Testen von Webanwendungen

Mirja Hasse
Fabian Stich

Agenda

1. Motivation und Unterschiede
2. Methoden
3. Werkzeuge
4. Fazit

1. Motivation

Unterschiede zu konventionellen Tests

- heterogene Benutzergruppen
- Antwortzeiten (Anzahl der Zugriffe,...)
- Erreichbarkeit (Hosting, Server,...)
- Aktualität (News-Portale)

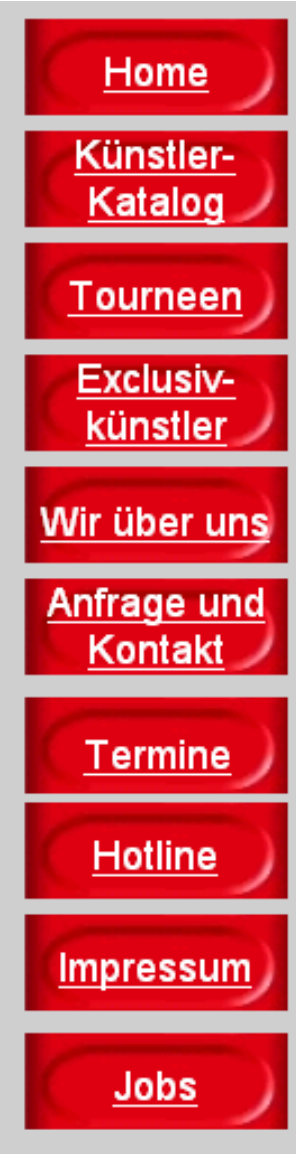
1. Motivation

Unterschiede zu konventionellen Tests

- Sicherheit (Shops, Online-Banking,...)
- Vielfalt der Plattformen
- Browser-Kompatibilität

1. Motivation

Beispiel Browser-Kompatibilität



1. Motivation

Web-spezifische Fehlerquellen

- Fehler im Content
  Schwierig zu finden, da Referenzsystem (Orakel) benötigt wird
- Ästhetik
  Keine konventionellen Methoden hierfür geeignet
- Netzwerkeigenschaften
  Netzwerkbandbreite, unterschiedliche Netzwerkanbindungen (Modem, ISDN, DSL, LAN)

1. Motivation

Web-spezifische Fehlerquellen

- Juvenilität und Multidisziplinarität
  Häufig junges Team, deshalb geringe Testbereitschaft, häufig fehlende Erfahrung

- Globalität (Mehrsprachigkeit / Useability)

1. Motivation

Beispiel: Globalität (www.aljazeera.net)

1. Motivation

Web-spezifische Fehlerquellen

- verschiedene Software von verschiedenen Unternehmen
  Datenbank, Webserver, Middleware

- Immaturität
  Häufig neue Technologien, ohne geeignete Testmethoden oder Testwerkzeuge

