Testen, was, warum, wie, und alles was etwas damit zu tun hat.

Naja, vieleicht nicht alles....

Johannes Schneider

26. April 2015

Agenda

- Einführung
- Verscheiden Teststrategien
- Wie? Ich soll alles testen?
- Mocken, Fixtures, ... was ist das eigentlich?
- Beispiele und Probleme

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)
- 80er: Testen mit dem Ziel Fehler zu finden.
  Rethinking Systems Analysis and Design - Jerry Weinberg (1988)

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)
- 80er: Testen mit dem Ziel Fehler zu finden.
  Rethinking Systems Analysis and Design - Jerry Weinberg (1988)
- frühen 90er: Verbreitung von Bugtracking Systemen und Software Versions Kontrol Systemem steigt.

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)
- 80er: Testen mit dem Ziel Fehler zu finden.
  Rethinking Systems Analysis and Design - Jerry Weinberg (1988)
- frühen 90er: Verbreitung von Bugtracking Systemen und Software Versions Kontrol Systemem steigt.
- seit den späten 90er: Aufkommen von Test Driven Development und Agiler Entwicklung

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)
- 80er: Testen mit dem Ziel Fehler zu finden.
  Rethinking Systems Analysis and Design - Jerry Weinberg (1988)
- frühen 90er: Verbreitung von Bugtracking Systemen und Software Versions Kontrol Systemem steigt.
- seit den späten 90er: Aufkommen von Test Driven Development und Agiler Entwicklung
- seit späten 00er: Oberflächen tests, Komplexe Hilstechnologien, Testing Frameworks usw.

Einführung

Übersicht und Entwicklung

Aus *The Growth of Software Testing - D. Gelperin, B. Hetze / History of Ideas in Software Testing* (`www.softwaretestpro.com`):

- Bis 1956: Debug orientiert. Also keine Unterscheidung zwischen Debuging und Testing.
- 60er/70er: Trennung von Testing und Debugging. Theoretische Korrektheitsbeweise:
  Equivalence Classes, Boundaries, Error Guessing, Cause/Effect Graphing (It's history, good and bad) - Functional to Unit Testing (1970)
- 80er: Testen mit dem Ziel Fehler zu finden.
  Rethinking Systems Analysis and Design - Jerry Weinberg (1988)
- frühen 90er: Verbreitung von Bugtracking Systemen und Software Versions Kontrol Systemem steigt.
- seit den späten 90er: Aufkommen von Test Driven Development und Agiler Entwicklung
- seit späten 00er: Oberflächen tests, Komplexe Hilstechnologien, Testing Frameworks usw.

Einführung

Test Philosophien

- Der (End-)User testet.

Einführung

Test Philosophien

- Der (End-)User testet.
- Nur Tests vor Releases.

