beispiel: Record+Play Werkzeug „Marathon“

```
if window('TREND UML Runner'):
        click('GTreeControl$TreeArea', 53, 21)
        click('GTreeControl$TreeArea', 54, 39)
        click('Search')
        if window('Information'):
                click('OK')
        close()
```

„Low-level“ Events aufgezeichnet

Beispiel: Skript-basiertes Werkzeug „FitNesse“

Create 1st flight

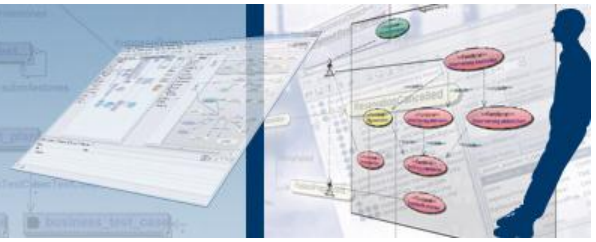
set values					
flightNumber	airline.code	departureTime	price	departureLocation	destinationLocation
LX123	LH	12:00:00	1234.00	Berlin-Tegel	Hamburg-Hamburg-Airport

Associate the airline

press	browse.airline

High Level Interaktionen per Skript beschrieben

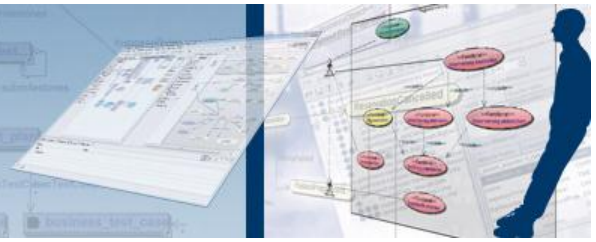
Automatisierung
Web-Clients versus Rich-Clients



- **Rich-Clients**
  - Komplexes Eventmodell
    - Fein granulare Events
    - Asynchrone Ausführung
  - Einfache „Adressierung von Komponenten“
    - Klares Komponentenmodell
    - i.d.R. eindeutig über einfache Namen erreichbar

- **Web Client**
  - Einfaches Eventmodell
    - HTTP Request-Reply Protokoll
    - Aber: Komplexität Eventmodell nähert sich Rich-Client an
    - Verwendung von Ajax + JavaScript
  - Problematische „Adressierung“ von Komponenten
    - XPath-Ausdrücke
    - Anpassung der Seite fürs Testen

Automatisierte GUI Werkzeuge
Beispiele



	Abbot	FitNesse	Marathon	TPTP (Eclipse)	Selenium
Verfahren	Record+ Play	Skript	Record+ Play	Record+ Play	Beides
Primäre Plattform	Swing	Beliebig	Swing	SWT	Web
Skript-sprache	XML	Wiki-Syntax	Jython, JRubi	XML	Diverse
IDE	Nein	Wiki	Ja	Ja	Ja
Version	1.0.2	Dec. 2008	2.0.3 (Oct. 2008)	Inoffiziell	1.0b2

