XCEPTANCE
Software Technologies

Software-Qualitätssicherung, Teil III: Testautomatisierung

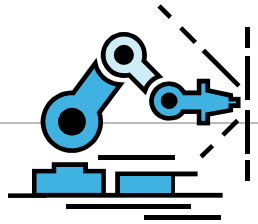
Friedrich-Schiller-Universität Jena, Wintersemester 2017/2018

Ronny Vogel, Xceptance GmbH

Agenda

- Testautomatisierung – Grundlagen
- Testautomatisierung – Ebenen
- Grenzen der Testautomatisierung
- TDD, BDD, ATDD

Testautomatisierung – Begriff

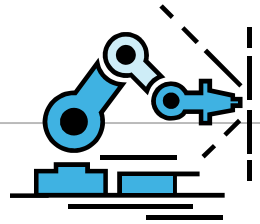


Was ist Testautomatisierung?

- Automatisierung von Aktivitäten im Test mit geeigneten Werkzeugen, z.B.:
  - Testdatengenerierung
  - automatisierte Testausführung nach jedem Build
  - Erzeugen von Testdokumentation mit Dokumentgeneratoren
  - automatisierte Testfallerstellung (Modellbasiertes Testen)
  - Lasttests

- Begriff meist jedoch im engeren Sinne gebraucht, so auch im Weiteren
  ➔ automatisierte Ausführung funktionaler Tests

Testautomatisierung – Ziele

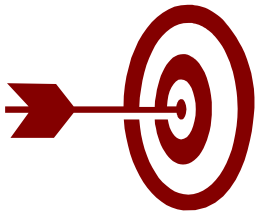


Höhere Qualität durch

- häufigere Ausführung umfassender Testsuiten
- umfassenden Test auch bei kleinen Änderungen oder kleinen Releases
- Testausführung mit einer Anzahl verschiedener Konfigurationen
- exakt reproduzierbare Wiederholung von Testschritten
- verringertes Restrisiko

Zeitersparnis

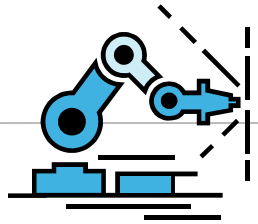
- schnellere Testdurchführung, früheres Feedback
- destabilisierende Änderungen und Regression schneller erkennen
- Voraussetzung für eine effiziente QS in agilen Vorgehensmodellen!



Kostenreduzierung

- weniger wiederkehrender manueller Testaufwand

Testautomatisierung – Vorteile (1)



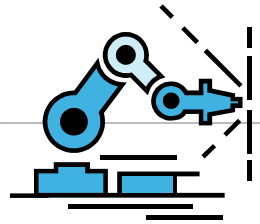
Befreit Tester von monotonen, wiederholten Tätigkeiten

- mehr Zeit für interessantere, wichtigere Aufgaben
- Konzentration auf anspruchsvollere Testszenarien, exploratives Testen
- höherer kreativer Anteil ➔ Motivation
- mehr Zeit für tiefere Testabdeckung

Automatische Ausführung

- unbeaufsichtigte Testausführung möglich
- unabhängig von der Verfügbarkeit manueller Tester
- in Continous Integration einbindbar
- Ergebnisse liegen schneller vor
- weniger fehleranfällig
- automatische Protokollierung

Testautomatisierung – Vorteile (2)



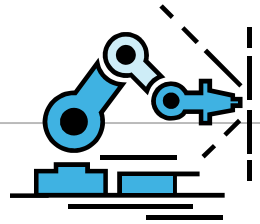
Schnell und häufig wiederholbar

- verkürzt Entwicklungs- und Releasezyklen
- erhöht das Vertrauen in die gelieferte Qualität
- „Sicherheitsnetz“ bei Refactoring und anderen Änderungen

Höhere Testabdeckung in wiederholten Tests

- kontinuierlicher Ausbau der Testfallanzahl bei nur minimal steigendem Ausführungsaufwand
- Prüfung vieler Varianten möglich (datengetriebene Tests)
- Fälle abdeckbar, die manuell nicht effizient getestet werden können
- Beispiele:
  - Prüfzifferberechnung von Kontonummern mit etwa 100 Verfahren
  - Crawling über eine größere Website zur Sicherstellung der Datenqualität; Alle Links funktional? Alle Bilder vorhanden? Grundelemente auf jeder Seite vorhanden (AGB, Impressum)?, …

