

Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit

Integration von manuellem Testen in das Continuous Deployment durch das SystemTestPortal

Steffen Schneider

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. Stefan Wagner

Betreuer/in: Dr. Ivan Bogicevic

Beginn am: 16. April 2018

Beendet am: 16. Oktober 2018

Kurzfassung

Qualitätssicherung durch manuelles Testen scheint heute in der agilen Softwareentwicklung fehl am Platz zu sein. Kontinuierliche Entwicklung benötigt für die hochfrequente Integration und Auslieferung schnelle und wiederholbare Tests. Menschen sind langsamer und bei repetitiven Aufgaben nicht konsequent und teilweise überfordert. Dennoch ist die Kreativität des Menschen beim Testen der ausschlaggebende Faktor.

Mit dieser Arbeit soll ein Ansatz der Integration von manuellen Tests in die kontinuierliche Entwicklung vorgestellt und untersucht werden. Hierbei werden die Ergebnisse der manuellen Tests verwendet um automatisch den Veröffentlichungsprozess zu steuern. Gleichzeitig wird der Verwaltungs- und Kommunikationsaufwand reduziert. Damit kann manuelles Testen für die Praxis attraktiver gemacht und eine Steigerung der Testqualität bewirkt werden. Das zeichnet sich durch weniger Defekte im Endprodukt, kürzere Entwicklungszeiten und Kosteneinsparungen aus.

Zur Evaluation wurde der Ansatz in einem Prototypen auf seine Praxistauglichkeit untersucht. Das Ergebnis zeichnet eine positive Resonanz gegenüber dem Einsatz in der Praxis ab. Um konkrete Änderungen in der Qualität eines Produktes zu messen, ist eine weiterführende Langzeitstudie nötig. Diese Studie misst die Auswirkungen des Einsatzes der Integration von manuellen Tests in das Continuous Deployment.

Basierend darauf lässt sich untersuchen, wie sich manuelle Tests weiter in die kontinuierliche Entwicklung einbinden lassen. Mögliche Gebiete sind die Reduzierung der Ausführungszeit oder die automatisierte Bestimmung einer ausreichenden Testabdeckung.

Inhaltsverzeichnis

1	Einleitung	17
2	Aufgabenstellung	19
2.1	Ziel der Arbeit	19
2.2	Aufbau der Arbeit	19
3	Grundlagen und Stand der Technik	21
3.1	Qualitätssicherung in der Ära der Continuous Deployment	21
3.2	Das SystemTestPortal als Basis der Integration	22
3.3	Integrationen weiterer Testverwaltungsprogramme	26
3.4	Systeme zur Verwaltung des DevOps-Lebenszyklus	26
3.5	Metriken und Feedback	29
3.6	Anwendung der Continuous Practices in der Praxis	30
4	Theoretische Grundlagen der Integration	31
4.1	Anwendungsfälle der Integration	31
4.2	Umsetzungsmöglichkeiten der Integration	34
4.3	Herausforderungen in der Umsetzung	37
4.4	Sicherheitsbedingte Entwurfbeschränkungen	40
4.5	Aktueller Stand der Integration manueller Tests	41
5	Integration in das Continuous Deployment	43
5.1	Grundsätzliche Funktionsweise	43
5.2	Erweiterungen im SystemTestPortal	45
5.3	Kommunikation zwischen dem CI/CD-System und dem SystemTestPortal	46
5.4	Schnittstelle für die Integration im SystemTestPortal	53
5.5	Ausführliches Feedback für die Entwickler	57
5.6	Benachrichtigung der Entwickler bei fehlgeschlagenen Tests	57
5.7	Nutzen der Integration manueller Tests	59
6	Evaluation der Lösung	61
6.1	Auswahl der Probanden	61
6.2	Aufbau der Evaluation	61
6.3	Ablauf des Experimentes	61
6.4	Ergebnisse der Evaluation	63
6.5	Weiterführende Studie zur Untersuchung der Auswirkung auf die Qualität	66
6.6	Probleme und deren Lösung während der Ausführung der Integration . .	68
7	Herausforderungen und Ergebnisse der Umsetzung	69
7.1	Herausforderungen bei der Entwicklung der Integration	69

7.2	Ergebnisse der Umsetzung	71
8	Zusammenfassung und Ausblick	73
8.1	Erweiterungsmöglichkeiten und Ausblick	73
8.2	Weiterführende Forschungsthemen	74
	Literaturverzeichnis	77

Abbildungsverzeichnis

3.1	Detailansicht eines Testfalls	22
3.2	Detailansicht einer Testsequenz	23
3.3	Protokoll des Testfalls „Install DuckDuckGo.com for Microsoft Edge“	24
3.4	Dialog zur Verwaltung der Versionen und Varianten eines Testsystems	25
3.5	Alle Schritte des DevOps-Lebenszyklus [19]	27
3.6	Typische Stages in einer Pipeline [12]	28
4.1	Alle Anwendungsfälle der Integration im Überblick	31
4.2	Ablauf der Integration über Webhooks	35
4.3	Evaluation der Testergebnisse des SystemTestPortals	39
5.1	Systemarchitektur der Integration manuellen Testens	44
5.2	Berechtigungseinstellungen einer Rolle	45
5.3	Aufgabe für Tester mit Version und Variante	46
5.4	Ablauf des manuellen Testens einer neuen Version im SystemTestPortal	54
5.5	Aufgabe für Manager mit Version und Variante	55
5.6	Ausführung der GitLab-Pipeline mit erfolgreichen Tests	57
5.7	ZIP-Archiv mit Testprotokollen in GitLab als Artefakt	57
5.8	Einstellungen des SystemTestPortals für die Benachrichtigung	58
5.9	Automatisch erstelltest Issue eines Testprotokolls zu fehlgeschlagenem Test	59
6.1	Bewertung der Fragen zur Benutzerfreundlichkeit	63
6.2	Vergleich des Ablaufs mit und ohne Integration	67
6.3	Abbruch der Pipeline aufgrund eines Pipeline-Fehlers	68

Tabellenverzeichnis

4.1	Manuelles Testen einer neuen Version	32
4.2	Ergebnisse der manuellen Tests einer Pipeline abrufen	33
4.3	Benachrichtigung der Entwickler bei fehlgeschlagenen Tests	34
7.1	Übersicht der umgesetzten Arbeiten	71

Verzeichnis der Listings

4.1	Beispiel für den Message-Body eines Webhook-Requests	38
5.1	Konfiguration des Docker-Images für die Webapplikation	48
5.2	Konfiguration der GitLab CI-Pipeline	50
5.3	Anfrage an das STP mit den benötigten Informationen im Message-Body .	52
5.4	Anfrage an GitLab zur Erstellung eines neuen Issues	58

Verzeichnis der Algorithmen

5.1	Algorithmus der Webapplikation zur Bestimmung des Testzustandes . . .	51
5.2	Algorithmus zur Evaluation der auszuführenden Aktion	53
5.3	Analyse der Testergebnisse	56

Abkürzungsverzeichnis

API Application Programming Interface. 28

CD Continuous Deployment. 21

CI Continuous Integration. 21

CMD Command. 47

DevOps Development and Operations. 27

GNU GNU General Public License. 21

HTTP HyperText Transfer Protocol. 34

JSON JavaScript Object Notation. 38

MB MegaByte. 47

QA Quality Assurance. 29

STP SystemTestPortal. 19

SUT System Under Test. 22

URL Uniform Resource Locator. 34

1 Einleitung

Parallel zu den Methoden der Softwareentwicklung hat sich das Softwaretesten ebenso weiter entwickelt. Die Zeit, in der ein Entwickler seinen Code „testet“ und anschließend in die Produktion leitet, wo er weiter getestet wird, ist lange vorbei. Die agile und iterative Entwicklung hat diese Art des Testens überfällig gemacht. Manuelles Testen wurde durch automatische Testmethoden abgelöst. Von der Installation bis zu den Lasttests kann alles automatisiert durchgeführt werden. Delivery Pipelines helfen Entwicklern, jede Art der Änderung wie neue Funktionen, Konfigurationsänderungen, Fehlerbehebungen oder Experimente automatisch testen zu lassen und an Nutzer auszuliefern. Regressionen zwischen verschiedenen Versionen werden schnell erkannt und können zeitnah behoben werden.

