UNIKASSEL
VERSITÄT

Fachbereich Elektrotechnik/Informatik
Fachgebiet Software Engineering

Toolgestütztes Event Storming für das Requirements Engineering

fulibWorkflows

BACHELORARBEIT

zur Erlangung des Grades eines Bachelor of Science
im Fachbereich Elektrotechnik/Informatik
der Universität Kassel

Eingereicht von:	Maximilian Freiherr von Künßberg
Anschrift:	Mönchebergstraße 31 34125 Kassel
Matrikelnummer:	33378673
Emailadresse:	maximilian-kuenssberg@uni-kassel.de
Vorgelegt im:	Fachgebiet Software Engineering
Gutachter:	Prof. Dr. Albert Zündorf Prof. Dr. Claude Draude
Betreuer:	M. Sc. Sebastian Copei
eingereicht am:	22. Februar 2022

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1 Einleitung

1.1 Motivation

TODO: Ach ja das Requirements Engineering in der agilen Softwareentwicklung ist und bleibt ein leidiges Thema. Dafür wurde von Alberto Brandolini das Event Storming ins Leben gerufen. Namensvetter Albert dachte sich: "Ja, geil - Hab ich Bock drauf." So begann er es zu erweitern und nun sind wir alle hier und schauen uns das an.

1.2 Ziele

TODO: Noch kurz warum Event Storming eine gute Idee ist und hilfreich in der Anforderungsanalyse sein kann. Natürlich schon mal kurz ansprechen, dass für Albert(o)s Event Storming eine Beschreibungssprache entwickelt wurde und man diese zum einfachen Bedienen mit einem Web-Editor versehen hat.

1.3 Methodik

TODO: Expertengespräch zum Prüfen der gesetzten Ziele. Was ist das und warum ist es für die Ziele ausreichend?

1.4 Existierende Konzepte

TODO: Oldschool alles auf papier -> keine mockups direkt mit framework Web Editoren wie Miro für das Erstellen von Boards.

1.5 Aufbau der Arbeit

TODO: Zuerst grundlegende Ideen und Technologien. Anschließend die Implementierung beschreiben. Evaluation mittels Expertengespräch mit Adam Malik. Fazit bezüglich Ziele und outcome. Ausblick für das Tool.

2 Grundlagen

Im folgenden Kapitel werden zuerst die grundlegenden Konzepte des *Event Stormings* erläutert. Hierbei wird auf dessen Herkunft und Entwicklung eingegangen. Neben diesen Grundlagen werden anschließend die für diese Arbeit notwendigen Änderungen und Erweiterungen dargelegt. Weiterhin werden die wichtigsten Technologien erläutert, welche für die Implementierung der Anwendungen nötig waren. Um eine bessere Übersicht zu schaffen, sind die Technologien nach dem Anwendungsteil, für welche diese nötig sind, unterteilt.

2.1 Event Storming

Dieses Unterkapitel befasst sich mit der Herkunft des *Event Stormings*, dem Domain-Driven Design (DDD). Zudem wird ein klassischer Ablauf eines *Event Stormings* erläutert und daran die Vorteile dieser Methodik für das Requirements Engineering beleuchtet. Abschließend werden die Änderungen und Erweiterungen, welche im Kontext dieser Arbeit vorgenommen wurden, erklärt.

2.1.1 Allgemein

TODO: Bevor ich dieses und das folgende Unterkapitel schreibe, erst noch mal das [Ver16] und [Bra21] lesen.

- Alberto Brandolini und das ES
- DDD als Grundlage
- Wie verläuft so ein ES Workshop (Beschrieben in seinem Buch, mehrfach)
- Wichtigsten Eckpunkte
- Warum ist es besser als Brain Storming oder ähnliches?

- Beispielhaftes Event Storming Board (Bild und beschreibungstext um später darauf bezug nehmen zu können)

2.1.2 Erweiterung

TODO: Hier das vorherige Kapitel abwarten um alle Änderungen/Erweiterungen besser daran fest zu machen.

- Erweiterungen für Wirtschaft (Pages -> daraus generierte Mockups, abgehen von dem "Wir wollen keinen PC benutzen"des ES)

- Ideen für die Lehre (Wird in dieser Arbeit nicht näher beleuchtet, da es für den Beleg der Funktionalität nicht mehr möglich ist dies ausreichend in der Bearbeitungszeit zu machen)

- ES -> Ablauf von Schritten -> Albert -> Workflow (Arbeitsablauf) beschreibungen -> Mögliche Idee zum besseren Nahebringen von komplexeren Abläufen in Vorlesungen. (Verbildlichung)

2.2 Technologien

Dieses Kapitel gibt einen Überblick über die verwendeten Technologien in der Umsetzung. Da die Implementierung aus verschiedenen Komponenten besteht, ist dieses Kapitel in drei weitere Unterkapitel aufgeteilt. Es wird somit getrennt auf die Java Library *fulib Workflows*, das Spring Boot Backend des *fulib Workflows Web-Editors* und das zum Editor dazugehörige Frontend, welches mit Angular umgesetzt wurde.

