Hochschule für angewandte Wissenschaften Augsburg
Fakultät für Informatik

Exposé

Vergleich von Python Test-Tools für Test-driven development

zur Erlangung des akademischen Grades
Bachelor of Science

Thema:	Vergleich von Python Test-Tools für Test-driven development
Autor:	Maximilian Konter maximilian.konter@hs-augsburg.de MatNr. 951004
Version vom:	7. März 2019
1. Betreuer:	Dipl.-Inf. (FH), Dipl.-De Erich Seifert, MA
2. BetreuerIn:	Prof. Dr. X

Inhaltsverzeichnis

Abkürzungsverzeichnis 3

Glossar 4

1 Ausgangslage 5

2 Problembeschreibung 5

3 Fragestellung 5

4 Zielsetzung 6

5 Theoretische Grundlage 6

6 Konzept 7

7 Vorläufige Gliederung 8

8 Motivation 10

Literaturverzeichnis 11

Abkürzungsverzeichnis

GNU	GNU is not Unix
GPL	GNU General Public License
GUI	Graphical User Interface
LGPL	GNU Lesser General Public License
TDD	Test-driven development

Glossar

Continous testing Bezeichnet den Prozess des automatischen testens mithilfe eines automatisierungs Tools. Hierbei werden die tests durch ein Ereignis automatisch ausgeführt, wodurch der Entwickler sofort ein Feedback bekommt.. 6

mocken Etwas mocken bedeutet ein Objekt durch ein Falsches Objekt zu ersetzen das genau so aussieht wie das erwartete Objekt, aber nicht das gleiche ist. 7

Pythonista Jemand der Python als seine favoritisierte Sprache verwendet. 10

1 Ausgangslage

TDD wird in der heutigen Softwareentwicklung immer verbreiteter und beliebter. Die Ansprüche an Software sind in den letzten Jahren immer weiter gestiegen. Dies liegt vor allem an der Reichweite, die Software heute hat. So besitzen im Jahr 2018 bereits circa 66% aller Menschen ein Smartphone [Sch17], im Arbeitsleben ist ein PC meist gar nicht mehr weg zu denken. Doch mit den steigenden Nutzerzahlen steigen auch die Anforderungen, die die Nutzer an die Software stellen. Somit wird die Anzahl der gefundenen Bugs dementsprechend größer.

2 Problembeschreibung

Im weltweiten Markt gibt es viele große Unternehmen, die gegenseitig um die Nutzer kämpfen. Selbstverständlich präferieren die Nutzer denjenigen Anbieter, welcher die bessere Software bietet. Dies kann sich heute jedoch stetig ändern. Mit der steigenden Anzahl an Bugs, die gefunden werden, steigt auch die Anzahl der Nutzer, die von diesen Bugs betroffen sind. Diese Bugs sollen natürlich schnellstmöglich gefixt werden um so zu verhindern, dass die Nutzer die Software wechseln.

So schwer es ist seine Nutzer zu halten, umso schwerer ist es zum Start einer Software Nutzer zu akquirieren. Es gibt bereits andere Software, die ähnliche Services anbieten. So ist es noch schwerer dem Markt beizutreten. Die Anforderungen sind durch die anderen Firmen höher als die Anforderungen an ein neues Produkt und alte Fehler, die in anderer Software schon gelöst wurden, sollten möglichst nicht auftauchen.

Für Unternehmen sind diese Anforderungen meist schwer zu meistern weshalb Software meist mit Fehlern released wird, um diese dann von den Nutzern aufdecken zu lassen und zu fixen.

3 Fragestellung

Welche Möglichkeiten stehen einem Entwickler/Arbeitgeber zur Verfügung Python als Sprache für TDD einzusetzen? Daraus resultierend stellt sich die Frage welches der Test-Tools das beste dafür ist, oder welche sich kombinieren lassen um das beste aus ihnen heraus zu holen?

Welchen Vorteil zieht ein Entwickle/Arbeitgeber daraus TDD zu verwenden? Ist der betriebene aufwand und die daraus resultierende Test-Abdeckung es wert TDD anderen Techniken vorzuziehen?