Neue Versionen können jederzeit bei Bedarf ausgeliefert werden. Dank automatisierter Tests ist der Code immer einsetzbar, sei es bei der Entwicklung in einem kleinen Team oder tausenden Entwickler in einem verteilten Open-Source-Projekt.

In der Praxis zeigt sich jedoch, dass die Automatisierung nicht die Lösung aller Probleme ist. Auf der Gegenseite zur Stärke bei Regressionstests steht die Schwäche bei änderungsanfälligen Produkten, besonders im frühen Stadium der Entwicklung. Das stetige Aktualisieren der Automatisierungsskripte kann den Aufwand des manuellen Testens auch mal übersteigen.

Aktuelle Forschungen beschäftigen sich mit dem Einsatz von künstlicher Intelligenz im Bereich des Softwaretestens. Basierend auf einer großen Menge an Trainingsdaten werden Modifikationen an bestehenden Elementen erkannt und Tests entsprechend angepasst. Beispielsweise wird bei Positionsänderungen von Schaltflächen in User-Interfaces bei nahegelegenen Elementen geprüft, ob diese der ursprünglichen Schaltfläche entsprechen.

Neben den Schwächen bei Änderungen fehlt den automatisierten Tests ein wichtiger Punkt: Menschliche Erfahrung. Besonders bei mobilen Applikationen, die uns heute immer und überall umgeben, ist die Benutzbarkeit der Knackpunkt, ob sie bei Nutzern gut ankommt. Ein Tester, der beim Testen eine Uhr in die Hand nimmt, benutzt sie so, wie er es beim Kauf machen würde. Dabei deckt er Probleme auf, die ein Skript übersehen hätte.

Abgesehen davon ist manuelles Testen sehr flexibel. Neue Ideen während der Entwicklung können sofort getestet werden, ohne dass die gesamte Automatisierung angepasst werden muss.

Manuelles Testen muss dort eingesetzt werden, wo seine Stärken liegen, beim Mensch. Aus diesem Grund sind die Schwerpunkte dieser Arbeit die Automatisierung von Teilen des manuellen Testprozesses, die Etablierung eines konsistenten und schnellen Feedbacksystems und die Reduzierung des Verwaltungsaufwandes für manuelle Tests.

2 Aufgabenstellung

2.1 Ziel der Arbeit

Diese Arbeit untersucht, ob die Verbesserung der Produktqualität und des Veröffentlichungsprozesses in der Entwicklung von Software mit Hilfe des Einbezugs einer Integration von manuellem Testen in die kontinuierliche Entwicklung möglich ist. Verbesserte Qualität spiegelt sich in weniger Defekten und Ausfällen wider, was wiederum eine höhere Benutzerzufriedenheit und -akzeptanz zur Folge hat.

Durch die Integration des manuellen Testens in die kontinuierliche Entwicklung können händische Tests organisiert durchgeführt und die oberflächlichen explorativen Tests ersetzt werden. Explorative Tests werden oft als einzige händische Testmethode verwendet. Sie bieten weder Nachvollziehbarkeit noch eine umfangreiche Testabdeckung. Das SystemTestPortal (STP) bietet als leichtgewichtige Webapplikation die Basis der Integration. Die zusätzliche Qualitätsuntersuchung kann als extra Instanz vor der Auslieferung einer neuen Produktversion an den Kunden eingesetzt werden.

Zudem soll die Integration die kontinuierliche Entwicklung und das manuelle Testen näher zusammenführen. Die Tester eines Systems sollen in den Entwicklungsprozess eingebunden werden, ohne einen erhöhten Kommunikationsaufwand zwischen Entwickler und Tester zu erzeugen.

Das soll jedoch nicht als Ersatz für automatisierte Tests angesehen werden, sondern als Möglichkeit, zusätzlich manuelle Tests in einem organisierten und nachvollziehbaren Arbeitsablauf durchzuführen. Zielführend für eine komfortable Lösung ist dabei unter anderem die automatische Ausführung von bestimmten Schritten während dem manuellen Testen. Das kann beispielsweise das Aktualisieren der Testsuite sein. Dadurch können der Zeitaufwand und die Kosten für manuelle Tests gesenkt werden.

2.2 Aufbau der Arbeit

In Kapitel 3 wird auf den aktuellen Stand der Technik eingegangen. Das beinhaltet bereits existierende Arbeiten und Systeme, die für die Umsetzung der Integration von manuellem Testen in das Continuous Deployment relevant sind.

Kapitel 4 bietet eine Übersicht über Funktionalität, Umsetzungsmöglichkeiten und zu beachtende Herausforderungen bei der Entwicklung. Diese werden aufgegriffen, detailliert erklärt und deren Lösungsansätze aufgezeigt.

Die entwickelte Lösung wird in Kapitel 5 vorgestellt. Darin sind die Umsetzung, der Ablauf der Integration und die Aufgaben der beteiligten Elemente enthalten.

Die entstandene Integration wurde evaluiert, um Mängel im Arbeitsfluss oder der Benutzbarkeit aufzudecken. Die Ergebnisse sind in Kapitel 6 vorgestellt.

Zur Abgrenzung des Arbeitsumfangs dient Kapitel 7. Dort sind die einzelnen Schritte während der Entwicklung der Arbeit ausgeführt.

Letztendlich werden mögliche Erweiterungsmöglichkeiten und weiterführende Themen zur Forschung in diesem Gebiet präsentiert.

3 Grundlagen und Stand der Technik

In diesem Kapitel werden verwandte Themen und Arbeiten, die im Zusammenhang mit dieser Arbeit stehen, analysiert. Damit wird ein Überblick über den aktuellen Stand der Technik gegeben.

3.1 Qualitätssicherung in der Ära der Continuous Deployment

Im Jahre 1994 wurde der Begriff Continuous Integration das erste Mal von Grady Booch in seinem Buch Object-Oriented Analysis and Design with Applications (2nd edition) [2] erwähnt. Damals war die Integration von ihm jedoch nicht als Teil des Build-Prozesses einer gesamten Applikation gemeint.

„... internal releases represent a sort of continuous integration of the system, and exist to force closure of the micro process.“

Booch [2, S.256]

Mit fortschreitender Zeit wurden Technologien ausgereifter und der Software-Code und die Business-Prozesse komplexer. Es war wichtig, dass nach jeder Änderung das gesamte Programm und die Softwareumgebung aufgebaut wurde, um funktionierende Applikationen zu garantieren. Folglich wurde die Continuous Integration (CI) für den Ansatz des Extreme Programmings sehr wichtig. Das Extreme Programming verfolgt den Ansatz von vielen, hochfrequenten Veröffentlichungen in kurzen Entwicklungszyklen. Dadurch sollen geänderte Kundenanforderungen schneller adaptiert werden können.