Testautomatisierung – Vorteile (3)



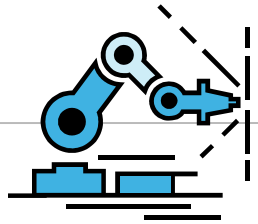
Bessere APIs, besserer Code

- Testautomatisierung ist oft die erste Nutzung von APIs
  ➔ Feedback zur API-Verbesserung
- bei Test-Driven Development (TDD) wird Testcode noch vor dem eigentlichen Code geschrieben
  ➔ Mängel schon bei Implementierung sicht- und behebbar
- Testautomatisierung auf UI-Ebene erzwingt oft Änderungen für Testbarkeit
  ➔ z.B. besserer HTML- und CSS-Stil bei Web-Anwendungen

Bessere Sichtbarkeit der Ergebnisse

- automatische Bereitstellung der Ergebnisse, z.B. auf Webseiten
- automatische Erstellung von Trendreports

Testautomatisierung – Nachteile



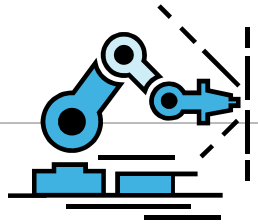
Aufwand

- hoher Initialaufwand
- permanente Pflege nötig
- Testcode ist Code
  - enthält Defekte
  - erfordert auch Review und Test
  - erfordert gelegentliches Refactoring
- Schulungsbedarf

Sonstiges

- teilweise komplexe Technologien und Werkzeuge
- Erstellung wartbarer, robuster automatisierter Testfälle ist anspruchsvoll
- erweitertes Wissen und Erfahrung erforderlich
- teilweise hohe Lizenzkosten

Testautomatisierung – Hinweise (1)

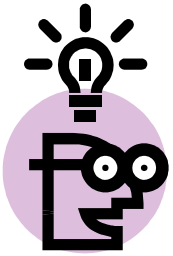


Keep it simple!

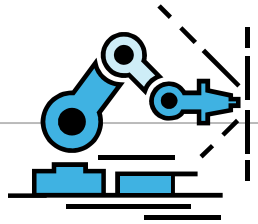
- mit einfachen, geschäftskritischen Fällen beginnen
- zunächst Erfahrungen sammeln, dann vom Einfachen zum Komplexen
- Testfälle klar, verständlich, leicht ausführbar gestalten
- komplexe Testsuite-Architekturen vermeiden
- nicht 100% Automatisierung versuchen; nur wo sinnvoll

Testfälle unabhängig voneinander gestalten!

- benötigte Vorbedingungen durch Testfall selbst oder in Setup-Methoden anlegen
- Ausführung in beliebiger Reihenfolge ermöglichen
- Fehlschlagen eines Testfalls darf keine Auswirkung auf andere haben
- wenn nötig, durch Tear-Down-/Clean-Up-Methoden Ausgangszustand wiederherstellen

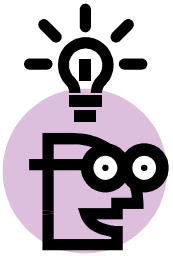


Testautomatisierung – Hinweise (2)