- Änderungsdynamik
  Häufig erneutes Testen derselben Funktionalität

1. Motivation

Zusätzlich

- restriktive zeitliche Bedingungen
- 24/7 Betrieb

Agenda

1. Motivation und Unterschiede
2. **Methoden**
3. Werkzeuge
4. Fazit

Konventionelle Methoden

müssen angepaßt und erweitert werden

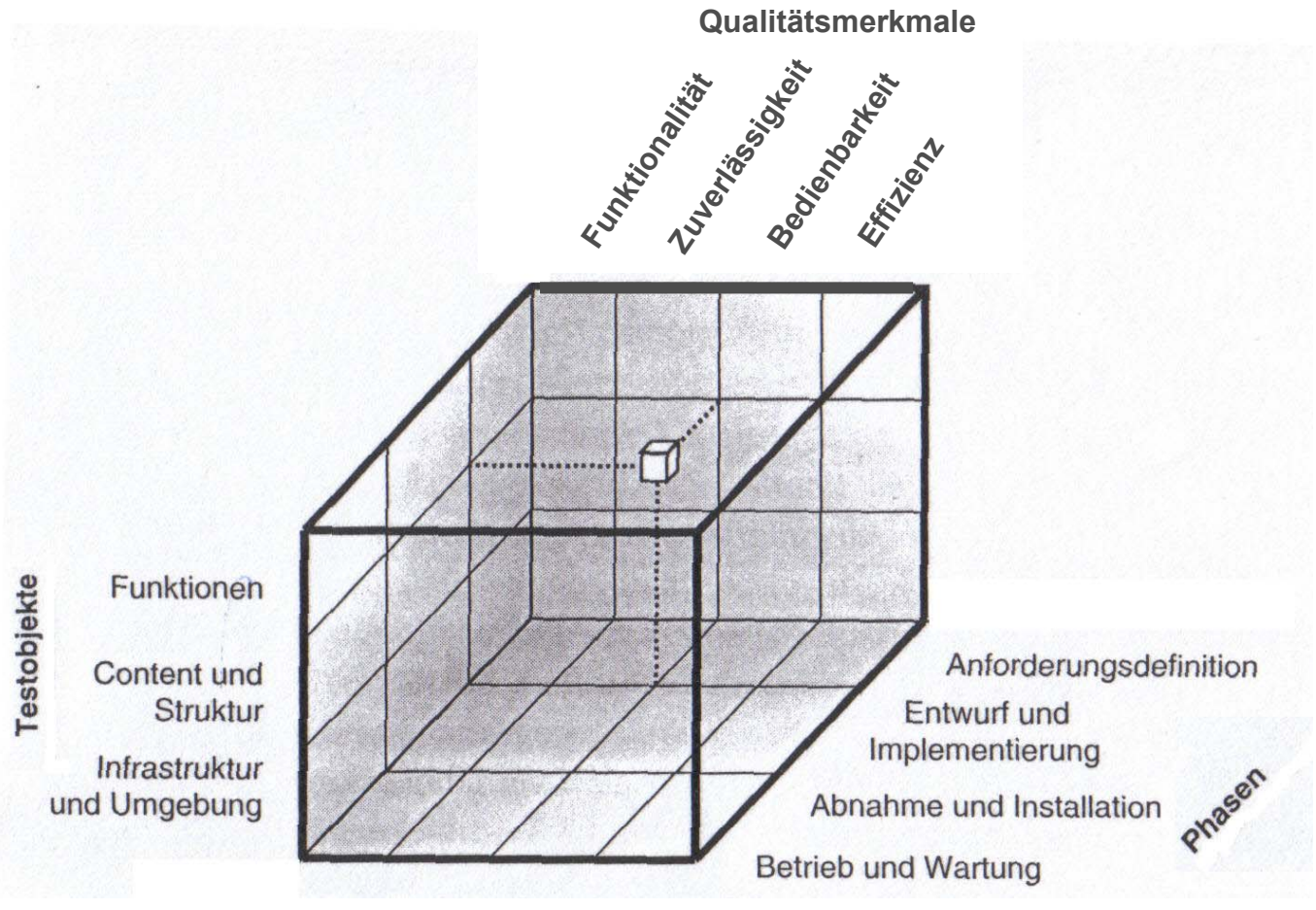
- Abstimmung mit den Erfordernissen des Projekts (Browser-Kompatibilität...)
- Kurze Entwicklungszyklen → nur die wichtigsten Ergebnisse auswählen, Arbeitsschritte streichen

2. Methoden

Web-spezifische Testmethoden

- Linktesten → spürt broken links und orphaned pages auf
- Browsertesten → deckt Fehler auf, die durch Inkompatibilität zu verschiedenen Browsern entstehen
- Testen der Sicherheit → Testet Qualitätsmerkmale wie
  - Vertraulichkeit
  - Autorisierung
  - Authentifikation
  - Zurechenbarkeit
  - Integrität