Basierend darauf entwickelte sich der Begriff der Continuous Delivery. Das Ziel dabei ist, den Code jederzeit veröffentlichen zu können, auch wenn man im Moment noch nichts veröffentlichen will. Das heißt nicht, dass ein Projekt zu 100 % fertig ist, sondern dass die eingebauten Funktionen geprüft und getestet sind. Es ist heute nicht denkbar, fehlerhafte Applikationen in die Öffentlichkeit zu geben. Das wäre der sichere Bankrott für jedes Unternehmen. Durch kurze Sprints können entstandene Fehler schneller entdeckt und behoben werden. In der Entwicklung entsteht dadurch eine stabile Codebasis.

Durch die ständig einsetzbare Applikation kann jede Änderung sofort automatisch dem Kunden ausgeliefert werden. Hierbei spricht man vom Continuous Deployment (CD). Diese Methodik bietet sich an, wenn der Endnutzer gleichzeitig auch als Tester agieren soll. Dessen Feedback wird genutzt, um die gefundenen Fehler in der nächsten Iteration zu beheben.

3.2 Das SystemTestPortal als Basis der Integration

Das SystemTestPortal [28] ist eine Webapplikation für den manuellen Systemtest [27]. Die Entwicklung der Anwendung wurde in einem Studienprojekt im Frühjahr 2017 an der Universität Stuttgart begonnen und ist unter der Open-Source-Lizenz GNU General Public License (GNU) v3.0 verfügbar [13].

Da sich diese Arbeit mit der Integration des SystemTestPortals in das Continuous Deployment beschäftigt, wird die Anwendung mit ihren wichtigsten Eigenschaften hier vorgestellt.

Das System befindet sich aktuell (Stand Sommersemester 2018) in aktiver Entwicklung durch das Team eines zweiten Studienprojekts. Durch den frühen Entwicklungsstand können sich manche Funktionalitäten nachträglich noch ändern.

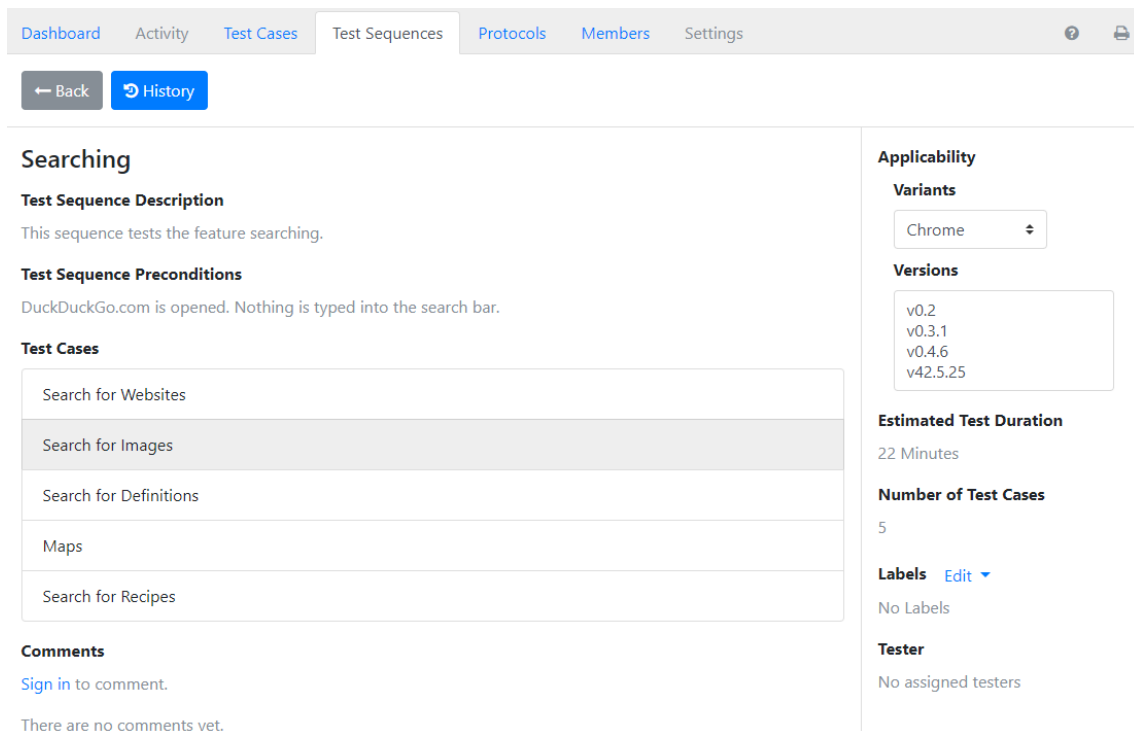
3.2.1 Testfälle und Testsequenzen

Die manuellen Tests werden im SystemTestPortal in Testfällen angelegt. Ein Testfall ist eine aus mehreren Schritten bestehende Anleitung zum Testen eines Anwendungsfalls des Testsystems. Abbildung 3.1 zeigt ein Beispiel eines Testfalls. Ein Testfall kann um weitere Informationen wie die voraussichtliche Testdauer, ein Label oder die System Under Test (SUT) Versionen, für die der Testfall anwendbar ist, ergänzt werden. Die System-Under-Test Versionen sind ein wichtiges Element für die Integration und werden in Abschnitt 3.2.3 näher beschrieben.

The screenshot displays the SystemTestPortal interface. At the top, there is a navigation bar with tabs for Dashboard, Activity, Test Cases, Test Sequences, Protocols, Members, and Settings. Below the navigation bar, there are buttons for 'Back' and 'History'. The main content area shows a test case titled 'Install DuckDuckGo.com for Chrome'. The test case description is 'Add DuckDuckGo.com as a search engine to your browser'. The test case preconditions are 'DuckDuckGo.com is opened'. The test steps are: 1. Click on the arrow down in the bottom center. 2. Click on the button 'Add DuckDuckGo to Chrome'. 3. Click 'Add extension'. 4. Press ALT + G. On the right side, there is a sidebar with 'Applicability' information, including 'Variants' (Chrome), 'Versions' (v0.2, v0.3.1, v0.4.6, v42.5.25), 'Estimated Test Duration' (8 Minutes), 'Number of Test Steps' (4), 'Labels' (High Priority), and 'Tester' (No assigned testers).

Abbildung 3.1: Detailansicht eines Testfalls

Um eine Reihe an verwandten Funktionen eines Systems zu testen, bietet das SystemTestPortal Testsequenzen an. Eine Testsequenz ist eine Reihe an Testfällen, die nacheinander ausgeführt werden. Informationen wie die voraussichtliche Testdauer oder die System-Under-Test Versionen werden automatisch aus den unterliegenden Testfällen berechnet. Abbildung 3.2 zeigt das Testen der Suchfunktionen für verschiedene Elemente in der Suchmaschine „DuckDuckGo“ [5].



The screenshot displays the 'Test Sequences' view in the SystemTestPortal. The navigation bar at the top includes 'Dashboard', 'Activity', 'Test Cases', 'Test Sequences', 'Protocols', 'Members', and 'Settings'. Below the navigation bar are 'Back' and 'History' buttons. The main content area is titled 'Searching' and contains the following sections:

- Test Sequence Description:** This sequence tests the feature searching.
- Test Sequence Preconditions:** DuckDuckGo.com is opened. Nothing is typed into the search bar.
- Test Cases:** A list of five search actions: Search for Websites, Search for Images, Search for Definitions, Maps, and Search for Recipes.
- Comments:** A link to 'Sign in to comment' and a note that there are no comments yet.