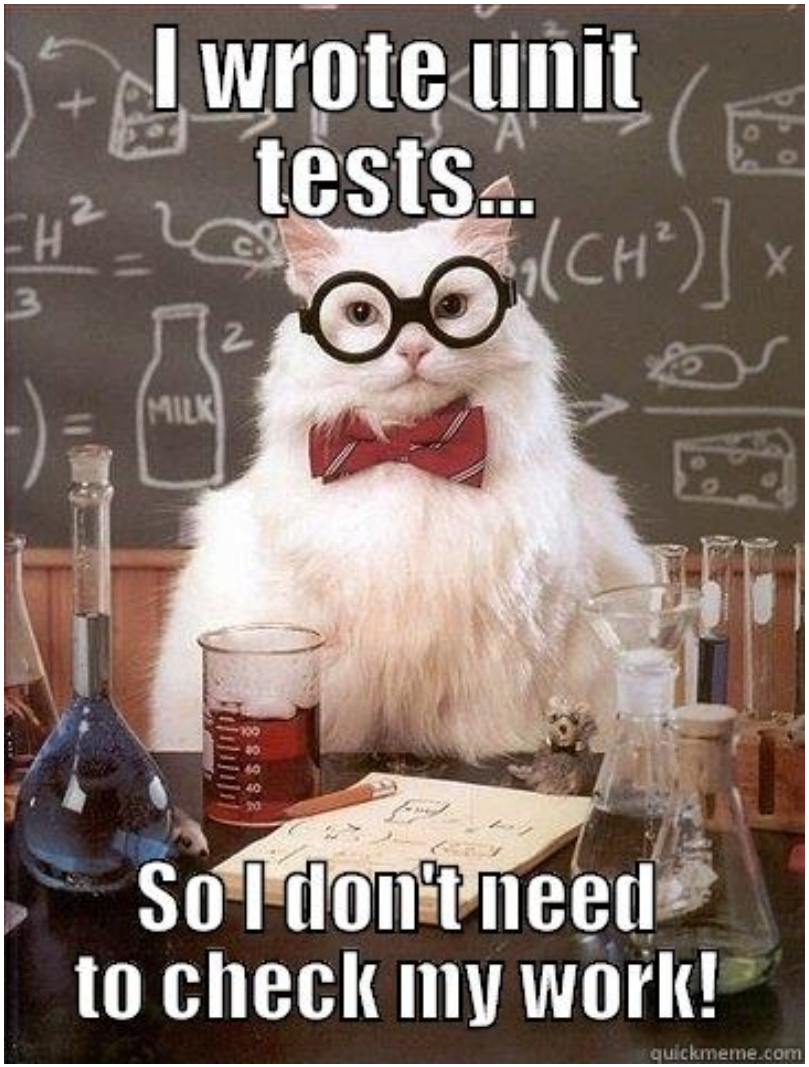
Einführung

Test Philosophien

- Der (End-)User testet.
- Nur Tests vor Releases.
- Tests sind grundlegender Bestandteil des Entwicklungsprozesses. (Test Driven Development, Behavior Driven Development)

Einführung

Was kann man von Tests erwarten?



Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests
- Infrastruktur Tests.

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests
- Infrastruktur Tests.
- System Tests

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests
- Infrastruktur Tests.
- System Tests

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests
- Infrastruktur Tests.
- System Tests

Test Ansätze:

- Blackbox Testen: Testen mit Kenntnis der Implementierung

Teststrategien

Überblick

Welche Strategien gibt es zum Testen?

- Unit Tests
- Integrations Tests
- Oberflächen Tests
- Infrastruktur Tests.
- System Tests

Test Ansätze:

- Blackbox Testen: Testen mit Kenntnis der Implementierung
- Whitebox Testen: Testen ohne Kenntnis der Implementierung

‘

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

- Rechenleistung

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

- Rechenleistung
- Speicherinfrastrutkur

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

- Rechenleistung
- Speicherinfrastrutkur
- Netzwerkinfrastruktur

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

- Rechenleistung
- Speicherinfrastrutkur
- Netzwerkinfrastruktur
- Hardware

Teststrategien

Infrastruktur Tests

Testen von Verfügbarkeiten, Failover, usw.

Testbare Komponenten

- Rechenleistung
- Speicherinfrastrutkur
- Netzwerkinfrastruktur
- Hardware

Keine Ahnung wie dies automatisiert werden kann.
Weder Black- noch Whitebox Testen.

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Man benötigt keine Entwickler :)

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Man benötigt keine Entwickler :)

Was ist testbar?

- Existens von GUI Elementen

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Man benötigt keine Entwickler :)

Was ist testbar?

- Existens von GUI Elementen
- Korrektes Verhalten der GUI Elemente

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Man benötigt keine Entwickler :)

Was ist testbar?

- Existens von GUI Elementen
- Korrektes Verhalten der GUI Elemente
- Korrekter Ablauf des Workflows (aus Anwendersicht)

Teststrategien

Oberflächen Tests I

Whitebox Test von Programen über die Oberfläche.

Grosser Vorteil:

Man benötigt keine Entwickler :)

Was ist testbar?