4 Zielsetzung

Das Ziel dieser Arbeit ist es, Test-Tools der Programmiersprache Python in den Aspekten Anwendbarkeit, Effizienz, Komplexität und Erweiterbarkeit zu vergleichen. Daraus resultierend soll eine Empfehlung entstehen wie man möglichst einfach TDD mit Python betreiben kann, um so die Motivation Tests zu schreiben zu steigern.

Um die effizient von TDD zu steigern wird in dieser Arbeit auf die Automation der Tests eingegangen, dies wird mithilfe des Tools Travis CI[1] darzustellen.

Nach dem Vergleich soll der Leser ein Bild darüber haben welche Tools ihm zur Verfügung stehen um Tests zu schreiben oder TDD zu betreiben. Zusätzlich soll er ein wenig Verständnis darüber haben wie er mithilfe von Continous testing effektiver TDD betreiben kann.
Am Ende der Arbeit werden die Vor- und Nachteile von TDD abgewogen, für die Nachteile soll eine Lösung angeboten werden (soweit dies möglich ist) um TDD attraktiver zu gestalten.

5 Theoretische Grundlage

Hier aufgeführt findet sich einiges an Literatur, welche relevant für diese Arbeit sein könnte.

- 5-jährige Studie von IBM aus dem Jahr 2007 `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.6319&rep=rep1&type=pdf`
- Diskussion zu einem Buch über Empirische Softwareentwicklung (Paywall) `https://www.infoq.com/news/2009/03/TDD-Improves-Quality`
- Studie über den Gebrauch von TDD und dem daraus resultierenden Design und Testverbesserungen `https://arxiv.org/pdf/1711.05082.pdf`

[1] Travis CI Website

- TDD vs Test-Last https://www.researchgate.net/publication/315743099_An_Experimental_Evaluation_of_Test_Driven_Development_vs_Test-Last_Development_with_Industry_Professionals
- TDD vs nicht TDD auf Github https://peerj.com/preprints/1920/
- Beispiel für TDD mit Python https://medium.freecodecamp.org/learning-to-test-with-python-997ace2d8abe
- Getting started with testing in Python https://realpython.com/python-testing/
- Minimum viable testing https://realpython.com/the-minimum-viable-test-suite/
- Continuous Integration With Python: An Introduction https://realpython.com/python-continuous-integration/
- Test-driven development with Python (Buch) https://www.obeythetestinggoat.com/

6 Konzept

Die Test-Tools aus der Standard Bibliothek von Python sowie die größten Test-Tools die es sonst noch gibt werden anhand folgender Aspekte verglichen:

- Anwendbarkeit

  Bietet das Tool alles um TDD betreiben zu können? (Bei TDD kann man Tests mocken, wodurch sie nicht schief gehen) Ist das Tool abhängig von vielen Paketen, die man extra installieren muss, da sie nicht in der Sandard Bibliothek sind?

- Effizienz

  Wie viel lässt sich mit diesem Tool möglichst einfach und schnell erreichen? Ist besonders viel Vorarbeit notwendig um die Tests auf zu setzen oder kann sofort mit dem schreiben der Tests begonnen werden.

  Genau so stellt sich die Frage wie schnell kann der Entwickler sehen und verstehen wieso Tests schiefgegangen sind?

- Komplexität

  Wie komplex ist das Tool? Das heißt, wie viel Funktionalität bietet das Tool dem Entwickler von Haus aus, aber auch wie schwer es ist Code zu schreiben oder wie schnell Code unübersichtlich wird, da das Tool viel Code abseits der Tests benötigt.

- Erweiterbarkeit

  Wie leicht lässt sich das Tool mit anderen Tools erweitern? Gibt es vielleicht Erweiterungen der Community für dieses Tool die sehr hilfreich sind?

- Eventuell werden beim schreiben der Arbeit weitere Aspekte entstehen die hier aufgeführt werden können.