2. Methoden


Qualitätsmerkmale
Funktionalität
Zuverlässigkeit
Bedienbarkeit
Effizienz
Testobjekte
Funktionen
Content und Struktur
Infrastruktur und Umgebung
Anforderungsdefinition
Entwurf und Implementierung
Abnahme und Installation
Betrieb und Wartung
Phasen

2. Methoden

		Funktionen	Content und Struktur	Infrastruktur und Umgebung
Funktionalität	**Angemessenheit**	Reviews und Inspektionen, testgetriebene Entwicklung	Checklisten, Lesetechniken, Style-Guides, Reviews	
	Richtigkeit	Capture/Replay-Verfahren, testgetriebene Entwicklung	Statische Analyse, Link-Testen, Lesetechniken, Reviews	Statische Analyse, Linktesten
	Interoperabilität	Browser- und Plattform-Kompatibilitätstests	Testdrucken, Kompatibilitätstests, Checklisten, Reviews	Browser- und Plattform-Kompatibilitätstests
	Ordnungsmäßigkeit	Kompatibilitätstests, Style-Guides, testgetriebene Entwicklung	Checklisten, Kompatibilitätstests, Style-Guides, Reviews	Browser- und Plattform-Kompatibilitätstests
	Sicherheit	Analyse typischer Attacken, Reviews und Inspektionen		Analyse typischer Attacken, Fehlerfalltest, ethisches Hacken

2. Methoden

		Funktionen	Content und Struktur	Infrastruktur und Umgebung
Zuverlässigkeit	Reife	Dauertest		Dauertest
	Fehlertoleranz	Fehlerfall- und Ausfalltest		Stresstest, Fehlerfall- und Ausfalltest, Wiederanlauftest
	Wieder-herstellbarkeit	Fehlerfall- und Ausfalltest		Fehlerfall- und Ausfalltest, Wiederanlauftest

2. Methoden

		Funktionen	Content und Struktur	Infrastruktur und Umgebung
Benutzbarkeit	**Verständlichkeit**	Benutzertest, Heuristische Evaluierung	Statische Analyse der Lesbarkeit, Benutzertest	
	Erlernbarkeit	Benutzertest, Heuristische Evaluierung		
	Bedienbarkeit	Benutzertest Heuristische Evaluierung		Heuristische Evaluierung
	Attraktivität		Publicity-Tests	

2. Methoden

		Funktionen	Content und Struktur	Infrastruktur und Umgebung
Effizienz	Zeitverhalten	Last- und Stresstests, Monitoring und Überwachung		Last- und Stresstests, Monitoring und Überwachung
	Verbrauchs-verhalten	Dauertest	Test der Ladezeiten	Dauertest, Monitoring und Überwachung

2. Methoden

Last-, Stress- und Dauertest

- Basieren auf ähnlichen Prinzipien
- messen Antwortzeiten und Durchsatz bei mehreren parallelen Anfragen
- Anfragen werden erzeugt von „Lastgeneratoren“

2. Methoden

Stresstest

- Test auf kontrolliertes Reagieren in „Stressituationen“
- Simulation durch unrealistisches Überlasten oder stark schwankende Belastung
- Ziel:
    - Werden geforderte Antwortzeiten bzw. Durchsatz erreicht?
    - Wird ggf. mit entsprechender Fehlermeldung reagiert?

Dauertest

- Das System wird über einen längeren Zeitraum einer Belastung ausgesetzt
- Ziel: z.B. Aufdeckung von Speicherlecks

2. Methoden

Lasttest

- bringt das System die geforderten Antwortzeiten und den geforderten Durchsatz?

