Verbesserung und Beibehaltung der Qualität bestehender Software mit Docker

Armin Grodon
Hochschule für Angewandte Wissenschaften München
me@armingrodon.de

***Zusammenfassung*—Ähnlich der Entropie in der Thermodynamik, entwickelt sich auch Software mit der Zeit von einem qualitativ hochwertigen, geordneten, hin zu einem qualitativ minderwertigen, ungeordneten, Zustand. Sei es indirekt, durch den wachsenden Qualitätsanspruch an Effizienz und Funktionalität, oder direkt, durch nachträgliche Änderungen an der Software.**
Ziel dieser Forschungsarbeit ist es zu untersuchen, ob und wie die Qualität von Software mit der Virtualisierungs- und Isolierungs-Software Docker verbessert oder zumindest beibehalten werden kann.
Dafür sollen zwei Methoden mit unterschiedlichen Grundgedanken entwickelt, an bestehender Softwareprojekten getestet und untereinander sowie gegen die ursprüngliche Entwicklung verglichen werden. Durch die gesammelten Ergebnisse soll gezeigt werden, für welche Art von Softwareprojekten sich welche Methode am besten eignet.

I. Einleitung

Oft ist es in Unternehmen der Fall, dass bestehende Software im Einsatz ist, deren ursprüngliche Entwickler das Unternehmen bereits verlassen haben. Besonders bei wenig dokumentierter und nicht handwerklich erstellter Software, können dadurch eine Vielzahl von Problemen entstehen.
Muss die Software erweitert oder ein Fehler behoben werden, besteht die Gefahr, stattdessen weitere Fehler hinzuzufügen.
Durch diese Gefahr und den Wegfall der Wartbarkeit solcher Projekte werden diese oft auch komplett neu entwickelt. Aber auch hierfür müssen erst einmal finanzielle sowie personelle Mittel zur Verfügung stehen.
Allgemein wäre es also wünschenswert, wenn sich bestehende Software in einer isolierten Umgebung weiterentwickeln ließe, ohne dabei die bestehende Qualität zu gefährden. Zu diesem Zweck würde sich die Software Docker, oder allgemein vergleichbare Container-Systeme, sehr gut anbieten. Durch den, im Vergleich zu Virtuellen Maschinen, sehr geringen Overhead[1] und die gute Portierbarkeit, sind diese besonders bei Unit-Tests durch kürzere Test-Zeiten weit überlegen.

Als sehr vielversprechend erscheinen dabei die Herangehensweisen, bestehende Software in einer isolierten und testbaren Umgebung weiterzuentwickeln oder in der bestehenden Form zu isolieren und als Microservice in eine Service Oriented Architecture (SOA) einzubetten.

II. Stand der Forschung

Da Docker eine sehr neue und schnell wachsende Software ist, gibt es in diesem Bereich eine Vielzahl aktueller Forschungsprojekte und -arbeiten. Die meisten dieser Arbeiten befassen sich allerdings mit den Themen "DevOps"[2] und "Deployment"[3], also eher dem Bereitstellen von Systemen und dem Ausspielen und Verteilen von Programmen. Dies hilft in sofern indirekt, dass dadurch die allgemeine Reife und Qualität von Docker an sich beleuchtet wird. Auch wurde bereits viel im ebenfalls sehr aktuellen Bereich SOA[4], [5] geforscht. Auch hier ist die Einsicht sehr hilfreich, in welchen Arten von Projekten sich eine SOA besonders oder eher weniger anbietet. Diese Themengebiete betrachten zwar den Einsatz von Containern, allerdings nicht im Kontext der Software Maintenance.
Zwei Arbeiten, die eher in diese Richtung gehen befassen sich mit der Reproduzierbarkeit von Forschung[6] und dem Testen von Software mit Docker[7]. Diese Arbeiten beleuchten aber auch lediglich wieder nur Teilbereiche, die hohe Reproduzierbarkeit und Parallelisierbarkeit von Softwaretests mit Docker. Der Einfluss auf die Qualität wird dabei nicht betrachtet.

III. Ziele und Arbeitsprogramm

In diesem Abschnitt wird das Ziel des Forschungsprojekts, sowie die Aufteilung des Arbeitsprogramms beleuchtet.

A. Voraussichtliche Dauer und Kosten des Projekts

Da sich das Projekt in sehr vielen Themenbereich gleichzeitig bewegt (Docker/Container, SOA und Software Maintenance) wird für die Literaturrecherche in Tabelle I auch ein verhältnismäßig hoher Zeitaufwand veranschlagt.