Nur, um ein SWT Beispiel zu haben

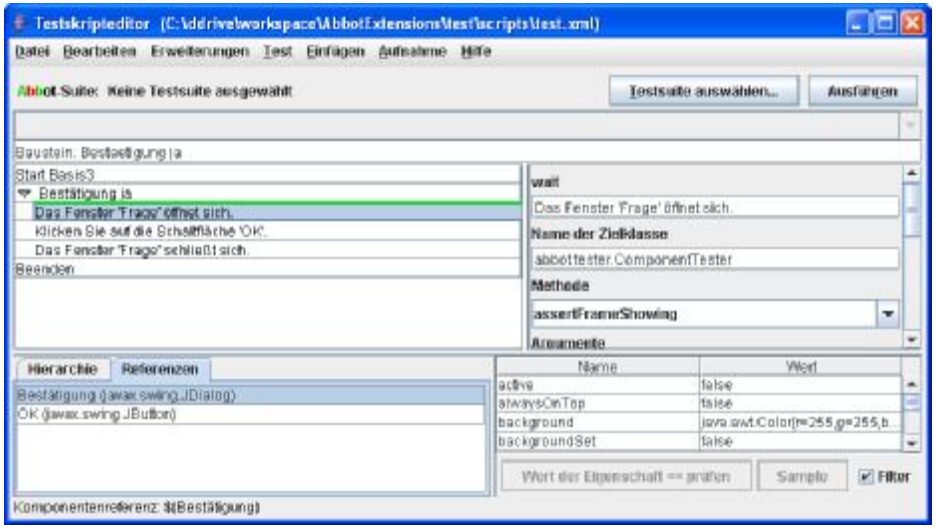
GEBIT Solutions

Die Experten für Java, Anwendungsentwicklung und Requirements Engineering

Fallbeispiele aus der Praxis

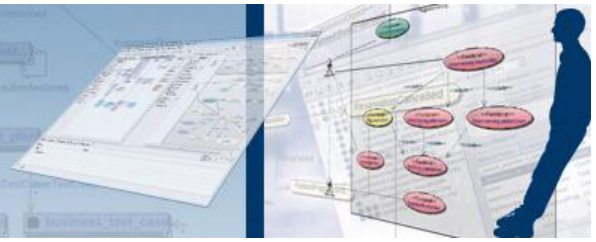
Werkzeug 1: „Abbot“

- **Werkzeug „Abbot“ & „Costello“**
  - Automatisierungswerkzeug für „Capture & Replay“ von Swing Anwendungen



- **Fallbeispiel**
  - Regressionstest einer Warenwirtschaftsanwendung
  - Suite mit 16000 umfangreichen Testfällen
  - Ablaufzeit 8 Stunden
  - Erstellungszeit der Testsuite > 5 PJ
  - Größtenteils erfahrene Java-Entwickler / Architekten in enger Zusammenarbeit mit dem Fachbereich

Werkzeug 1: Fazit beim Einsatz von Abbot



- **Positiv:**
  - Sehr guter Regressionstest mit hoher Testabdeckung
  - Dadurch Test neuer Releases (Datenbank, JDK, WWS-Versionen) sehr zuverlässig

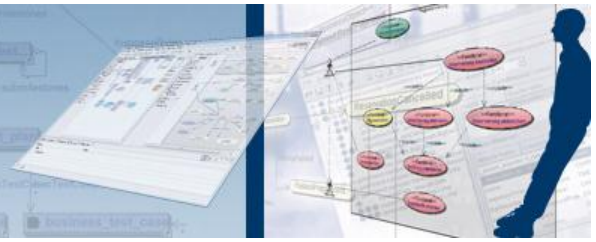
- **Negativ:**
  - Vor der produktiven Verwendung diverse technische Anpassungen von Abbot notwendig
  - Lange Laufzeiten Testsuite
  - Reproduzierbarkeit von Tests war schwer zu erreichen
  - Aufwendige Reproduktion von Fehlerfällen
  - Aufwendig, Test-Suite gegen Programmänderungen stabil zu machen

Werkzeug 2: „FitNesse“

- **Werkzeug „FitNesse“**
  - Ein Collaborations-Werkzeug zum gemeinsamen Erstellen von Acceptance Tests

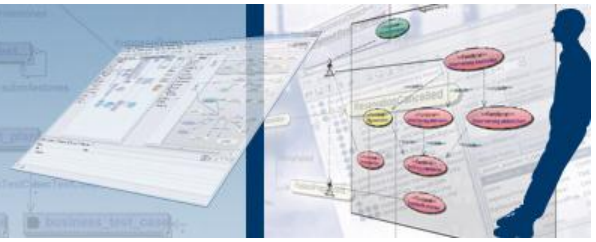
- **Verwendung bei GEBIT**
  - Einsatz in diversen Projekten
  - Test modellbasierter Anwendungen