- Vorgehensweise:

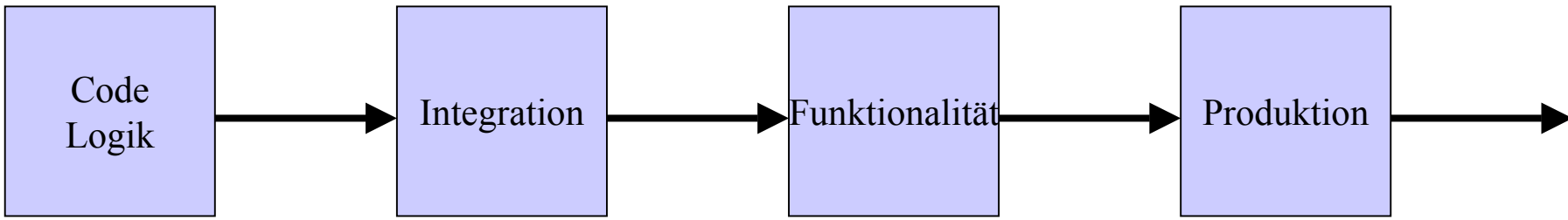
    1. Ermittlung von Lastprofilen
    2. Ermittlung von Zielen für Antwortzeiten und Durchsatz
    3. Durchführung der Tests und Auswertung