2.2.1 fulibWorkflows

fulib Workflows ist eine Java Library, welche Arbeitsabläufe, im Folgenden "workflows", in YAML Ain't Markup Language (YAML)-Syntax notiert als Eingabe nimmt und daraus sowohl ein Event Storming Board, im workflow beschriebene Mockups und Objekt-/ Klassendiagramme generiert. Welche Form die YAML-Eingabe haben muss und wie die Dateien aussehen und generiert werden, folgt in einem späteren Kapitel.

2.2.1.1 Antlr

Antlr bietet die Möglichkeit einen Parser über eine eigens geschriebene Grammatik zu generieren. Die Grammatik muss Links ableitend sein und ist in EBNF. TODO: Was bedeutet das? Der generierte Parser ermöglicht zudem das Aufbauen und Ablaufen eines *Parse trees*. Hierdurch ergibt sich die Möglichkeit während dem Parsen weitere Aktionen durchzuführen, welche den späteren Programmablauf eines Tools unterstützen können.

Listing 2.1: Beispiel einer einfachen Grammatik in Antlr

```
1 grammar AntlrExample;
2 prog:   (expr NEWLINE)* ;
3 expr:   expr ('*'|'/') expr
4 |       expr ('+'|'-') expr
5 |       INT
6 |       '(' expr ')'
7 ;
8 NEWLINE : [\r\n]+ ;
9 INT     : [0-9]+ ;
```

In Listing 2.1 ist ein Beispiel für eine einfache Grammatik zur Erkennung von mathematischen Gleichungen dargestellt.[Par14] Hierbei ist es lediglich möglich Zahlen mittels Klammern, Addition, Subtraktion, Multiplikation und Division miteinander zu kombinieren. Die Länge eines Ausdruckes ist durch den rekursiven Aufbau der Grammatik nicht begrenzt.

Listing 2.2: Einfacher mathematischer Ausdruck

```
1     ((199+2324)*43)/55
```

Die zuvor beschriebene Grammatik kann mittels weiterer Tools auf eine Eingabe geprüft werden. Eine zulässige Eingabe für die festgelegte Grammatik aus 2.1 ist in 2.2 dargestellt. Die Überprüfung auf die Richtigkeit einer Eingabe oder auch der Grammatik kann über Tools bereits vor einer Generierung von Code durchgeführt werden. Hierzu wurde das Diagramm aus Abbildung 2.1 mittels dem Antlr Plugin für IntelliJ generiert.

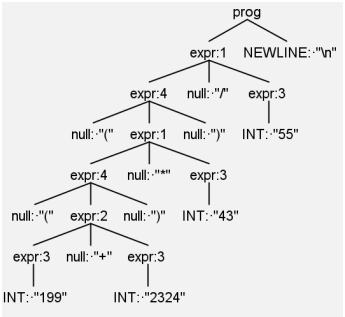

Abbildung 2.1: ParseTree für einen mathematischen Ausdruck

Hierbei ist ersichtlich, dass die Wurzel bei der obersten Regel **prog** beginnt und alle weiteren Kindknoten durch verschachtelte **expr** Regeln kreiert wurden. Der oberste Teilausdruck ist die Division aus einem komplexeren linken Ausdruck und dem rechten Ausdruck, welcher direkt einer Nummer zugeordnet werden konnte und somit keine weiteren Kindknoten mehr haben kann. Dies entsteht durch die Unterscheidung bei der Grammatik in Terminale und Nicht-Terminale. Hierbei werden wie in 2.1 dargestellt Nicht-Terminale Regeln kleingeschrieben und Terminale werden in Großbuchstaben verfasst. In der vereinfachten Grammatik sind lediglich ganze Zahlen als Eingabe erlaubt.

Dies sind lediglich die Grundlagen von Antlr, auf die genauere Verwendung des generierten Parsers und Besonderheiten der Grammatik wird in 3.1.1 eingegangen.

2.2.1.2 String Templates

TODO: jo *StringTemplates* ermöglichen es Schablonen (im Weiteren: Templates) für Dateien zu erstellen und diese dynamisch mit Werten füllen zu können.