The right sidebar provides additional details under the 'Applicability' section:

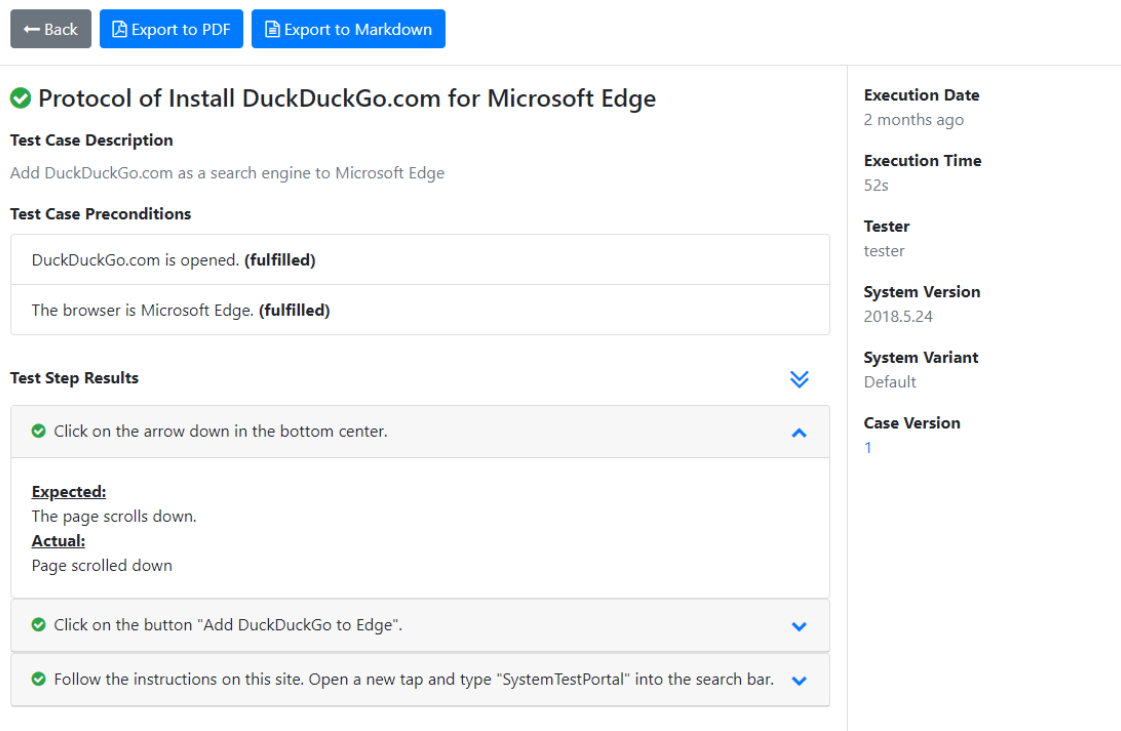
- Variants:** Chrome
- Versions:** v0.2, v0.3.1, v0.4.6, v42.5.25
- Estimated Test Duration:** 22 Minutes
- Number of Test Cases:** 5
- Labels:** No Labels (with an 'Edit' dropdown)
- Tester:** No assigned testers

Abbildung 3.2: Detailansicht einer Testsequenz

3.2.2 Testprotokolle

Die Ergebnisse der einzelnen Testausführungen sind in der Datenbank hinterlegt und können in den Protokollen abgerufen werden. Die gespeicherten Informationen umfassen:

- Der ausgeführte Test (mit der Version des Tests)
- Endresultat der Ausführung
- Resultat der einzelnen Schritte
- Ausführungsdatum
- Die Ausführungsdauer
- Der Tester
- Die Version des Testsystems
- Die Variante des Testsystems



← Back Export to PDF Export to Markdown

✓ Protocol of Install DuckDuckGo.com for Microsoft Edge

Test Case Description
Add DuckDuckGo.com as a search engine to Microsoft Edge

Test Case Preconditions

- DuckDuckGo.com is opened. **(fulfilled)**
- The browser is Microsoft Edge. **(fulfilled)**

Test Step Results ⌵

- ✓ Click on the arrow down in the bottom center. ⬆
Expected:
The page scrolls down.
Actual:
Page scrolled down
- ✓ Click on the button "Add DuckDuckGo to Edge". ⌵
- ✓ Follow the instructions on this site. Open a new tap and type "SystemTestPortal" into the search bar. ⌵

Execution Date
2 months ago

Execution Time
52s

Tester
tester

System Version
2018.5.24

System Variant
Default

Case Version
1

Abbildung 3.3: Protokoll des Testfalls „Install DuckDuckGo.com for Microsoft Edge“

Abbildung 3.3 zeigt ein Beispiel eines Testprotokolls. Darauf sind die Ergebnisse der Ausführung für die Installation der Suchmaschine DuckDuckGo für Microsoft Edge gegeben.

Die Testprotokolle ermöglichen eine nachvollziehbare Testausführung. Das erleichtert das Finden von fehlerhaften Elementen in einem Produkt.

Für die Analyse des Testsystems in der Continuous Integration müssen diese Protokolle ausgewertet werden. Wie dies gemacht wird, ist in Kapitel 4.3.2 näher erklärt.

3.2.3 Versionierung des Testsystems

Versionen

Während der Entwicklung entstehen in zeitlichen Abständen verschiedene Ausführungen eines Produktes. Diese Ausführungen unterscheiden sich beispielsweise im Umfang der Funktionalitäten, im Design oder anderen Elementen. Sie werden als Versionen bezeichnet und sind oftmals durch fortlaufende Versionsnamen gekennzeichnet. Dadurch ist bei einem Vergleich klar, welche Versionen neuer und dementsprechend fortgeschrittener sind.

Varianten

Zusätzlich zu den Versionen gibt es den Begriff der Varianten. Eventuell möchte ein Kunde bei einem Produkt eine personalisierte Ausführung, die nicht dem Grundprodukt entspricht. Bei weltweit eingesetzten Produkten können Varianten z.B. Anpassungen an die Standards der einzelnen Länder aufgrund von unterschiedlichen Spannungsanforderungen im Stromnetz sein.

Versionen und Varianten im SystemTestPortal

Das SystemTestPortal bietet die Möglichkeit der Abbildung dieser beiden Begriffe, um die Anwendbarkeit von Testfällen und Testsequenzen klar zu definieren. Bei der Erstellung oder beim Editieren von Testfällen kann ausgewählt werden, auf welche Versionen und Varianten ein Testfall anwendbar ist. Bei fehlgeschlagenen Ausführungen ermöglicht das eine transparente und konsequente Rückverfolgung.

Die Handhabung der Versionierung im SystemTestPortal wird aktuell noch diskutiert und die aktuelle Darstellung und Funktionsweise können sich ändern. Bei der Verwaltung von Versionen und Varianten muss trotz einfacher Bedienbarkeit eine gute Flexibilität gegeben sein, da nicht alle Produkte ein einheitliches Versionierungsschema verwenden. Von Zahlenfolgen wie „v1.1.0“ bis ausgeschriebenen Namen wie „Supercar-Version-One“ kann das Schema frei gewählt werden. Herausforderungen wie diese sind der Grund für die schwierige Umsetzung des Themas.