Fallbeispiel: Test einer Logistikanwendung
Merkmale der Anwendung



- **Java EE**
  - JBoss Applicationserver, Apache Webserver, JSF, mobile Geräte via WLAN eingebunden
- **Dauer**
  - ~ 2 Jahre Projektdauer, ~ 10 Personenjahre
  - ~ 10% = ~ 1 Personenjahr FitNesse Testaktivitäten
- **Umfang**
  - ~ 100 Use-Cases mit zugeordneten Aktivitäten
  - ~ 2000 Java-Klassen (~ 120 Business-Klassen)
  - ~ 200000 LOC
  - ~ 100 Datenbanktabellen
- **Ziele**
  - Haupttestsuite jeden Tag ohne Fehler laufen lassen
    - Regressionstests: Vermeidung von neuen Fehlern im getesteten Code
    - Nebeneffekt: Jederzeit ein stabiles Produkt an den Kunden lieferbar

Fallbeispiel: Test einer Logistikanwendung
Durchgeführte Tests und Ergebnisse



- **Ca. 30 generische Fixtures[1] im Projekt entwickelt**
- **Testsuiten**
  - Haupttestsuite: 350 Tests mit 17450 Assertions
  - Zusätzliche Testsuite: 220 Tests mit 15000 Assertions
  - ~ 500 Anpassungen der Testcases (~ 2 Testcases/Tag)
- **Coverage**
  - Haupttestsuite: 100% Coverage der Use-Cases, Aktivitäten und deren Transitionen
- **Gefundene Fehler**
  - **~ 500 von 800 Bugs im Issue-Tracking-System**

[1] Adapter der Anwendung an das Test-Framework

Verwendung von FitNesse – Ein Beispiel

```
|de.gebit.fitnesse.samples.Sample1|
|numerator|denominator|quotient?  |
|10       |2          |5          |
|12.6     |3          |4.2        |
|100      |4          |33         |
|100      |0          |error      |
```

Erfassung: Wiki Syntax

de.gebit.fitnesse.samples.Sample1		
numerator	denominator	quotient?
10	2	5
12.6	3	4.2
100	4	33
100	0	error

Test schlägt fehl

Ergebnisprotokoll

de.gebit.fitnesse.sampl .Sample1		
numerator	denominator	quotient?
10	2	5
12.6	3	4.2
100	4	33 *expected* 25.0 *actual*
100	0	error

Erwartung: Test liefert Exception (!), schlägt aber nicht fehl

FitNesse Fixtures

de.gebit.fitnesse.samples.Sample1		
numerator	denominator	quotient?

Standard-Input-Verhalten

Standard-Output-Verhalten
(inklusive Ergebnisprüfung)

```
/**
 * Sample 1 FitNesse test (division).
 */
public class Sample1 extends ColumnFixture {

    public double numerator;
    public double denominator;

    /**
     * Simple fixture code for division test.
     */
    public double quotient() {
        return Sample1Division.divide(numerator, denominator);
    }
}
```