- Existens von GUI Elementen
- Korrektes Verhalten der GUI Elemente
- Korrekter Ablauf des Workflows (aus Anwendersicht)
- Korrektes Verhalten in verschiedenen Browsern

Teststrategien

Oberflächen Tests II

Was ist nicht testbar

Teststrategien

Oberflächen Tests II

Was ist nicht testbar

- Nicht direkt für den Benutzer sichtbares Verhalten (z.B. Mail Versand)

Teststrategien

Oberflächen Tests II

Was ist nicht testbar

- Nicht direkt für den Benutzer sichtbares Verhalten (z.B. Mail Versand)
- Anforderungen, welche keine Interaktion erfordern.

Teststrategien

Oberflächen Tests II

Was ist nicht testbar

- Nicht direkt für den Benutzer sichtbares Verhalten (z.B. Mail Versand)
- Anforderungen, welche keine Interaktion erfordern.
- Sicher noch mehr.

Teststrategien

Oberflächen Tests II

Was ist nicht testbar

- Nicht direkt für den Benutzer sichtbares Verhalten (z.B. Mail Versand)
- Anforderungen, welche keine Interaktion erfordern.
- Sicher noch mehr.

Frameworks

- Selenium(Webseitentests, Bindings u.a. für Java, Python, C#, PHP, Ruby)
- Windmill

Teststrategien

System Tests

Whitebox Test gegen die gesamten Anforderungen.

Teststrategien

System Tests

Whitebox Test gegen die gesamten Anforderungen.

Was genau ist das?

- Vereinen Oberflächen-, Integrations, sowie Infrastruktur Tests

Teststrategien

System Tests

Whitebox Test gegen die gesamten Anforderungen.

Was genau ist das?

- Vereinen Oberflächen-, Integrations, sowie Infrastruktur Tests
- Darum kein einheintliches Framework.

Teststrategien

System Tests

Whitebox Test gegen die gesamten Anforderungen.

Was genau ist das?

- Vereinen Oberflächen-, Integrations, sowie Infrastruktur Tests
- Darum kein einheintliches Framework.
- Testumgebung sollte der Produktivumgebung sehr ähnlich sein.

Teststrategien

System Tests

Whitebox Test gegen die gesamten Anforderungen.

Was genau ist das?

- Vereinen Oberflächen-, Integrations, sowie Infrastruktur Tests
- Darum kein einheintliches Framework.
- Testumgebung sollte der Produktivumgebung sehr ähnlich sein.
- Oftmals zusätzlich zu automatisierten Tests.

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Was ist nicht testbar?

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Was ist nicht testbar?

- Oberflächen ($\rightarrow$ Oberflächentests)

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Was ist nicht testbar?

- Oberflächen ($\rightarrow$ Oberflächentests)
- Infrastruktur ($\rightarrow$ Infrastrukturtests)

Teststrategien

Integrations Tests I

Whitebox Test gegen die API Anforderungen oder von komplexer Funktionalität.

Was ist testbar?

- (öffentliche) APIs
- Business Logik

Was ist nicht testbar?

- Oberflächen ($\rightarrow$ Oberflächentests)
- Infrastruktur ($\rightarrow$ Infrastrukturtests)
- Konkrete Implementierungsdetails ($\rightarrow$ Unittests)

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik
- Große Hilfe beim Refakturieren

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik
- Große Hilfe beim Refakturieren

Frameworks

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik
- Große Hilfe beim Refakturieren

Frameworks

- Abhangig vom Einsatzgebiet und Programmiersprache

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik
- Große Hilfe beim Refakturieren

Frameworks

- Abhangig vom Einsatzgebiet und Programmiersprache
- Erlauben automatisierung

Teststrategien

Integrations Tests II

Also, was macht ein Integrations Test?

- Testet, dass implementiert wurde, was vereinbart wurde.
- Testet konkretes Verhalten
- Dokumentiert Business Logik
- Große Hilfe beim Refakturieren

Frameworks

- Abhangig vom Einsatzgebiet und Programmiersprache
- Erlauben automatisierung
- Beispiele[a]: PyTest, EUnit, JUnit, Opmock(C/C++), Jasemine (JavaScript)

[a] `en.wikipedia.org/wiki/List_of_unit_testing_frameworks`