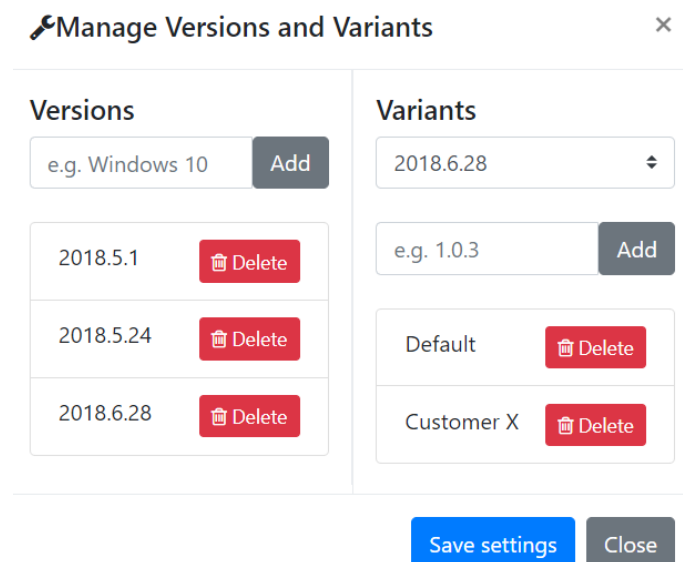


Abbildung 3.4: Dialog zur Verwaltung der Versionen und Varianten eines Testsystems

Abbildung 3.4 zeigt den Dialog, in dem die Versionen und Varianten des Testsystems erstellt und gelöscht werden können. Links sind die Versionen in einer Liste dargestellt. Gibt es nun für eine bestimmte Version eine andere Ausführung als die Standardvariante, kann im Dropdown-Menü auf der rechten Seite die Version ausgewählt werden und darunter die Variante erstellt werden.

Für Personen, die mit dem Zusammenhang zwischen Versionen und Varianten nicht vertraut sind, ist diese Darstellung verwirrend. Eine Alternative dazu wäre eine Matrixdarstellung, wobei die x-Achse die Versionen in zeitlich geordneter Reihenfolge und die y-Achse die verschiedenen Varianten sind. In der Matrix kann nun durch einfaches Klicken auf eine Zelle ausgewählt werden, dass für eine Version auch eine Variante existiert.

3.3 Integrationen weiterer Testverwaltungsprogramme

Neben dem SystemTestPortal gibt es auch andere Applikationen zum Verwalten von manuellen Tests. Außer den „Standardfunktionalitäten“ eines Test-Management-Tools bieten diese oft eine Integration mit Application-Lifecycle-Management-Tools wie Jira oder automatischen Test-Tools wie Selenium. Eine Integration in den automatischen Deploymentprozess wie in dieser Arbeit ist bei keinem der beliebtesten Tools zu finden [29]. Die Software qTest beispielsweise bietet das Analysieren und Anzeigen der Testergebnisse eines Builds in Jenkins, jedoch nicht umgekehrt in dem Sinne, dass das CI/CD-System die Testergebnisse des Test-Management-Tools verarbeitet.

Die Integration von qTest mit Jira ist sehr interessant, da für jedes Issue in Jira eine Testabdeckung im Test-Management-Tool verfügbar ist. Dadurch können Entwickler sehen, wie viele der Anforderungen erfüllt sind. Das erleichtert die Planung der nächsten Entwicklungsabschnitte.

Eine weitere Funktionalität zum Verbinden von Entwicklung und manuellem Testen ist das Erstellen eines Issues in Jira, sobald ein Test fehlschlägt. Den aufgetretenen Fehler können die Entwickler in der Planung berücksichtigen.

3.4 Systeme zur Verwaltung des DevOps-Lebenszyklus

Das Angebot an Systemen zur Continuous Integration ist groß. Die Beschreibung von [3] bietet einen Überblick über verschiedene Systeme. Im Rahmen dieser Arbeit ist es jedoch sinnvoll, sich auf ein oder wenige CI- und CD-Systeme zu konzentrieren. Beliebte Systeme sind Jenkins, Travis CI oder CircleCI. Das Entwicklerteam des SystemTestPortals arbeitet ausschließlich mit der GitLab CI. Deswegen und da das SystemTestPortal aktuell noch einen sehr kleinen Anwenderkreis besitzt, ist eine Auslegung der Integration fürs Erste in die GitLab CI von Vorteil.

Das Team des SystemTestPortals kann die Integration selbst nutzen, um ihr System zu testen. Diese Praktik wird als „Eating your own dog food“ bezeichnet [1]. Genauer gesagt heißt das, dass etwas erstellt wurde, was noch nicht fertig ist um von externen Personen konsumiert zu werden. Somit muss man es erst selbst probieren.

Für das CI-System Jenkins gibt es unzählige Plugins [16]. Im Bereich des manuellen Testens ist die Auswahl jedoch beschränkt. Die vorhandenen Plugins konzentrieren sich ausschließlich auf das Ausführen und Analysieren von automatisierten Tests.

3.4.1 GitLab

GitLab ist eine Webapplikation zur Versionsverwaltung von Softwareprojekten basierend auf Git. Es war anfangs als Plattform zur Verwaltung von Source-Code und für die Kollaboration von Teammitgliedern während der Entwicklung gedacht. Mit der Zeit entwickelte GitLab sich zu einer Lösung für den Software Development Life Cycle und des gesamten Development and Operations (DevOps)-Lebenszyklus.

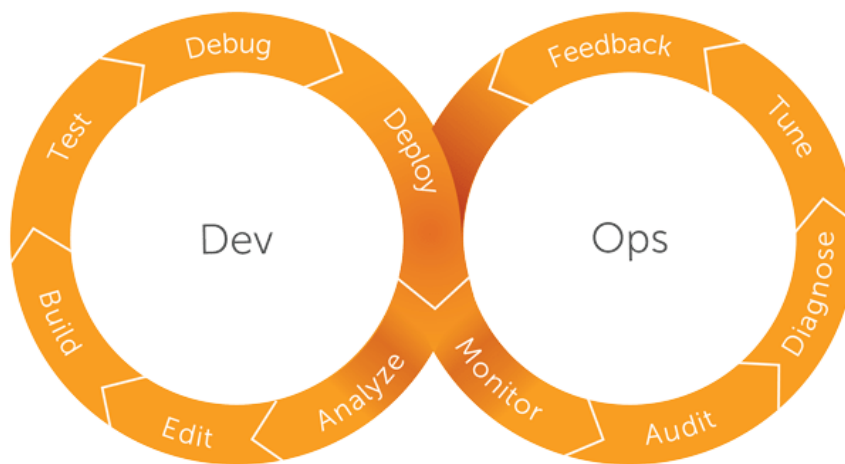


Abbildung 3.5: Alle Schritte des DevOps-Lebenszyklus [19]

Der DevOps-Lebenszyklus, wie er in Abbildung 3.5 dargestellt ist, umfasst alle Schritte der Entwicklung und Wartung eines Softwareproduktes. Neben dem Schreiben und Testen von Code ist dort auch die Überwachung, Analyse und das Feedback während des Einsatzes inbegriffen.

GitLab bietet integrierte CI/CD-Pipelines zum Bauen, Testen, Ausliefern und Überwachen des Codes an. Eine Pipeline wird nach jeder Neuerung des Codes ausgelöst. Sie ist aufgeteilt in mehrere Stages. Die Stages sind aufgeteilt nach funktionalen Aufgaben und geordnet nach den jeweiligen Abhängigkeiten. Je nach Konfiguration kann eine Pipeline unterschiedlich umfangreich aufgebaut sein. Eine typische Pipeline umfasst die Stages Build, Test, Staging und das Ausliefern in die Produktion (Abbildung 3.6).

