Richtlinie zum „Software-Sprint“

COVER-REST

Schlussbericht

Zuwendungsempfänger:

Cover-Rest-Thorsten-Behrens-Urs-Schubert-Linus-Behrens-GbR

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS21S42 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Kurze Darstellung der Aufgabenstellung und Motivation

Der Software-Sprint „COVER-REST“ löst ein Problem, dem Softwareentwickler häufig begegnen. Es schafft Synergien durch (Coverage) Test Tools, indem diese zunächst über eine einheitliche (REST) Schnittstelle angesprochen werden. Nicht nur einheitliche Statistiken sind dadurch möglich, sondern auch “Landkarten” für Source Code werden erzeugt, wodurch sich der Ort von Funktionalität (Features) im Source Code lokalisieren läßt. Dies hilft Neulingen, sich in großen Projekten zu orientieren. Es können auch gezielt die richtigen Tests für ein Feature gestartet werden, um ein effektives, schnelles Testfeedback zu erhalten.

Hierzu wurden anhand zweier bestehender Software-Projekte (libreoffice sowie odftoolkit) Feature-Tests implementiert, sowie anhand prototypischer Implementierungen von Analysetools, CI-Pipelines und grafischer Visualisierungstechniken eine erste Lösung erstellt.

Beitrag des Projektes zu den Zielen der Förderinitative „Software-Sprint“

In der OpenSource-Entwicklung sind nicht alle Menschen unseres Planeten gleich repräsentiert. Ganz im Gegenteil, es ist nach wie vor eine kleine Elite, die es sich (gesponsert oder durch reichhaltige Freizeit) leisten kann. Software ist groß und komplex - und der Einstieg ist häufig nur mit großem Zeitaufwand möglich. Mit Cover-Rest wollen wir Software zugänglicher machen, und den Einstieg in die Entwicklung leichter & spaßvoller gestalten. Hierzu wollen wir mit Cover-Rest das Auffinden von Features automatisieren.

Über geschicktes Auswerten von Testergebnissen und der einheitlichen Orchestrierung von Testwerkzeugen, erhalten Entwickler einen Blumenstrauß von Vorteilen, welche die Softwareentwicklung beschleunigen, die Softwarequalität erhöhen und vor allem dem Entwickler lästige Arbeit ersparen, um seine Lebensqualität zu verbessern. Damit nehmen wir

Softwareentwicklern, die rar und gefragt, und unter ständigem Zeitdruck stehen, lästige, zeitraubende Aufgaben ab – denn diese können von innovativen Softwarewerkzeugen übernommen werden und gleichzeitig die allgemeine Softwarequalität verbessern.

Ausführliche Darstellung der Ergebnisse

Das erste zu adressierende Problem war, dass viele Software-Entwickler an Projekten mit verschiedenen Programmiersprachen arbeiten, wobei jede Programmiersprache gewöhnlich mit einer eigenen Testsoftware daherkommt, wie zum Beispiel verschiedene Code-Coverage Werkzeuge für C++ und Java mit unterschiedlichen APIs und Ausgabeformaten (meist HTML). Um einheitliche Analysen und Statistiken über Testergebnisse zu ermöglichen, haben wir die (Coverage) Test Tools über eine Service API gewrappt (https://gitlab.com/cover-rest/restapi). Diese reduziert Komplexität und vereinheitlicht die Ergebnisse in einer Datenbank. Für den Prototypen haben wir derzeit noch einen Zwischenschritt in Betrieb, welcher statt einer Datenbank das Cobertura-XML-Format als Austauschformat verwendet. Dazu haben wir Konverter erstellt (z.B. https://gitlab.com/cover-rest/odftoolkit/-/blob/coverage/odfdom/src/main/java/org/odftoolkit/odfdom/changes/CoberturaXMLHandler.java)