Getestetes Objekt

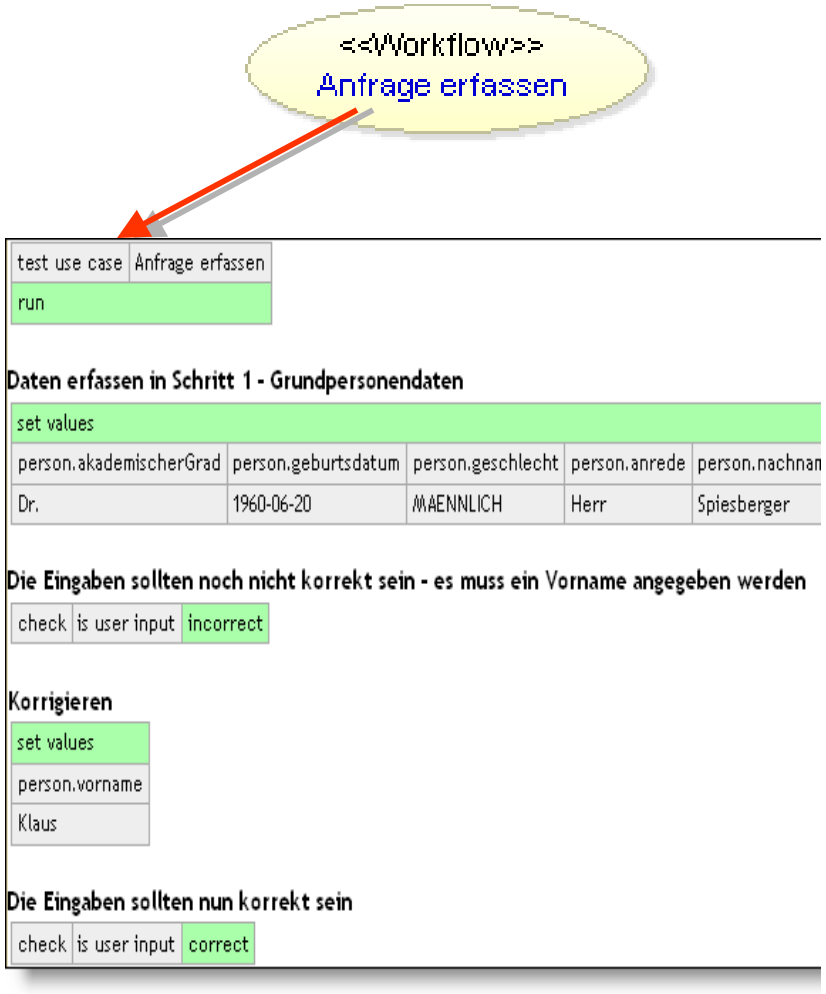
Werkzeug 2: Fazit beim Einsatz von FitNesse

- **Positiv:**
  - Sehr guter Regressionstest – Test neuer Releases zuverlässig
  - Einbindung von „Nicht-Programmierern“ in die Erstellung von Test-Suiten war möglich
    - Allerdings nur mit entsprechendem Support
  - Hoher Abstraktionsgrad der Test-Suiten macht diese robuster gegen Programmänderungen

- **Negativ:**
  - Umfangreiche initiale Anpassung von FitNesse erforderlich
  - Test von speziellen Interaktionen erfordert Programmieraufwände
    - Drag&Drop, spezielle Controls, ...
  - Miserable Qualität des FitNesse Codes (Dokumentation) machen Anpassungen aufwendig

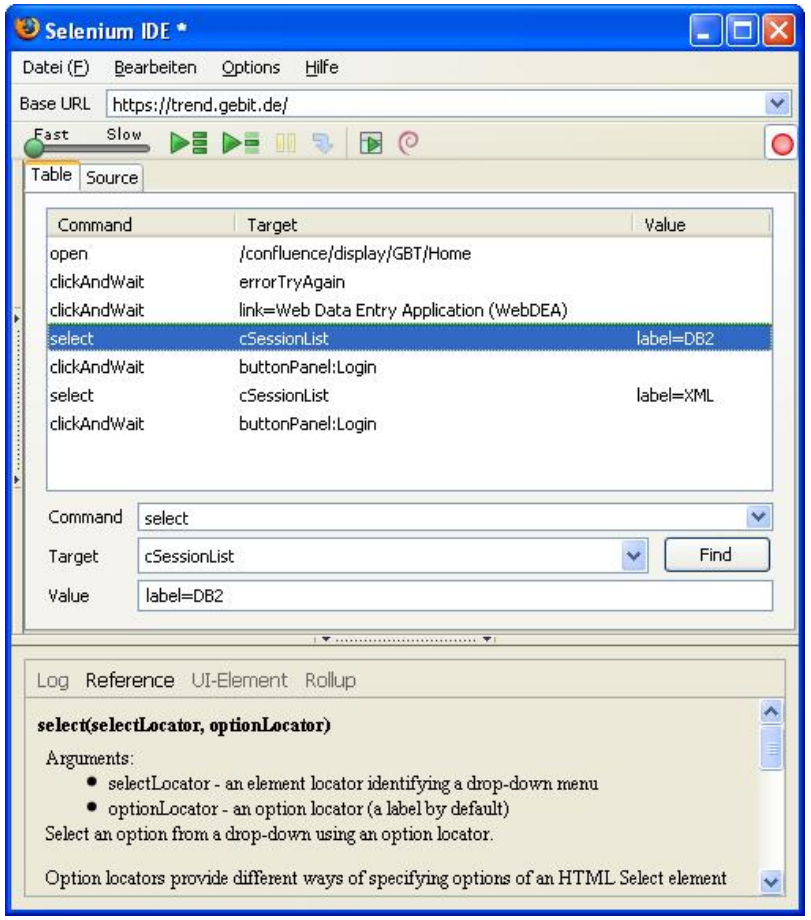
Werkzeug 2: Anmerkung zum Test modellbasiert erstellter Anwendungen