2.2.1.3 JSON-Schema

Bei einem JSON-Schema kann man die erlaubten Eingaben eines Nutzers, sowohl für JSON-Dateien als auch YAML-Dateien beschränken.

TODO: Welche Version?, Schemastore.org, wie funst das mit den schemas?

Durch ein fest definiertes Schema ist es vielen IDEs, darunter auch IntelliJ und VSCode, welche aus einem Schema nicht nur die fertige Datei auf Fehler überprüfen, sondern dem Entwickelnden bereits zum Zeitpunkt des Schreibens einer Datei mittels Autovervollständigung unterstützen kann. Eine Liste aller IDEs, welche diesen support mittels schemastore unterstützen sind unter folgendem Link zu finden: `https://www.schemastore.org/json/#editors`

2.2.1.4 fulibTools

TODO: Dank fulibTools ist auch fulib mit drin. FulibTools ist zur Generierung von Objektidiagrammen genutzt worden. Fulib (Bei FulibTools mit integriert) zur Generierung von Klassendiagrammen.

2.2.2 fulibWorkflows Web-Editor FE

TODO: Da brauchte es etwas mehr als beim BE. Die Entscheidungen für die verwendeten Technologien im Frontend wurden aufgrund der Idee der Integrierung vom Editor auf fulib.org getroffen.

2.2.2.1 Angular

TODO: Wir kennen es. Wir lieben es.

2.2.2.2 Bootstrap

TODO: Alles für den Dackel, alles für den Club unser Leben für die schön gestylten FEs. Simpel, oder? Ja, okay. Man nutzt dann auch ng-bootstrap für Angular Anwendungen. Natürlich auch noch bootstrap-icons. Will ja niemand traurig machen und von Adrian verdroschen werden.

2.2.2.3 Codemirror

TODO: Schönes Ding. Ngx-codemirror ist es dann speziell für eine Angular Anwendung geworden. Eigentlich alles out of the box benutzt.

2.2.2.4 Angular-split

TODO: Find ich schon gut zu erwähnen. Ohne die geile dependency wäre das FE nie so pornös geworden.

2.2.2.5 file-saver

TODO: Weitere erwähnenswerte dependency. Wird genutzt um Dateien, die vom BE kommen, auch herunterladen zu können.

2.2.2.6 ajv

2.2.2.7 js-yaml

2.2.3 fulibWorkflows Web-Editor BE

TODO: Yeey alle Technologien die ich im Backend benutzt habe.

2.2.3.1 Spring Boot

TODO: Framework mit dem man easy mal ein backend generiert bekommt. Durch Java und viele Dependency allerdings alles andere als ein leichtgewicht.

Dennoch musste ein Java backend her, da sonst fulibWorkflows nicht hätte integriert werden können. Jedenfalls nicht ohne noch mehr middle ware.

Zudem hatte ich im Praktikum mit Spring Boot Erfahrungen gesammelt. Die Verwendung von Annotations und dem aufsplitten zwischen Controller und Service ist mir bereits durch Nest.js bekannt gewesen.

Dennoch muss man sagen, dass durch die von Spring Boot bereits integrierten libraries nichts weiter außer fulibWorkflows hinzugefügt werden musste. Immerhin umfasst der Code vom Backend vielleicht 300 Lines of Code.

2.2.3.2 fulibWorkflows

TODO: Sollte mindestens irgendwo erwähnt werden. Kann man ggf. auch in dem Text zum Kapitel schreiben.

2.2.4 Deployment

TODO: Ein Web Editor will natürlich für alle erreichbar sein. Und fulibWorkflows muss auch irgendwo bereitgestellt werden, damit es das Backend und alle anderen interessierten benutzen können.

2.2.4.1 MavenCentral

TODO: MavenCentral ein wirklicher Hussarones was das publishen angeht. Glücklicherweise ist fulibWorkflows Teil der Fujaba Tool Suite, wodurch die benötigten Zugriffsrechte bereits vorhanden und andere Libraries bereits gepublished wurden. Hierdurch war es recht schnell möglich mit dem zuvor erworbenem Wissen fulibWorkflows zu publishen.

2.2.4.2 Heroku

TODO: Der Web-Editor soll immer erreichbar sein. Dies ist durch Heroku nur bedingt möglich. Heroku bietet allerlei Möglichkeiten verschiedenste Anwendungen bereitzustellen. Auch mit einem kostenlosen Plan ist es ohne Probleme möglich solch kleine Anwendungen bereitzustellen.

FE Deployment war easy, auch wenn ich erstmal wieder in eins meiner früheren Projekte gucken musste. BE Deployment war kniffliger, doch man ist nie der erste der eine Spring Boot application auf Heroku deployen will. Daher Tutorial reingefahren und ab ging der gebutterte Lachs.