Der innovative Trick ist, wie hier über Testabdeckung ein Feature getestet wird: wir laden zum Beispiel ein Dokument, in dem “Hello World!” in kursiv steht und merken uns die Testabdeckung. Dann testen wir das Laden des gleichen Dokumentes, nur diesmal ohne kursiven Text. Das erste Dokument unterscheidet sich nur durch das Kursiv-Feature. Subtrahiert man die beiden Testabdeckungen, erhält man die Testabdeckung, die ausschließlich für das Kursiv-Feature verwendet wird. Da bereits automatisch die Features in einem Testdokument bestimmt werden, ist es durch Arithmetik und automatischem Test möglich, eine Terra Cognita unseres Source Codes zu erstellen und via Git-Hooks Tests zu Zeilen zu verlinken, um schnelle Smoke Tests zu ermöglichen. Ein Prototyp hierfür befindet sich hier: https://gitlab.com/cover-rest/odftoolkit/-/blob/coverage/odfdom/diff-coverages.sh

Die diversen Bausteine haben wir darauf folgend für zwei Demo-Setups verwendet, welche basierend auf der Jenkins-Continuous-Integrationsplatform die Coverage-Tests laufen lässt, danach das Feature-Diff berechnet sowie die Codestellen grafisch anzeigt, welche einzigartig für ein gegebenes Feature sind: https://coverrest.allotropia.de/

Eine Illustration des Konzepts auf sehr allgemein verständlichem Niveau findet sich im Video unserer Demowoche: https://demoweek.prototypefund.de/projects/07-cover-rest.html

Weitere eher technische Illustrationen finden sich auf der Projektseite: https://cover-rest.gitlab.io/

Zielgruppe, Nutzen und mögliche Weiterentwicklungen

Unmittelbaren Nutzen zieht das LibreOffice-Projekt (sowie das ebenfalls unter dem Dach der Stiftung The Document Foundation beheimateten ODFToolkit-Projekt), da für beide Projekte prototypische Setups implementiert wurden, welche über die Zeit hoffentlich die Verständlichkeit, Testbarkeit und Auffindbarkeit des Codes verbessern werden.

Weiterhin und etwas unspezifischer in der konkreten Nutzbarkeit adressiert das Projekt Software-Entwickler im allgemeinen:

Zum einen sollen sich Entwickler, welche Cover-Rest für ihre Software verwenden, über die Feature Map schnell einen Überblick verschaffen können, an welchen Stellen im Code Feature zu finden sind. Damit können so auch mit geeigneten schnellen Tests z.B. eine GUI-Funktionalität im Source Code gefunden werden.

Gerade hier hatten wir während der Projektarbeit noch viele neue Ideen, und hoffen, dass wir über weitere Nutzer & den OpenSource-Ansatz noch weiter wachsen können. Den aktuellen Projektrahmen hätte dies jedoch gesprengt.

Zum anderen sollen Entwickler über schnelle Feature-Tests zeitnah Rückmeldung über mögliche Probleme erhalten. Der Plan ist es, in LibreOffice die wahrscheinliche passenden Feature-Tests gleich zu Beginn aller Tests auszuführen, um Probleme möglichst schnell aufzudecken (sowie potentiell Resourcen und Energie zu sparen).

Hier wurde durch den Prototypen klar, dass wir eigentlich nicht die Zeilen des Programms, sondern den Control-Flow-Graphen unter Beobachtung haben sollten. Denn der Control-Flow-Graph repräsentiert, wie das Programm laufen sollte, ähnlich wie Wasser durch einen Garten fließen würde. Werden Änderungen im Control-Flow-Graphen stromaufwärts vorgenommen (z.B. an zu testenden Codebeiträgen), kann dies erwartete Feature-Codestellen von der Aufrufbarkeit abklemmen. Diese Tests bräuchten dann nicht mehr ausgeführt zu werden.

Schlussendlich ist noch die Erkenntnis gereift, dass zumindest im OpenSource-Umfeld die Code-Coverage kaum nennenswerte Fortschritte im vergangenen Jahrzehnt gemacht hat. So wird das Cobertura-XML Format, welches ein de-facto XML-Standard für Coverage -Auswertungen ist, seit Jahren nicht mehr gepflegt. Könnte das Format erweitert werden, würden auch andere Bereiche der Softwareentwicklung, wie die Resultate von Performance Tests damit speicherbar sein.