- **Modellbasierte Entwicklungsansätze vereinfachen automatisierte Testverfahren**
  - Anwendungsfallbeschreibungen
  - Beschreibung von Anwendungsabläufen über Aktivitätsdiagramme
  - Zusicherungen, die im Modell fest gelegt sind
- **„Smoke-Tests“ für eine Anwendung können vollständig generiert werden**

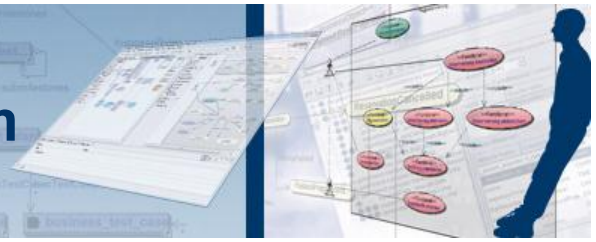


Werkzeug 3: „Selenium“

- **Werkzeug: „Selenium“ & Selenium IDE**
  - Werkzeug zum automatisierten Test von Web-Oberflächen
  - Installiert sich als spezieller Treiber im Browser (Firefox oder Internet Explorer)
  - dadurch gute Simulation des Verhaltens spezieller Browser
- **Fallbeispiel:**
  - Test von Web-Oberflächen für ein JSF-basiertes Framework
  - Kleinere Testsuite (einige 100 Fälle)
  - Ablaufzeit wenige Minuten
  - Erstellen der Testsuite einige PM

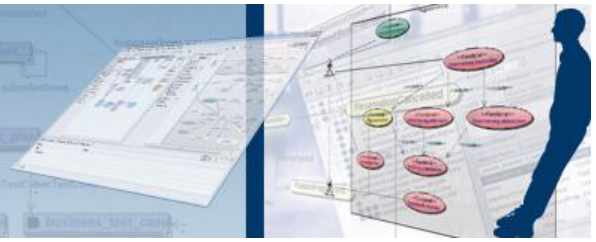


Beispiel 3: Fazit beim Einsatz von Selenium



- **Positiv:**
  - „Getting Started“ – schnell und einfach
  - „Selenium IDE“ (Record & Play) erleichtert die Verwendung auch durch „Nicht-Programmierer“
- **Negativ:**
  - Hohe Fragilität der Testsuite bei Änderungen im Design der Web-Oberflächen
  - Nur durch deutlich mehr Aufwand bei der Erstellung der Testfälle vermeidbar
  - Beim Test von sehr dynamischen Web-Oberflächen lässt die Reproduzierbarkeit von Tests zu wünschen übrig