Agenda

1. Motivation und Unterschiede
2. Methoden
3. **Werkzeuge**
4. Fazit

3. Werkzeuge

Übersicht



Mock-Objekte

JUnit

Cactus

HTTPUnit

Capture & Replay

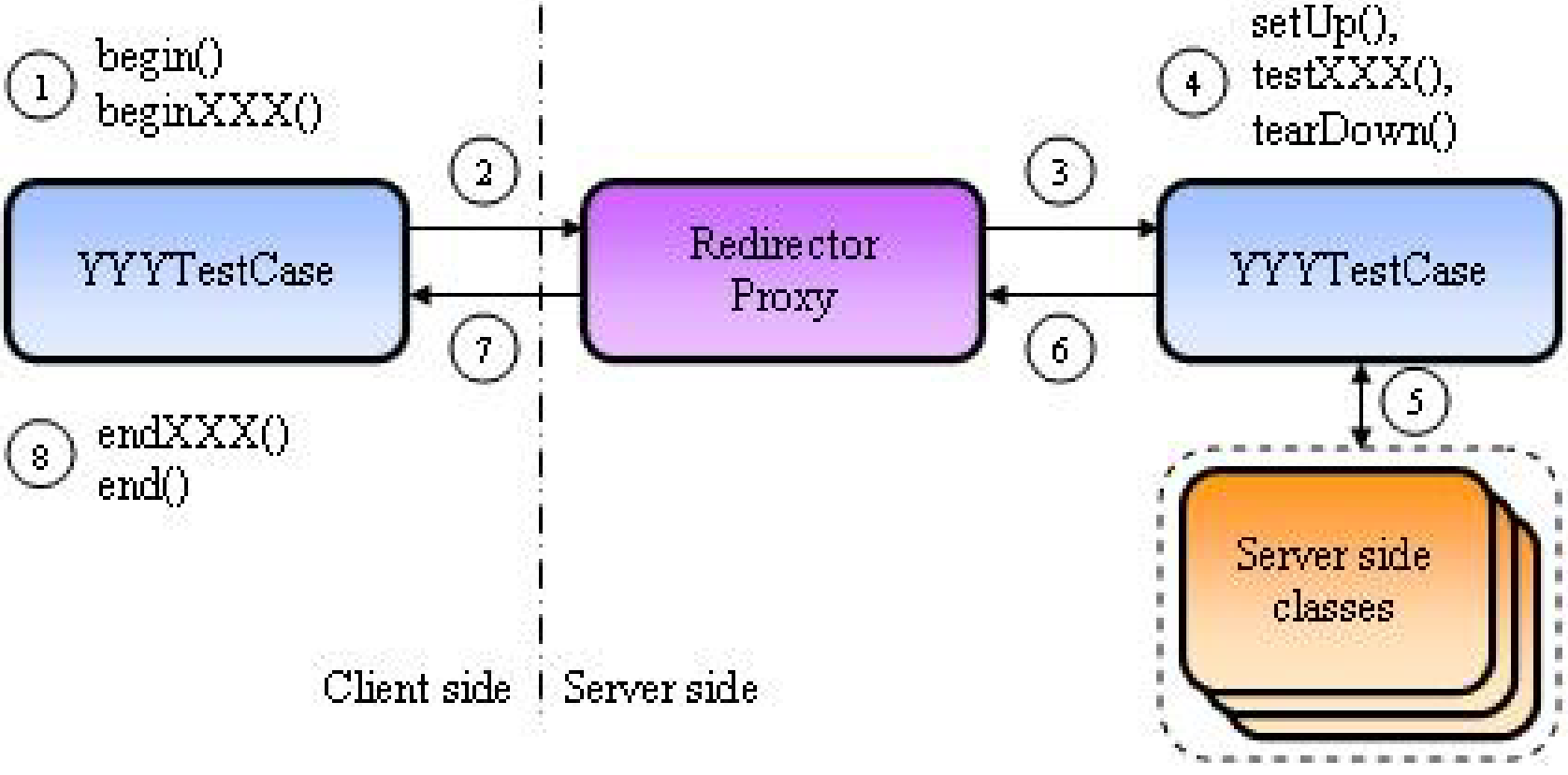
Logfiles

Capture & Replay

Cactus

- Testet sowohl Client als auch auf dem Server
- geeignet für Integrationstests aber auch für Unit-Tests
- Erweiterung von JUnit, so dass bisheriger JUnit Code ohne Probleme in Cactus integriert werden kann

Cactus


1
begin()
beginXXX()
2
YYYTestCase
Redirector
Proxy
3
4
setUp(),
testXXX(),
tearDown()
YYYTestCase
5
Server side
classes
6
7
8
endXXX()
end()
Client side
Server side

3. Werkzeuge

HTTPUnit

- Ergänzung zu JUnit
- OpenSource Software
- stellt einen in Java programmierbaren Web-Client zur Verfügung
- Simulation eines Web-Browsers
- geeignet zur „Fernsteuerung“ von Web-Applikationen
  aber auch natürlich zum Testen
- geeignet für:
  + funktionale retrospektive Tests
  - Test-First-Ansatz

Beispiel: getResponse

Spiegel Quellcode

Spiegel Test

```
public void testHomePage() throws Exception
{
        // Startet WebConversation
        wc = new WebConversation();

        // Fordert die Homepage der Adresse  an
        WebResponse homePage = wc.getResponse("http://www.spiegel.de");

        assertEquals(200, homePage.getResponseCode());

        String[] copyrightContentValue = homePage.getMetaTagContent("name", "copyright");
        assertTrue(copyrightContentValue.length == 1);
        assertEquals("SPIEGEL ONLINE, Hamburg, Germany ", copyrightContentValue[0]);

        ...
}
```

Beispiel: getResponse (2)

```
WebLink TestLink = homePage.getLinkWith(„TEST");
assertNotNull(TestLink);

WebResponse TestPage = TestLink.click();
assertEquals(200, TestPage.getResponseCode());
```

HTTPUnit

- Ebenso geeignet für folgende Funktionen:
  - Verfolgung von Links
  - Testen von Frames
  - Testen von Cookies
  - Testen von SSL-Verschlüsselung
  - Ausfüllen von Webformularen

3. Werkzeuge

Beispiel: getForm

```
public void testChangeName() throws Exception
{
        // Startet WebConversation
        Webconversation con = new WebConversation();
        WebRequest request = new GetMethodWebRequest(URL);

        //ruft die Seite auf und fragt Formular ab
        WebResponse response = con.getResponse(request);
        WebForm form = response.getFormWithName(„form1“);
        request = form.getRequest(„button“);

        // setzt Parameter name auf „Fabian“ und ruft neue Antwort ab
        request.setParameter(„name“, „Fabian“);
        response = con.getResponse(request);

        // vergleicht Wert mit den Vorgaben
        assertTrue(response.getText().indexOf(„Hello, Fabian !“) != -1);
}
```

HTTPUnit

Grenzen von HTTPUnit

- benutzt „Rhino“ Engine und kann deshalb unterschiedliche Ergebnisse als andere Browser (IE, Netscape) liefern
- deshalb auch nicht geeignet, um Kompatibilität der Anwendung bezüglich unterschiedlichen Browsern zu testen
- unterstützt nur Antworten vom Typ „text/html“
- Nicht geeignet zum Testen von Browser-Skriptsprachen wie JavaScript.