Das ist deswegen von Interesse, weil sowohl bei der Code-Coverage, als auch beim Performance Test das Programm vom aufgerufen Test durchlaufen wird. Beide Male wird der Binärcode vom Compiler so instrumentiert, dass er tracebar herausschreibt, welche Zeilen von welchen Sourcefiles durchlaufen worden sind. Dies geschieht äquivalent auch bei einem Performance Test, nur wird dann auch die Reihenfolge und die Zeitdauer der Ausführung einer Zeile (bzw. Statements) festgehalten.

Die Entflechtung des üblichen monolithischen Coverage-Ansatzes in einen codenahen Teil und in eine Datenbank, sowie einer Visualisierung (zunächst als Teil eines JenkinsPlugins), sowie eine mögliche Verknüpfung dieser Blöcke via REST API, ist eine längst überfällige Herangehensweise, die sich auch auf Performance-Tests übertragen ließe.

Fraglich ist, ob und in welcher Weise existierende Coverage-Projekte dazu bewegt werden könnten, diesem Ansatz zu folgen, und welche Anreize dazu nötig wären.

Für die Mitglieder des Cover-Rest-Teams war dieser Software-Sprint in jeder Beziehung eine lehrreiche Erfahrung (Erarbeitung neuer Lösungen & Realisierung einer Idee, Einarbeitung in neue Themenbereiche in einer Tiefe, welche unter normalen Freizeit bzw. wirtschaftlichen Erwägungen so nicht möglich wäre, sowie Teamwork und gegenseitiges Coaching).

Kurze Darstellung der Arbeiten, die zu keiner Lösung geführt haben

Der Lösungsansatz mit dem Versionsverwaltungstool git, aufgrund der Zeilen aus der Feature-Map schnell ermitteln zu können, welche Feature-Tests für eine Änderung benötigt werden, wurde während des Projektes zunächst verworfen, weil wir nicht nur das Verschieben von Funktionalität beim Refaktorieren erfassen müssen, sondern eigentlich noch mehr hochwertiges Control-Flow-Wissen im Tooling haben müssten, um z.B. bei geänderten Funktionsaufrufen korrekt reagieren zu können. Wir sind aber ganz am Ende zum Schluß gekommen, dass es als Heuristik sehr wohl sehr nützlich sein kann und daher doch enabled werden sollte (Fail Fast).

Kurze Angabe von Präsentationsmöglichkeiten für mögliche Nutzer

Interessenten können sich auf den folgenden Webseiten detailliert über die Projektergebnisse informieren:

Website: https://cover-rest.gitlab.io/

Code-Repository: https://gitlab.com/cover-rest

Demo-Server: https://coverrest.allotropia.de/

Weitere Updates sind geplant (insbesondere Ausrollen der Lösungen für das LibreOffice-Projekt). Wir werden auf der Projektseite darüber informieren.

Kurze Erläuterung zur Einhaltung der Arbeits- und Kostenplanung

Wir haben im Vorfeld des Projektes aufgrund der recht langen Vorlaufzeit nach der Bewerbung von einem Einpersonenprojekt auf ein Teamprojekt umgestellt. Dies hat die Schwerpunkte etwas verschoben, und wir haben nicht alle Teilaufgaben erledigen können. Demzufolge resultierte ein Minderaufwand im Gesamtprojekt. Nicht erledigt wurden die folgenden Punkte aus dem ursprünglichen Projektplan:

5. Feature Coverage DB
6. “Zeile-zu-FeatureTest(s)” DB mit Zeilen als Key
7. Git-Hook der geänderte Zeilen mit Zeilen aus der Feature Test DB vergleichtund Featuretests anstößt

Kurze Darstellung von etwaigen Ergebnissen bei anderen Stellen

Wir hatten im Team einen sehr fruchtbaren Ideenaustausch mit Prof. Ullrich Hafner der Fachhochschule München. Prof. Hafner ist Maintainer einiger zentraler Jenkins-Komponenten, sowie im Vorstand des Jenkins-Projektes tätig.