Dieser Vergleich entsteht zum einen durch das analysieren der Tools auf der jeweiligen Website sowie durch ein komplexes Code-Beispiel das mit dem jeweiligen Tool geschrieben wird. Passend zum Code-Beispiel wird auch analysiert wie einfach die Automatisierung mit Travis CI verläuft indem dies auch mit echtem Code getestet wird.

Sind die Grundlagen für den Vergleich geschaffen, werden die Tools untereinander verglichen und ihre schwächen und stärken noch einmal zusammen gefasst.

Sofern dies möglich ist wird versucht die Tools zu kombinieren um etwaige Schwächen einen Tools zu umgehen.

Nachdem das/die beste(n) Tool(s) gefunden wurde, wird auf die schwächen von TDD im allgemeinen eingegangen. Dies behandelt den erforderlichen Aufwand und die daraus resultierende Zeit die "verloren"geht (Zeit die statt für Tests schreiben auch für Code schreiben genutzt werden könnte) sowie die Disziplin die den Entwicklern abverlangt wird, zuerst Tests zu schreiben bevor die tatsächliche Funktionalität geschrieben wird. Dabei wird auch noch auf den Wirtschaftlichen Aspekt von TDD eingegangen den ein Arbeitgeber in zu beachten hat bei der Entscheidung TDD zu verwenden.

Am Ende der Arbeit werde ich meine Persönliche Meinung über TDD und ob ich es als Entwickler oder Arbeitgeber nutzen würde Preis geben.

7 Vorläufige Gliederung

1. Abstract
2. Einleitung

2.1. Was ist Python?

2.2. Was ist Test-driven development?

3. Python Test-Tools

3.1. Welche Tools bieten die Standard Bibliothek von Python?

3.2. Welche Tools gibt es abseits der Standard Bibliothek?

3.2.1. Python Test-Tools

3.2.2. Python-Framework Test-Tools (Nicht im vergleich)

3.2.3. Weitere Tools

3.3. Tool XY

3.3.1. Code-Beispiel

3.3.2. Anwendbarkeit

3.3.3. Effizienz

3.3.4. Komplexität

3.3.5. Erweiterbarkeit

4. Zusammenfassung

4.1. Tool XY

5. Vergleich der Tools

6. Kombinierung von Tools

7. Diskussion: Lohnt sich TDD

7.1. Welche stärken hat TDD?

7.2. Welche schwächen hat TDD?

7.2.1. Welche Möglichkeiten gibt es diese schwächen zu umgehen?

7.2.2. Welche schwächen müssen akzeptiert werden?

7.3. Welche Wirtschaftlichen Aspekte müssen bei TDD beachtet werden?

7.4. Zusammenfassung

8. Fazit

9. Nachwort

8 Motivation

Da ich selbst ein Pythonista bin, war es für mich naheliegend meine Bachelorarbeit dieser Sprache zu widmen. Durch mein Praxissemester und dem daraus resultierenden Werkstudentenjob, in dem ich seit eineinhalb Jahren an der Test- und Buildautomatisierung arbeite, wurde mein Interesse in die automatisierte Ausführung von Tests geweckt. Auch mein Bedürfnis, Tests für meine Programme zu schreiben ,ist enorm gewachsen, da ich festgestellt habe, wie gut das Gefühl ist, wenn man etwas im Code ändert und alle Tests danach noch zu 100% durchlaufen.

Seitdem ist Testen ein Teil meiner Entwicklung und auch meines Denkens. TDD selbst habe ich jedoch noch nicht ausprobiert, aber der Reiz für mich ist hoch, da ich denke, dass Testen von Anfang an sich nur positiv auf die Software auswirken kann.

Auch mein Interesse für die Automatisierung möchte ich in dieser Bachelorarbeit widerspiegeln, da dies den Entwicklungsprozess erheblich erleichtert und schneller gestaltet.

Literaturverzeichnis

[Sch17] Schobelt, Frauke: Weltweite Smartphone-Verbreitung steigt 2018 auf 66 Prozent. (2017). – Online erhältlich unter `https://www.wuv.de/digital/weltweite_smartphone_verbreitung_steigt_2018_auf_66_prozent`; abgerufen am 25. Januar 2019.