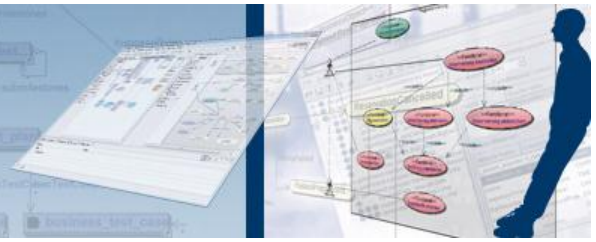
Automatisierte GUI Tests
Erfahrungen



- **Kosten zum Erstellen von Testsuiten werden oft unterschätzt**
- **Schulungsbedarf für die Tester wird oft unterschätzt**
- **Erwartungshaltung: Kurzfristig sehr produktiv zu werden**
  - Automatisierte Tests zahlen sich insbesondere bei der Auslieferung mehrerer Releases einer Software aus
- **100%-Automatisierung oft nicht möglich – manuelle Tests nötig**
- **Testdokumentation ist essentiell**
- **Testplanung ist essentiell**
- **Tests sollten modularisiert werden**
  - Auch wenn das nicht mit allen Werkzeugen einfach ist
- **Implementierung der Tests sollte wie jedes andere IT-Vorhaben geplant werden**

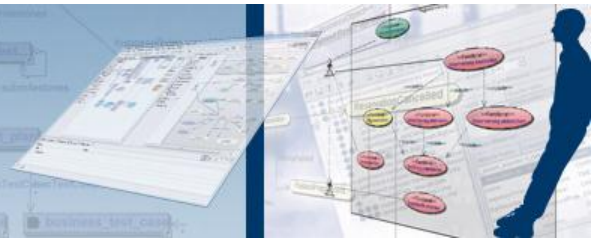
Bei Beachtung dieser Faktoren ist Kosten-Nutzen-Verhältnis positiv!

Fazit



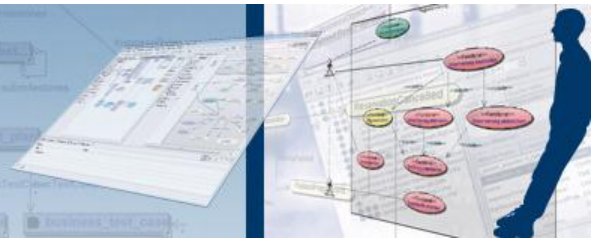
- **Der Einsatz von Werkzeugen für Testautomation lohnt sich**
  - vor allem bei lang laufenden Projekten
  - oder bei einer Produktentwicklung
- **Werkzeuge aus dem Open Source Bereich durchaus geeignet für Testautomatisierung**
- **Aber: Verwendung „out of the box“ meist nicht möglich:**
  - Anpassungsaufwände der Werkzeuge an Zielumgebung und Aufwende für Testerstellung sollten nicht unterschätzt werden
- **Skript-orientierte Verfahren in unserer Wahrnehmung mit leichten Vorteilen gegenüber Capture&Replay Verfahren**
- **Zuverlässige Abspielbarkeit oft „nicht einfach erreichbar“**
- **Und: beim Schreiben von Anwendungen denkt oft keiner daran, wie diese getestet werden sollen**

Automatisierte GUI Tests
Ausblick



- **Einfache Eingabesprachen für scriptbasierte Werkzeuge**
  - "Endnutzerfreundlichkeit"
- **Kombination mit modellbasierten Entwicklungsframeworks**
  - Scriptbasierte Testwerkzeuge: Informationen aus dem Modell zur Testdefinition verfügbar machen
    - Welche Aktion ist wann sinnvoll / überhaupt möglich?
  - Direkte Aufzeichnung nicht von GUI-Events, sondern von Modellinformationen
    - Aktuell bei GEBIT: Modellbasierter FitNesse-Recorder
- **Wie verhindert man das "Verrotten" von Testfällen?**
  - Bisher: Entkopplung Testfälle – System
  - Refactoring mit dem System statt ständige nachträgliche Pflege der Tests

Zeit für Ihre Fragen



Danke für Ihre Aufmerksamkeit!

GEBIT Solutions

Technologie beherrschen ▶ Prozesse gestalten ▶ Lösungen schaffen ▶▶