Allgemeine Hinweise

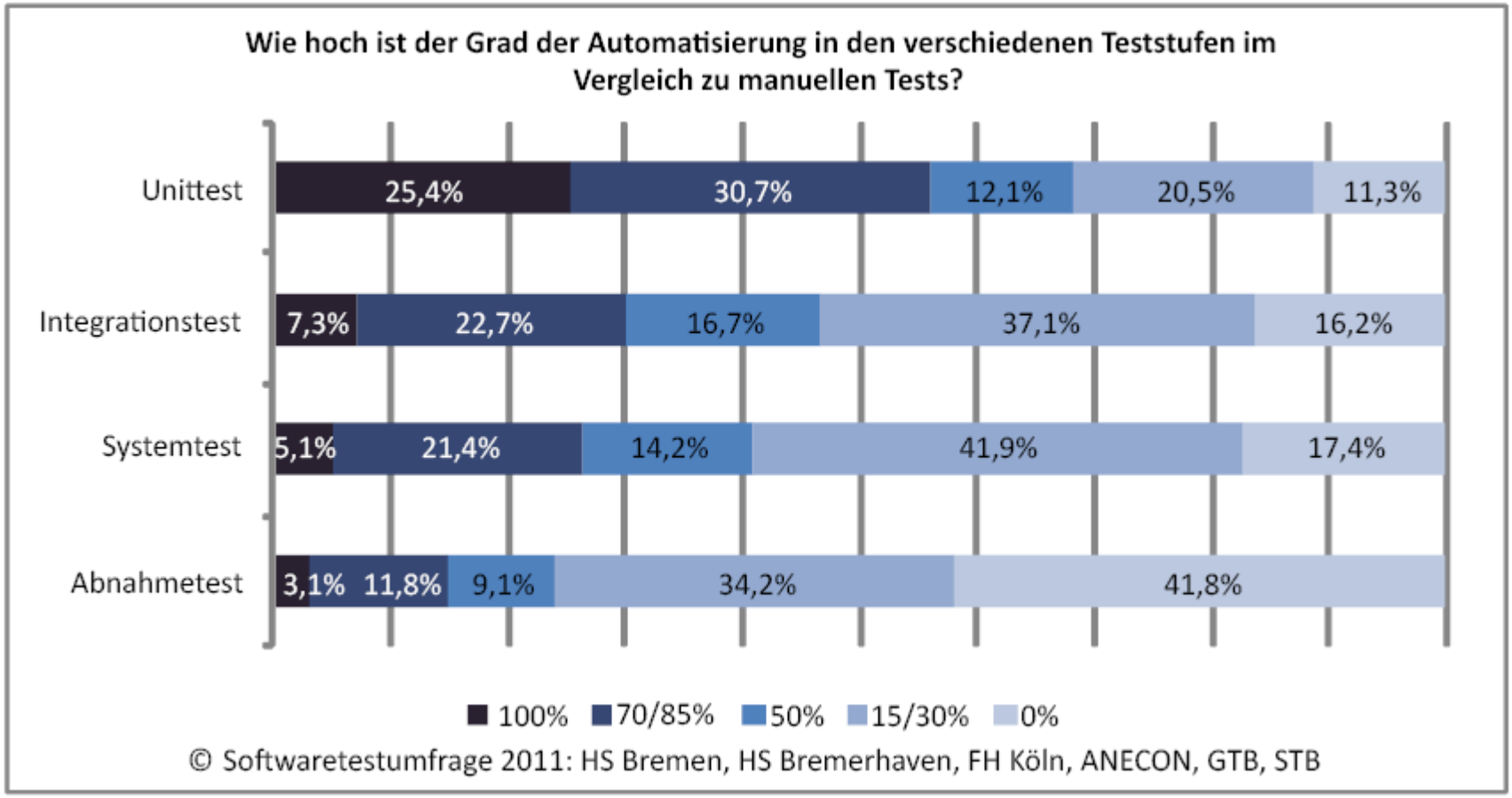
- vor Automatisierung von Testfällen immer manueller Test nötig/sinnvoll
- Testfälle möglichst atomar
- Testdaten auslagern, nicht hart codieren
- auf Robustheit der Testfälle achten
- Testfälle mit Produktcode versionieren
- auf möglichst geringe Laufzeit achten
- schnelle und langsame Testfälle nicht mischen, eventuell separate Testsuiten
- permanente Pflege und Erweiterung einplanen