Capture & Replay

- Simulation des Anwenderverhaltens bzgl.
  - Mausclicks
  - Mausbewegungen
  - Tastatureingaben
- praktisch bei Websites, die häufig aktualisiert und angepasst werden
- keine Programmierkentnisse erforderlich -> Einsatzgebiet auch beim Kunden

3. Werkzeuge

Logfiles

liefert eine Übersicht über (je nach Programmierung und Einstellungen)

- die verwendeten Browser
- Benutzerdaten
- aufgetretene Fehler
- Lastprofile

Nützlich in der Produktionsphase, um unentdeckte Fehler zu beheben

3. Werkzeuge

Beispiel: Logfiles

```
[Wed Jun 02 19:42:05 2004] [error] [client 192.168.0.2] File does not exist: /opt/lampp/htdocs/cms
[Wed Jun 02 19:42:08 2004] [error] [client 192.168.0.2] File does not exist: /opt/lampp/htdocs/cms
[Wed Jun 02 19:44:00 2004] [notice] caught SIGTERM, shutting down
[Wed Jun 02 19:44:13 2004] [notice] Apache/2.0.48 (Unix) mod_perl/1.99_08 Perl/v5.8.0 mod_ssl/2.0.48
OpenSSL/0.9.7c PHP/4.3.3 DAV/2 mod_jk/1.2.5 configured -- resuming normal operations
[Wed Jun 02 19:46:42 2004] [notice] caught SIGTERM, shutting down
[Wed Jun 02 21:15:05 2004] [notice] suEXEC mechanism enabled (wrapper: /opt/lampp/bin/suexec)
[Wed Jun 02 21:15:06 2004] [notice] Digest: generating secret for digest authentication ...
[Wed Jun 02 21:15:06 2004] [notice] Digest: done
[Wed Jun 02 21:15:07 2004] [notice] Apache/2.0.48 (Unix) mod_perl/1.99_08 Perl/v5.8.0 mod_ssl/2.0.48
OpenSSL/0.9.7c PHP/4.3.3 DAV/2 mod_jk/1.2.5 configured -- resuming normal operations
[Wed Jun 02 23:18:58 2004] [error] [client 192.168.0.2] File does not exist: /opt/lampp/htdocs/example
```

Agenda

1. Motivation und Unterschiede
2. Methoden
3. Werkzeuge
4. **Fazit**

4. Fazit

- Web-Test:

  → Summe der Qualitätsmerkmale ist ausschlaggebend für den Erfolg einer Anwendung

- Klassische Softwareentwicklung:

  → oft nur einzelne Qualitätsmerkmale von Bedeutung

4. Fazit

- Bedeutung von Wiederverwendbarkeit und Änderbarkeit von Tests steigt

→ Testautomatisierung gewinnt an Bedeutung

4. Fazit

- Vielzahl zu berücksichtigender Qualitätsmerkmale
- sehr hohe Anzahl von Kombinationen von Hard- und Software-Konfigurationen der Clients
- sehr kurze Entwicklungszyklen:

→ Unmöglich, eine auch nur annähernd vollständige Testabdeckung zu erreichen!

4. Fazit

- Trend:

„risikobasiertes Testens“

→ Teile, bei denen die Konsequenzen eines übersehenen Fehlers am schwerwiegendsten sind, werden zuerst und mit höherem Aufwand getestet

Literatur

- G. Kappel, S.Reich: *Web Engineering: Systematische Entwicklung von We-Anwendungen*, dpunkt.verlag 2004
- J. Link: *Unit Tests mit Java*, dpunkt.verlag, 2002
- www.devx.com
- www.dev2dev.com
- www.jakarta.apache.org/cactus/