3 Implementierung

TODO: Hier bin ich mir tatsächlich unsicher was alles reinsoll. Ich teile es auch erstmal in drei auf.

3.1 fulibWorkflows

TODO: Was gibt es hier so interessantes zu sehen? Gehen Sie weiter.

3.1.1 fulibWorkflows Grammatik

TODO: Beschreiben der ANTLR4 Grammatik für fulibWorkflows

Und natürlich auch wie mir das beim parsen der yaml geholfen hat. Dat wird ne lange Sektion.

3.1.2 Generierung dank fulibTools

TODO: FulibTools gibt gute Anbindung an Graphviz, wenn man sowieso schon mit fulib arbeitet.

3.1.3 schema

TODO: Da hab ich lange dran gesessen. Und ich glaube, selbst jetzt ist es kein gutes Schema. Allerdings tut es was es soll.

3.1.3.1 Mockups

TODO: Eigenes Datenmodell gebaut. Daraus die wichtigsten Infos gezogen. Dank String-Templates von antlr richtig easy zu bauen. Gilt für Html als auch Fxml.

3.2 editor-frontend

TODO: Alles was es zum FE so gibt.

3.2.1 iframes

TODO: Naja der editor basiert einfach darauf, dass es iframes gibt. Könnte man auch weg lassen?

3.2.2 codemirror

TODO: Eingerichtet und los ging es. Noch einen eigenes kleines Hint Add on geschrieben. Feddig. Musste es erstmal simpel halten. Gibt noch genügend zukünftige Ideen.

3.2.3 angular split

TODO: Danke an Adrian, der mit dazu geraten hat.

Nachdem ich mit purem css da grids erstellt habe, stand ich vor dem Problem der Veränderung von Größen. Angular split löst das Problem auf eine wunderbare Art und Weise.

Die dreiteilung der View war damit wirklich einfach. Auch wenn man aufpassen musste beim Verändern der größen, wenn man iframes benutzt. Da musste ein kleiner Fix rein, der aber auch bereits von den machern vorgegeben war.

3.3 editor-backend

TODO: Das wird wieder kurz.

3.3.1 Spring booterino

TODO: Gabelstablinski hier drüben war dank ein paar Annotations schnell erstellt und macht bisher keine Probleme.

Die zip Datei wird im BE zusammengebaut. Dafür gibt es schon alles fertig in java.util.zip. Da hab ich nach gegoogelt und alles lief wie von selbst.

Na gut, ich muss zugeben, dass die zip vom BE ans FE schicken und dann richtig runterladen können, ohne das mir alles um die Ohren fliegt schon einen Arbeitstag gebraucht hat. Manchmal muss man lediglich seine Dummheit für einen Moment ablegen.

4 Evaluation

TODO: Eigentlichen Anwendugszweck beschreiben

Event Storming nachvollziehbar für alle machen. Htmls einfach noch mal öffnen.

Besser mit Kunden über das Layout sprechen können -> Mockups mit grundlegendem Styling Kunden können einen Workflow mal durchklicken -> Next prev page

4.1 Expertengespräch

TODO: Gespräch mit Adam. Fragenkatalog erstellen. Alles aufzeichnen, wenn Adam zustimmt auch Bild und Ton -> Youtube bereitstellen nicht gelistet

4.2 Auswertung

Was ist die Meinung des Experten zu der Anwendung?

Kann ihm das helfen? (Begründung durch seine Vorerfahrungen)

5 Fazit

Machen die Erweiterungen Sinn?

Kann der Editor von Leuten verwendet werden, die nur wenig Programmier erfahrung haben?

Hilft es beim RE in der Wirtschaft?

Füllt diese Anwendung eine bestehende Lücke im RE? Wurde das zuvor gesetzte Ziel erreicht? Ist die grundlegende Idee gut, aber die Umsetzung nicht ausreichend durchdacht?

Alberts Drang immer wieder neues zu haben xD

6 Ausblick

Was hatten die Leute zu meckern und sollte daher umgesetzt werden?

Was ist mir selbst an Ideen gekommen?

fulibWorkflows ist Teil von Fulib und sollte daher auch über fulib.org erreichbar sein. Alles zusammen an einem Ort macht mehr Sinn als alles verstreut zu haben.

7 Quellenverzeichnis

[Par14] Terence Parr. *Antlr*. 2014. URL: https://www.antlr.org/.

[Ver16] Vaughn Vernon. *Domain-Driven Design Distilled*. Addison-Wesley, 2016. ISBN: 978-0-13-443442-1.

[Bra21] Alberto Brandolini. *Introducing Event Storming*. 2021.