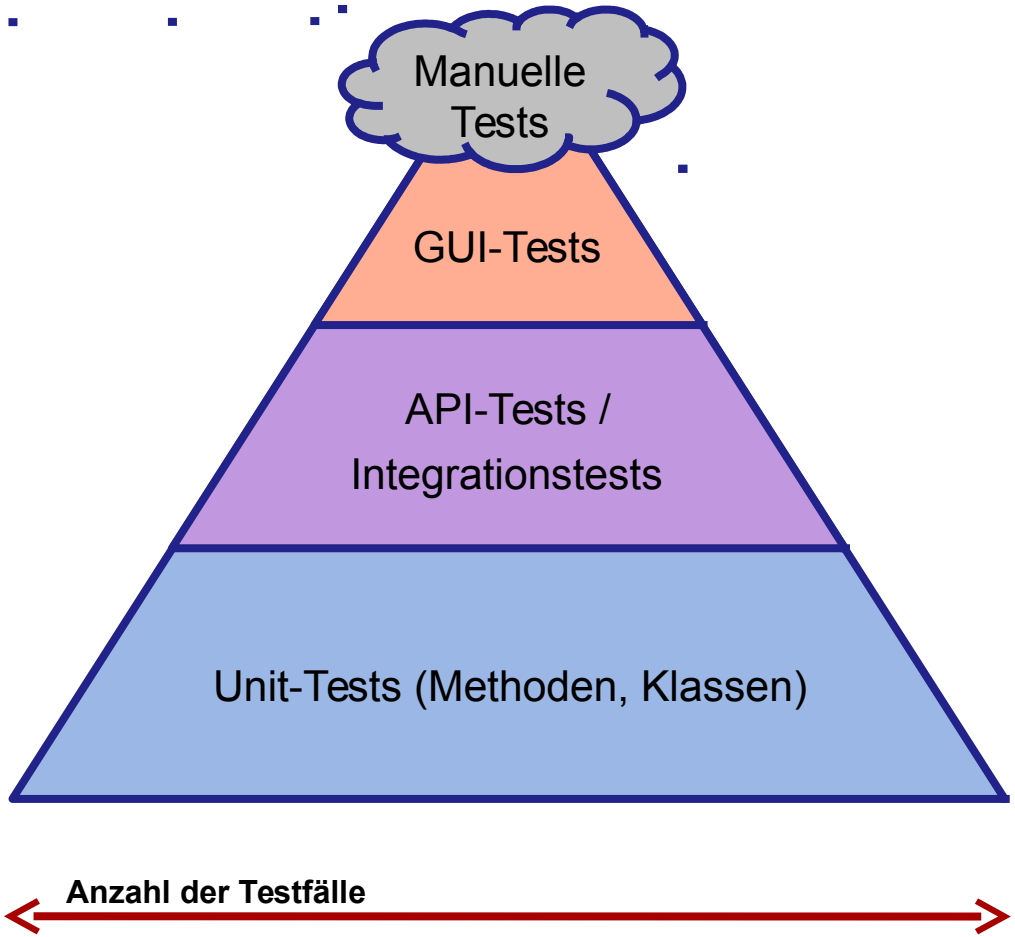
Agenda

- Testautomatisierung – Grundlagen
- Testautomatisierung – Ebenen
- Grenzen der Testautomatisierung
- TDD, BDD, ATDD

Ebenen der Testautomatisierung (1)

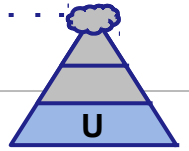


Ebenen der Testautomatisierung (2)



Testpyramide nach Mike Cohn:
"Succeeding with Agile", 2009

Ebene Unit-Tests (1)



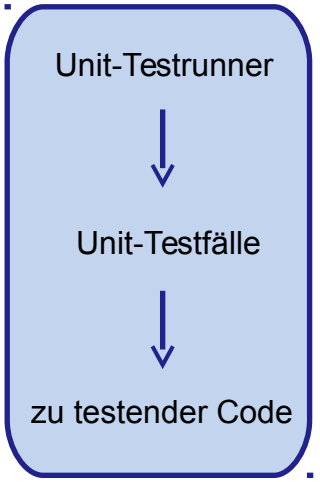
Erstellung (durch Entwickler selbst)

- je Testfall wird eine Testmethode in der jeweiligen Programmiersprache erstellt
- Testmethoden werden gekennzeichnet, z.B. mittels Annotationen:

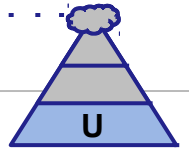
```
@Test
public void searchCustomersWithMultipleWildcards()
{…
     // Aufruf des zu testenden Codes
     …
     // Prüfung der Ergebnisse mit assert(…)-Konstrukten
     assert(…);
}
```