Tabelle I
Zeitplanung

Tätigkeit	**Dauer in Monaten**
Literaturrecherche	3,0
Erarbeitung der Methoden	2,0
Test in Kontrollgruppen	3,0
Auswertung der Ergebnisse	0,5
Zusammenfassung in Empfehlung	1,0
Summe	**9,5**

Nach der Entwicklung der beiden Referenz-Methoden ist der nächste größere Block das vergleichsweise Testen dieser in möglichst ähnlichen Kontrollgruppen an verschiedenen Softwareprojekten.
Die Ergebnisse dieser Tests sollen anschließend von den Probanden gesammelt und ausgewertet werden. Diese Informationen sollen abschließend zusammengefasst werden, mit dem Ziel, eine Empfehlung geben zu können, für welche Art von Projekt sich welche Methode am Besten eignet.
Dadurch ergibt sich die in Tabelle I angegebene Gesamtdauer von 9,5 Monaten.

Tabelle II
Kostenplanung

Posten	**Kosten**
Personal	91.875 €
Hardware	10.000 €
Labor	20.000 €
Summe	**121.875 €**

Die Personalkosten in Tabelle II setzt sich aus einer Vollzeitstelle über den gesamten Forschungszeitraum (9,5 Monate, je 3500 €), 10 Teilzeitkräfte für die Kontrollgruppen (3 Monate, je 1750 €) sowie einer Teilzeitkraft für die Mithilfe bei der Betreuung der Tests und der Auswertung der Ergebnisse (3,5 Monate, je 1750 €) zusammen.
Zusätzlich muss für die Forschungsarbeit ein Labor und für den Zeitraum der Tests ein zusätzlicher Laborraum und Hardware angemietet werden.

B. Ziele

Die Forschungsarbeit hat das Ziel zwei verschiedene Herangehensweisen zu Betreiben und Weiterentwickeln bestehender Software zu entwickeln, diese gegeneinander zu testen und eine Empfehlung zu geben, für welche Art von Softwareprojekt welche Art von Methode geeignet ist.
1) Gesicherte Weiterentwicklung: Die erste Methode lehnt sich an das Prinzip der Unit Tests und Test Driven Development an. Die bestehende Software soll in einem Docker Container isoliert und eine Spezifikation von Unit Tests gegen diesen Container geschrieben werden. Die Unit Tests beschreiben dabei das Verhalten der gesamten Software als Black Box und stellen die Korrektheit der Software, auch über die Weiterentwicklung hinweg, sicher.
2) Gekapselte Weiterentwicklung: Die zweite Methode baut auf das Prinzip der SOA auf. Die bestehende Software wird ebenfalls in einem Docker Container isoliert, allerdings anschließend nicht weiterentwickelt. Stattdessen wird die Software in eine SOA eingebettet und zusätzliche Funktionen als neue Microservices implementiert.
3) Test: In den Kontrollgruppen soll durch Code Reviews, eine Zufriedenheitsbefragung der Teilnehmer und die Messung der Implementierungsdauer und -güte ermittelt werden, wie gut sich das getestete Verfahren für die gegebene Software eignet. Dies soll durch die Auswahl verschiedener Projekte Größe, Komplexität) geschehen.
4) Empfehlung: Aus den Testergebnissen soll abschließend eine Empfehlung erstellt werden, ob und welche der beiden Methoden sich besonders gut für welche Art von Softwareprojekt eignet.

Literatur

[1] S. Binet and B. Couturier, "docker & hep: containerization of applications for development, distribution and preservation," in *Conference on Computing in High Energy and Nuclear Physics–CHEP2015*, 2015.

[2] M. Stillwell and J. G. F. Coutinho, "A devops approach to integration of software components in an eu research project," in *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, QUDOS 2015, (New York, NY, USA), pp. 1–6, ACM, 2015.

[3] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, Mar. 2014.

[4] A. Tosatto, P. Ruiu, and A. Attanasio, "Container-based orchestration in cloud: State of the art and challenges," in *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*, pp. 70–75, July 2015.

[5] V. Baraldo, A. Zuccato, and T. Vardanega, "Reconciling service orientation with the cloud," in *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pp. 195–202, March 2015.

[6] C. Boettiger, "An introduction to docker for reproducible research, with examples from the R environment," *CoRR*, vol. abs/1410.0846, 2014.

[7] M. Rahman, Z. Chen, and J. Gao, "A service framework for parallel test execution on a developer's local development workstation,"