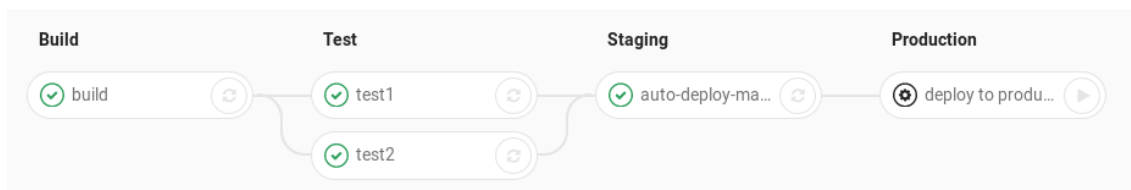


Abbildung 3.6: Typische Stages in einer Pipeline [12]

3.4.2 Manuelles Testen im Continuous Deployment

Das Thema des manuellen Testens im Continuous Deployment ist nur sehr gering verbreitet. Der Grund ist, dass manuelles Testen einen enorm hohen zeitlichen Aufwand hat und sich somit nicht in den automatisierten Arbeitsfluss einfügt.

Heutzutage gibt es die Möglichkeit, während der Continuous Integration eine Applikation automatisch in einer Testumgebung zu installieren. GitLab beispielsweise bietet Review Apps an, um in dynamisch angelegten Umgebungen eine Applikation manuell zu testen [10]. Diese Funktion wird aktuell (Stand Sommersemester 2018) in der Entwicklung des SystemTestPortals bereits eingesetzt [27]. Neue Funktionen können dadurch getestet werden, bevor sie aus der Entwicklung in die Produktion kommen. Das Herunterladen und Starten in einer lokalen Umgebung entfällt somit. Zusätzliche Werkzeuge werden auf dem Rechner nicht benötigt, da der Zugang über den Browser erfolgt. Die Umgebungen, in denen die Applikation bereit gestellt wird, werden automatisch über die Continuous Integration-Pipeline von GitLab erstellt, wenn alle vorherigen Tests erfolgreich waren.

Die Review Apps alleine bieten jedoch keine Möglichkeit, manuelle Tests geordnet und nachvollziehbar durchzuführen. Dafür gibt es Applikationen wie das SystemTestPortal. Sie ermöglichen das Organisieren von manuellen Tests.

3.4.3 Anpassung des Integrations- und Auslieferungsprozesses durch Webhooks

Bestehende Applikationen, die in der Continuous Integration verwendet werden, kommunizieren über eine von der jeweiligen Applikation bereitgestellten Application Programming Interface (API). CI-Systeme stellen Anfragen an diese APIs über Webhooks [33]. Webhooks sind HTTP POST Callbacks, also einfache Benachrichtigungen bei bestimmten Events. Beispiele für Webhooks sind in GitHub [30], GitLab [31] oder Jenkins [8] zu finden.

Die Integration des STP in die Continuous über ein Webhook würde eine große Bandbreite an unterstützten CI-Systemen ermöglichen. Andere Systeme die fürs Testen eingesetzt werden, sind ebenso über eine API eingebunden. Beispiele dafür sind TestLink oder eXtensive Testing.

3.5 Metriken und Feedback

Ein konsistentes und transparentes Feedback ist wichtig für den Nutzer. Das Feedback kann beliebig gestaltet werden. Es muss jedoch darauf geachtet werden, dass das Feedback für jede Änderung zurückgeliefert wird. Außerdem muss es so früh wie möglich an die korrekte Zielgruppe geliefert werden [17].

Klar definierte Metriken ermöglichen es den Entwicklern, ein zu testendes Produkt effizienter zu verbessern, da sie eine quantitative Bewertung bezüglich der Eigenschaften des Produkts bekommen.

Mögliche Metriken für eine erfolgreich getestete Version können

- Totale Anzahl ausgeführter Tests
- Totale Anzahl bestandener Tests
- Totale Anzahl fehlgeschlagener Tests
- Prozentuale Anzahl ausgeführter Tests
- Prozentuale Anzahl bestandener Tests
- Anzahl an kritischen Fehlschlägen
- Anzahl an nicht-kritischen Fehlschlägen
- Kosten für die Ausführung der Tests (Zeitliche Kosten)

sein [14].

Die Phasen des Feedbacks teilen sich ein in die Analyse der Metriken, also welche Metriken sollen verwendet werden, die Evaluation der vorhandenen Daten und der Metriken und zuletzt die Kommunikation der Ergebnisse mit der Zielgruppe des Feedbacks [25].

Die Übermittlung der Ergebnisse ist ein wichtiger Schritt. Je nachdem, wie das Feedback bereitgestellt wird, können die Entwickler schneller oder langsamer darauf reagieren. Das Senden einer E-Mail bei fehlgeschlagenen Tests erzielt schneller eine Wirkung als ein kleines Symbol, das irgendwo auf einer Webseite angezeigt wird.

Das Feedback sollte jedoch nicht an alle gesendet werden, sondern nur an die Gruppe, die es betrifft. Ansonsten beginnt jeder irgendwann die Nachrichten zu ignorieren [21]. Des Weiteren muss das Feedback zeitig gesendet werden. Je mehr Zeit zwischen der Einführung des Fehlers und der Rückmeldung vergeht, desto weniger beteiligte Projektmitglieder erinnern sich daran was passiert ist. Es kann vorkommen, dass der Fehler bereits behoben ist und eine „Geisterjagd“ nach Fehlern, die nicht mehr da sind, beginnt.

3.6 Anwendung der Continuous Practices in der Praxis

Continuous Integration, Continuous Delivery und Continuous Deployment, die sogenannten Continuous Practices, werden von Entwicklerteams in Firmen aus verschiedenen Gründen gemieden. In [24] wurden diese Gründe klassifiziert und analysiert. Dazu gehören unter anderem die geringe Qualität der Tests und das Fehlen einer angemessenen Teststrategie, sowohl für automatische als auch für manuelle Tests. Im Weiteren wurde herausgefunden, dass es für einen erfolgreichen Prozess hilfreich ist, verschiedene Rollen in einem Team zu etablieren. Durch die Zusammenarbeit zwischen dem Quality Assurance (QA)-Team und den Entwicklern erhöht sich die Qualität der Software.

Eine Einbindung des Kunden in den kontinuierlichen Entwicklungsprozess hat für das Entwicklerteam den Vorteil, Feedback von der Person zu erhalten, für die sie das System entwickeln. Fehlerhafte Entscheidungen können somit frühzeitig entdeckt werden. Wie in [6] vorgeschlagen, können Kunden im SystemTestPortal ihre Akzeptanztests formulieren, welche wiederholt getestet werden können.

Die Zuständigkeit bei fehlschlagenden Tests ist unterschiedlich. Manchmal ist der Entwickler, der nach dem letzten funktionierenden Stand eingecheckt hat, für die Korrektur des Fehlers zuständig. In anderen Fällen ist der Entwickler, der zuletzt Code eingecheckt hat, dafür zuständig den Fehler zu beheben oder die Änderungen zurückzusetzen. Ein weiterer Ansatz ist, dass nach einer kurzen Untersuchung des Fehlers ein Entwickler diesen behebt, während die anderen Entwickler mit ihrer Arbeit fortfahren [26].

4 Theoretische Grundlagen der Integration

In diesem Kapitel wird der Entwurf zur späteren Umsetzung beschrieben. Die umzusetzenden Funktionalitäten werden gezeigt, die Umsetzung an sich wird evaluiert und Entscheidungen, Probleme und mögliche Lösungsansätze werden erklärt.