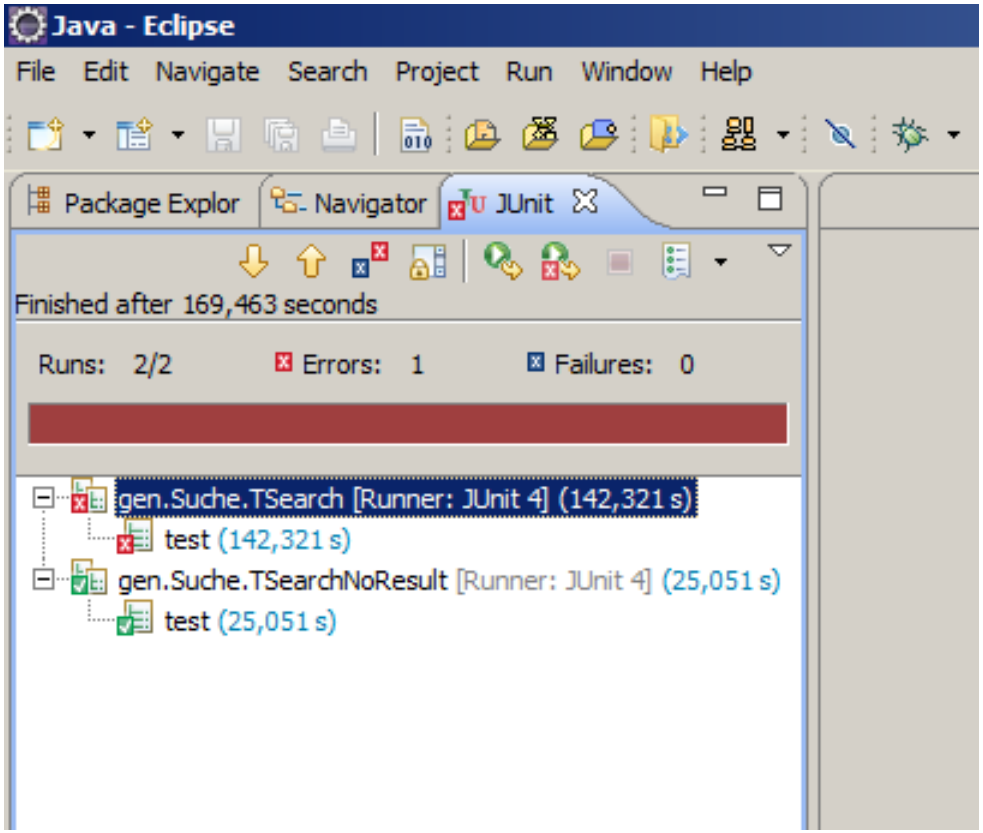
Ausführung

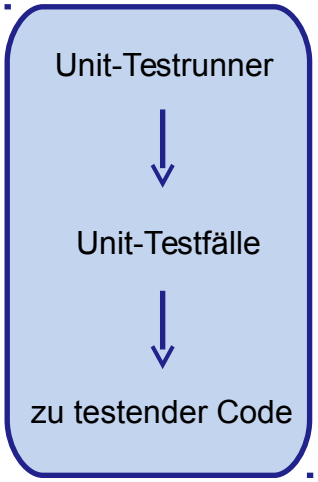
- trifft eine per assert() definierte Annahme nicht zu, wird ein Fehler über eine Exception an den Testrunner kommuniziert
- ein Unit-Testrunner erkennt die Testmethoden, führt sie aus und erstellt einen Ergebnisreport
- Testrunner als GUI-Anwendung oder kommandozeilenbasiert



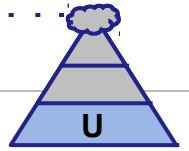
Ebene Unit-Tests (2)





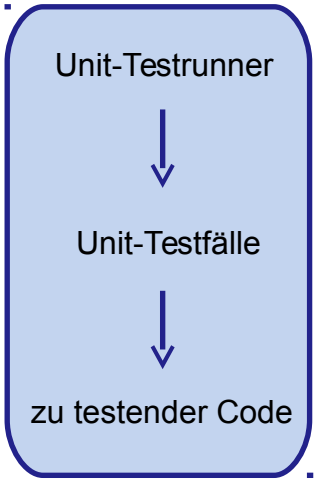


Ebene Unit-Tests (3)

