Softwaretests

Einführung[1]

Um Fehler in Softwaresystemen frühzeitig zu erkennen, werden Softwaretests in der Softwareentwicklung auf verschiedenste Art und Weise eingesetzt. Softwaretests als Teil der Qualitätssicherung haben das Ziel definierte Anforderungen zu überprüfen und die Qualität zu messen. Weiter werden Softwaretests eingesetzt um die Qualität des Quellcodes zu erhöhen. Gut testbarer Code erfordert Modularisierung, was zu einer besseren Strukturierung führt.

Softwaretests gehören zu den testenden Verfahren im analytischen Teil des Qualitätsmanagements. Viele andere Massnahmen und Verfahren können zusammen eingesetzt werden um die Qualität eines Softwaresystems sicherzustellen. Es wird zwischen White-Box-und Black-Box-Tests unterschieden. White-Box-Tests nutzen die Kenntnisse über die innere Funktionsweise eines Systems, während Black-Box-Tests funktionsorientiert testen. Sie werden zum Beispiel geschrieben um Anforderungen zu überprüfen. Weiter wird zwischen automatisiertem Testen und manuellem Testen unterschieden. Wobei der zeitliche Aufwand der Hauptunterschied ist.

Arten von Softwaretests[2]

Testart	**Was wird getestet?**
Unit Tests	Einzelne Module
Akzeptanztests	Erfüllung der Anforderungen
Usability-Test	Workflows
Regressionstest	Altes Verhalten vs. neues Verhalten
Performancetest	Ressourcenverbrauch
Integrationtest	Zusammenarbeit verschiedener Module
Systemtest	Gesamtsystem
GUI-Tests	Verhalten des GUI

[1] Vorlesung: CS108 Programmierprojekt, Software Qualitätssicherung
[2] Vorlesung: Software Engineering, Franz-Josef Elmer

„Seven Principles of Software Testing“[3]

Principle 1: Definition
Beim Testen geht es darum Fehler zu provozieren/finden, nicht diese zu beheben.
Principle 2: Tests versus specs
Spezifikationen abstrahieren und generalisieren das Problem, wobei Tests jeweils individuelle Fälle abdecken.
Principle 3: Regression testing
Überprüfe, ob Fehler, welche behoben wurden, zu einem späteren Zeitpunkt wieder auftreten.
Principle 4: Applying oracles
Ein Test welcher sagt, es ist „vielleicht“etwas schief gegangen bringt nichts. Eine Überprüfung vom Tester kostet Zeit. Spezifikationen und Verträge (z.B. zwischen Modulen) können genutzt werden, um einen klaren Soll-Zustand zu formulieren.
Principle 5: Manual and automatic test cases
Nicht die Ausführung, sondern die Wahl der Eingabewerte betreffend. Automatisch generierte Tests sind gut in der Breite. Viele verschiedene Eingabewerte und Parameter, welche der Mensch eventuell vergessen würde, werden getestet. Manuell generierte Tests sind gut in der Tiefe. Nutzen das Wissen des Programmierers, welcher das Problem und die Datenstrukturen kennt.
Principle 6: Empirical assessment of testing strategies
Wähle die Teststrategie, welche am meisten Nutzen bringt. Auch Teststrategien die wenig bis kein Wissen über das Problem nutzen, können Fehler aufdecken.
Principle 7: Assessment criteria
Der Nutzen einer Teststrategie wird über die Anzahl aufgedeckter Fehler pro Zeit gemessen.

Continuous Integration[4]

Continuous Integration ist ein Verfahren, bei dem nach jedem Commit überprüft wird, ob durch die erfolgte Änderung Fehler in das System gelangt sind. Um dies zu erreichen werden vom Continuous Integration Tool Builds ausgelöst und automatische Softwaretests ausgeführt. So ist jederzeit ein Build auf aktuellem Stand des Projekts zur Verfügung, und durch die Softwaretests abgedeckte Fehler werden direkt erkannt. Die Eigenschaften dieses Prozesses ermöglichen die Steigerung der Softwarequalität, da zum Beispiel Integrationsprobleme vor einem Meilenstein viel früher erkannt werden und häufiges Commiten motiviert wird.

Tools

- https://travis-ci.com/
- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Mozmill
- https://wiki.ubuntu.com/QATeam/Testdrive

[3]Seven Principles of Software Testing, Professor Bertrand Meyer, ETH Zürich
[4]http://en.wikipedia.org/wiki/Continuous_integration