4.1 Anwendungsfälle der Integration

Mit Hilfe von Anwendungsfällen kann übersichtlich gezeigt werden, für welche Aktionen die Integration hilfreich ist und wie diese ablaufen. Darin sind beteiligte Personen, ihr Ziel und die Bedingungen der Aktion aufgezeigt. Abbildung 4.1 zeigt alle Anwendungsfälle für die Integration, die in dieser Arbeit umgesetzt werden. Mögliche Erweiterungen sind aus Gründen der Übersicht nicht berücksichtigt.

In diesem Diagramm sind die drei beteiligten Systeme gekennzeichnet. Auf der linken Seite ist das CI/CD-System mit dem Entwickler. Gegenüber ist das SystemTestPortal mit dem Tester für die manuellen Tests. Die Integration bildet die Brücke zwischen den Systemen und verknüpft die Entwicklung im CI/CD-System mit dem Testvorgang im SystemTestPortal. Hierbei ist hervorzuheben, dass die Personen ausschließlich mit den ihnen vertrauten Systemen arbeiten. Sie benötigen keinen Zugriff und Einarbeitung in andere Umgebungen.

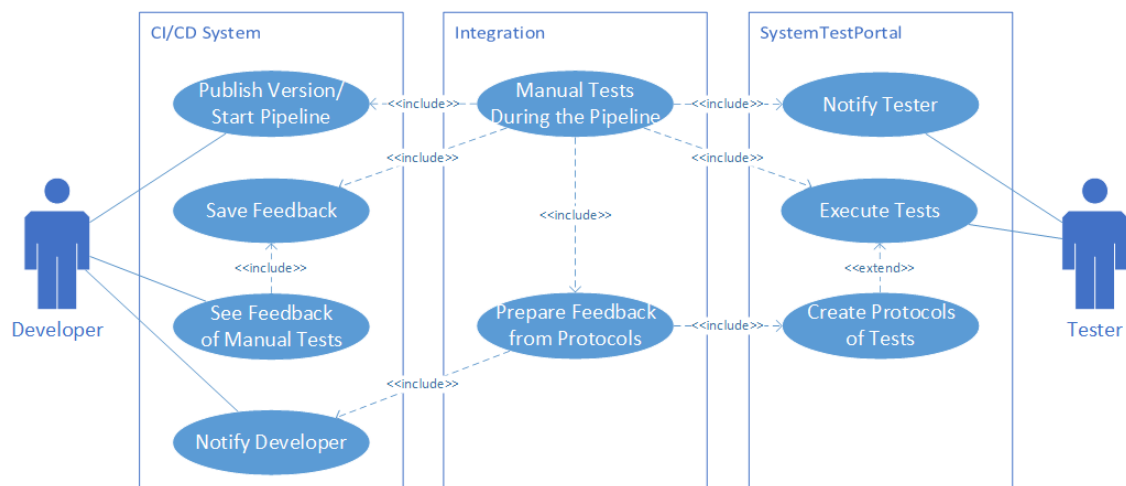


Abbildung 4.1: Alle Anwendungsfälle der Integration im Überblick

Manuelles Testen einer neuen Version während der Ausführung der CI/CD-Pipeline

Der primäre Anwendungsfall zur Integration des manuellen Testens in die kontinuierliche Entwicklung ist das manuelle Testen während der Ausführung der Pipeline. Im Diagramm 4.1 ist dies der zentral dargestellte Anwendungsfall, der andere Anwendungsfälle enthält und CI/CD-System und das SystemTestPortal verbindet.

Information	Beschreibung
Primärer Akteur	Entwickler, Tester
Ziel	Der Tester möchte während der Ausführung der Pipeline die manuellen Tests ausführen
Vorbedingungen	<ul style="list-style-type: none"> Das CI/CD-System ist korrekt konfiguriert
Nachbedingungen	<ul style="list-style-type: none"> Die Ergebnisse der Tests können abgerufen werden Die Pipeline wird je nach Testausgang fortgesetzt oder abgebrochen
Erfolgreicher Ablauf	<ol style="list-style-type: none"> Der Entwickler veröffentlicht eine neue Systemversion Das CI/CD-System startet die CI/CD-Pipeline Das CI/CD-System sendet an das SystemTestPortal eine Anfrage, die neue Version zu testen Das SystemTestPortal erstellt automatisch die zu testende Version Das SystemTestPortal erstellt automatisch eine Aufgabe für den Tester/benachrichtigt den Tester dass er diese Version testen soll Der Tester führt die Tests aus Das System meldet an das CI/CD-System, ob die Tests bestanden wurden
Erweiterungen	<p>[3.a] Die Version existiert bereits. Dann wird keine Version erstellt</p> <p>[5.a] Im SystemTestPortal wird evaluiert, welche Tests ausgeführt werden sollen. Dies kann automatisch oder manuell durch eine Rückmeldung eines Test-Managers geschehen</p>

Tabelle 4.1: Manuelles Testen einer neuen Version während der Ausführung der CI/CD-Pipeline

Ergebnisse der manuellen Tests einer Pipeline abrufen

Die agile Entwicklung setzt auf schnelles Feedback zu Fehlern im Programm. Um die Ursache von Fehlern schnell und zuverlässig finden zu können, ist ein möglichst detailliertes Feedback für die Entwickler nötig. Da diese nicht immer Zugriff auf das SystemTestPortal haben, sollten das Feedback im CI/CD-System verfügbar sein, sobald die Ergebnisse der Tests bereit stehen.

Information	Beschreibung
Primärer Akteur	Entwickler
Ziel	Der Akteur möchte die Protokolle der Tests im CI/CD-System abrufen
Vorbedingungen	<ul style="list-style-type: none"> • Die manuellen Tests wurden bereits ausgeführt • Die CI/CD-Pipeline wurde bereits ausgeführt • Das von der Integration vorbereitete Feedback wurde vom CI/CD-System gespeichert
Nachbedingungen	<ul style="list-style-type: none"> • Der Akteur kann alle Protokolle zur gewünschten Version einsehen
Erfolgreicher Ablauf	<ol style="list-style-type: none"> 1. Der Akteur ruft die Pipeline der gewünschten Version auf. Dies kann sich je nach CI/CD-System unterscheiden 2. Der Akteur ruft die Protokolle der Tests, die für diese Pipeline hinterlegt sind, ab. Wie diese Protokolle hinterlegt sind, kann sich je nach CI/CD-System unterscheiden

Tabelle 4.2: Ergebnisse der manuellen Tests einer Pipeline abrufen

Benachrichtigung der Entwickler bei fehlgeschlagenen Tests

Zusätzlich zu den Ergebnissen einer Pipeline für eine neue Version ist für die Entwickler auch das Protokoll einer älteren Version wichtig, wenn der Test fehlschlug. Damit Tester nicht selbst mit den Entwicklern Kontakt aufnehmen müssen, ist das automatische Benachrichtigen eine hilfreiche Funktion in der Integration.