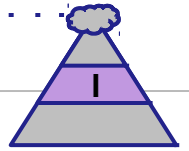


Merkmale

- testen Methoden/Funktionen der Anwendung (Code-Ebene)
- Fokus ist technische Korrektheit der Implementierung
- sollen sehr schnell und lokal begrenzt arbeiten, daher oft
  - ohne Datenbank-Kommunikation
  - ohne Netzwerkverbindungen
  - ohne Arbeit mit dem Dateisystem
  - …
- Nachbildung benötigter Komponenten durch „Test Doubles“
  ➔ „Mocking“-Frameworks
- bei Code-Änderung sofort auch Änderung des Testcodes
- führt Entwickler bereits vor Commit in zentrales Repository aus
- tragen massiv zur Verbesserung der Codequalität bei
- hohe Anzahl, oft zehntausende



Ebene Integrations- und API-Tests (1)



Tests für lokales API

- testen Zusammenspiel von Komponenten; Geschäftsfunktionen
- testen technische Korrektheit und korrekte Umsetzung von Anforderungen
- technologisch auch als Unit-Tests oder ähnlich realisierbar
- ebenfalls lokale Aufrufe von Code im gleichen Prozess
- Tests komplexer, höhere Funktionalität

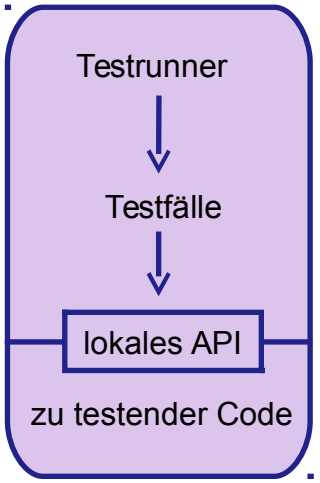
Vorteile:

- testen im Idealfall ein wohldefiniertes API
- über den Wartungszeitraum der Software wesentlich stabiler
- besser dokumentiert

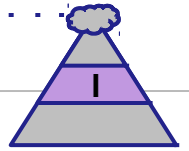
Nachteil:

Nur die Funktionen sind testbar…

- ...die im API zur Verfügung gestellt werden
- ...für die Setup und Ergebnisprüfung über das API möglich sind

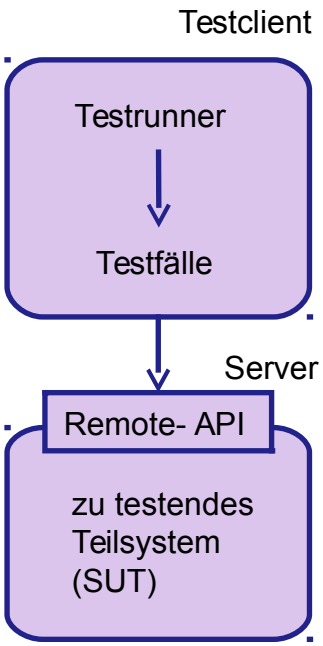


Ebene Integrations- und API-Tests (2)



Tests für Remote-API

- analog zum Test über lokales API, aber getrennte Prozesse
- synchrones API oder asynchrone Schnittstelle
- Testclient kommuniziert über bestimmte Protokolle mit SUT
- unterschiedliche Client-Technologien und Programmiersprachen von Testcode und SUT möglich
- Beispiel:
  - Remote Services über HTTP, REST-API
  - Messaging-Schnittstelle, z.B. Websphere MQ



Vorteile:

- wie bei lokalem API

Nachteile:

- wie bei lokalem API
- Testumgebung und Testfälle komplexer, oft langsamer

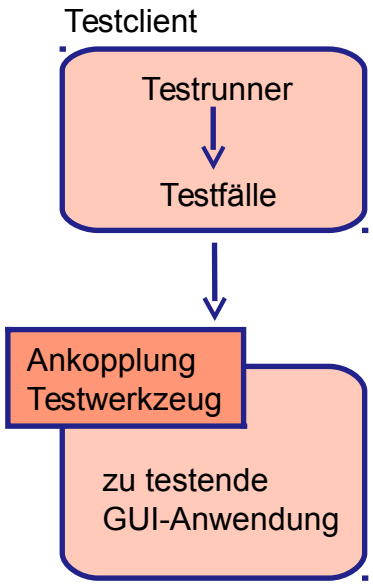
Ebene GUI-Tests, lokal

GUI-Tests für lokale Anwendungen („Fat Clients“)

- Test der Anwendung durch das GUI (nicht Test des GUI allein)
- Anwendersicht auf ein möglichst vollständiges Gesamtsystem
- Werkzeug koppelt sich an Anwendung oder Betriebssystem; versucht, GUI-Objekte und deren Manipulation zu erkennen, aufzuzeichnen und später nachzubilden

Varianten

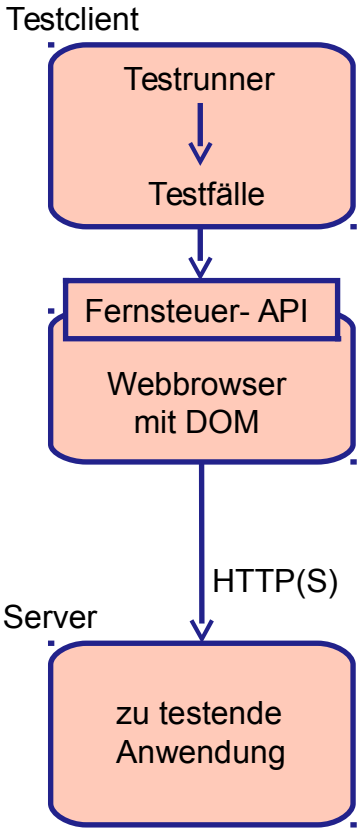
- Maus- und Tastaturaktionen koordinatenbasiert aufzeichnen und beim Testlauf wieder nachbilden
  ➔ schwierig, unflexibel, fehleranfällig, wenig robust
- Oberflächenelemente und Ereignisse vom darunter liegenden Fenstersystem erfassen
  ➔ schwierig, oft fehleranfällig
- über ein in das GUI integriertes Fernsteuerungs-API
  ➔ technisch ideal, aber Funktionalität oft eingeschränkt



Ebene GUI-Tests, Web-Anwendungen

GUI-Tests für Web-Anwendungen (browserbasiert)

- Sonderform des automatisiertenTests auf GUI-Ebene
- Webbrowser heute oft als universeller „Thin-Client“ genutzt
- durch Browser-Features oder Browser-Erweiterungen relativ gut umsetzbar
- arbeitet oft über DOM-/HTML-Zugriffe: Werkzeug erkennt und verändert DOM-Elemente im Browser (*DOM = Document Object Model*, standardisierte Schnittstelle mit Zugriffsmethoden auf die HTML-Struktur)
- viele existierende Werkzeuge, gut unterstützt
- Vorteil: Technologien und Erfahrungen mit Browser-Automatisierung breit verfügbar
- nutzt ferngesteuerten echten oder simulierten Webbrowser
- Beispiel: Selenium WebDriver als Fernsteuer-API steuert z.B. Firefox oder HtmlUnit („Headless Browser“ ohne GUI)



Agenda

- Testautomatisierung – Grundlagen
- Testautomatisierung – Ebenen
- Grenzen der Testautomatisierung
- TDD, BDD, ATDD

Grenzen der Testautomatisierung (1)

Aufwand

- Automatisierung auf höheren Ebenen komplex und daher oft teuer
- anfällig gegenüber Änderungen an der zu testenden Anwendung
  ➔ ohne permanente Pflege sterben Testfälle aus
- Aufwand versus Nutzen bei Erstellung und Pflege kritisch
  ➔ Testumfang gut auswählen und gegebenenfalls nachjustieren

Fehlende Intelligenz

- vergleichsweise schmalbandige Validierung der Ergebnisse
- prüft weniger Eigenschaften, als ein Tester; „Blick zur Seite“ fehlt
- erkennt keine GUI- und Layout-Fehler
- entwickelt sich nicht selbst weiter
- kann Abweichungen vom Soll-Ergebnis nicht bewerten (Testorakel-Problem)
- findet keine Fehler außerhalb vordefinierter Szenarien
- James Bach: „ein automatisierter Test testet nicht, er prüft nur“

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie Fragen?

Kontakt

Xceptance Software Technologies GmbH
Leutragraben 2-4
07743 Jena

Tel.: +49 (0) 3641 55944-0
Fax: +49 (0) 3641 376122

E-Mail: kontakt@xceptance.de
http://www.xceptance.de

Copyrights