Information	Beschreibung
Primärer Akteur	Tester, Entwickler
Ziel	Der Tester möchte die Entwickler automatisch bei einem fehlgeschlagenen Test benachrichtigen
Vorbedingungen	<ul style="list-style-type: none"> Die Benachrichtigungen sind aktiviert
Nachbedingungen	<ul style="list-style-type: none"> Die Entwickler wurden über den fehlgeschlagenen Test benachrichtigt
Erfolgreicher Ablauf	<ol style="list-style-type: none"> Der Tester führt den Test aus und bewertet ihn als fehlgeschlagen Das SystemTestPortal sendet an die Entwickler eine Benachrichtigung mit allen wichtigen Details
Erweiterungen	[2.a] Die Entwickler wurden bereits benachrichtigt. Dann soll keine weitere Benachrichtigung gesendet werden, um Duplikate zu vermeiden

Tabelle 4.3: Benachrichtigung der Entwickler bei fehlgeschlagenen Tests

4.2 Umsetzungsmöglichkeiten der Integration

Das SystemTestPortal lässt sich auf verschiedene Arten in die CI/CD einbinden. Eine übliche Schnittstelle um externe Systeme in die CI/CD einzubinden sind die in Abschnitt 3.4.3 vorgestellten Webhooks. Eine andere Möglichkeit ist das Erstellen und Konfigurieren einer eigenen Stage in der Pipeline, welche für das manuelle Testen zuständig ist [15]. Dies ist aufwändiger, bietet jedoch größere Flexibilität.

4.2.1 Integration über Webhooks

Die Integration des SystemTestPortals in die Continuous Integration über Webhooks ist für den Endnutzer nur mit geringem Aufwand verbunden. Im CI-System wird eine Uniform Resource Locator (URL) definiert, an welche ein HTTP POST-Request gesendet wird.

Das HyperText Transfer Protocol (HTTP) Protokoll wird verwendet, um die Kommunikation zwischen Clients und Servern zu ermöglichen. Das CI-System agiert hier als Client und das STP als Server. Es gibt verschiedene Methoden im HyperText Transfer Protocol, die für unterschiedliche Anwendungsfälle eingesetzt werden. Ein „GET“-Request beispielsweise wird verwendet, um Daten vom Server anzufordern. Der „POST“-Request dient dazu, Daten an den Server zu schicken und Ressourcen zu ändern oder zu erstellen. Im Vergleich

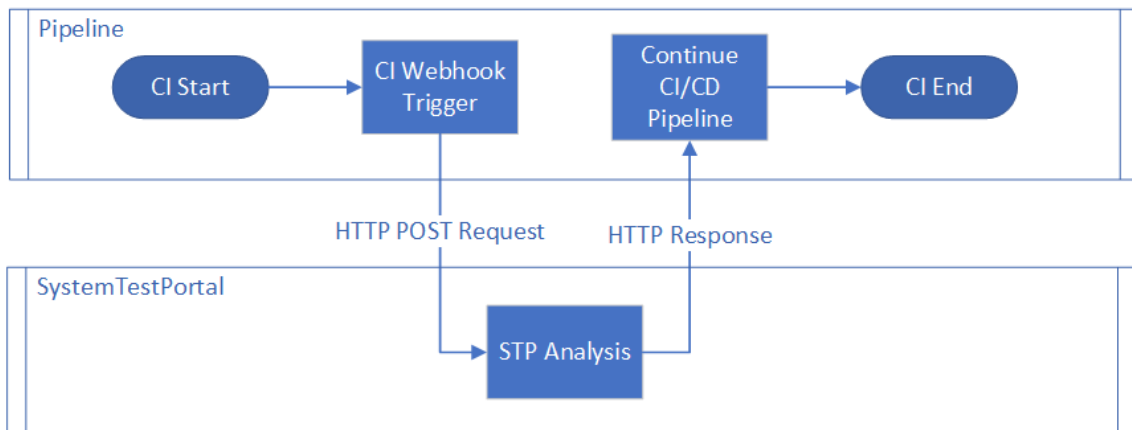


Abbildung 4.2: Ablauf der Integration über Webhooks

zum GET-Request kann ein POST-Request Daten in seinem „Message Body“ transportieren. Die POST-Requests der Webhooks in GitLab stellen darin z.B. Informationen zu geänderten Daten zur Verfügung.

Wenn die Anfrage beim SystemTestPortal eintrifft, analysiert dieses die Ergebnisse der Tests für die entsprechende Version und sendet einen HTTP-Response zurück. Daraufhin wird die Pipeline für die Integration und das Deployment weiter ausgeführt (Abbildung 4.2).

Webhooks können bei verschiedenen Ereignissen ausgelöst werden. GitLab bietet Webhooks für

- Push Ereignisse
- Tag Ereignisse
- Issues Ereignisse
- Comments Ereignisse
- Merge Requests Ereignisse
- Job Ereignisse
- Pipeline Ereignisse
- Wiki Page Ereignisse.

Ausschließlich Push-, Tag- oder Merge Request- Ereignisse werden bei Änderungen am Code und somit bei einer neuen Version ausgeführt. Das bedeutet konkret, dass nur manuelle Tests ausgeführt werden, wenn eines dieser drei Ereignisse eintritt. Im SystemTestPortal muss zwischen diesen Ereignissen unterschieden werden, da je nach Ereignis andere Informationen aus dem POST-Request gelesen werden müssen.

Ob es in der Praxis sinnvoll ist, für jedes einzelne Push-Ereignis manuelle Tests auszuführen, ist fraglich. Ein Push-Ereignis ist das „Verschieben“ des Codes vom lokalen Rechner eines Entwicklers auf das öffentliche Code-Repository, auf das alle Entwickler Zugriff haben. Wichtiger sind hierbei die Tag- und Merge-Request-Ereignisse, da diese größere Änderungen an einem System beinhalten.

Webhooks sind für Entwickler einfach zu integrieren, jedoch kann ein entscheidendes Problem nicht gelöst werden. Um eine neue Version manuell testen zu können, muss diese irgendwo für die Tester zugänglich sein. Ein Webhook wird aber direkt nach Veröffentlichung einer neuen Version ausgeführt und erwartet, dass die Testergebnisse zur Verfügung stehen. Das ist praktisch nicht möglich.

4.2.2 Integration über eine gesonderte Stage in der Pipeline

Für diese Art der Umsetzung wird in der CI/CD-Pipeline des zu integrierenden Projektes ein eigenständiger Abschnitt definiert, der für das manuelle Testen zuständig ist. In der Pipeline spricht man dabei von einer Stage. Weitere Informationen zu den Stages sind in Abschnitt 3.4.1 zu finden.

Die Stage für das manuelle Testen sollte zwischen dem Deployment in eine Testumgebung und der endgültigen Auslieferung zum Kunden definiert werden. Dadurch ist das manuelle Testen nochmals eine weitere Stufe der Qualitätssicherung. Zur Erzeugung der Testumgebung können Review Apps verwendet werden [23]. Sie ermöglichen eine automatische Generierung einer Live-Vorschau zu Codeänderungen. Dadurch müssen Designer und Produktmanager die Änderungen nicht mehr auf ihren Rechner übertragen und dort lokal testen.

In der definierten Stage muss das CI/CD-System mit dem SystemTestPortal interagieren um Befehle und Testergebnisse auszutauschen. Je nach Ausgang der Tests reagiert die Stage dementsprechend und schließt entweder erfolgreich ab oder schlägt fehl.

Im Gegensatz zur Umsetzung mit Webhooks ist es hierbei möglich, zuerst automatisiert eine Live-Testumgebung zu generieren, welche von den Testern verwendet wird. Die Testumgebung sollte der Produktionsumgebung ähneln, um Fehler aufzudecken, die auch in der Anwendung auftreten. Durch die Auslagerung in einen eigenen Abschnitt in der Pipeline werden nicht im gleichen Schritt die Version veröffentlicht und die Testergebnisse müssen schon bereit stehen.

Die lange Ausführungszeit der manuellen Tests wirkt sich auch auf die Laufzeit der CI/CD-Pipeline aus. Sie wird so lange ausgeführt, bis die Testergebnisse vorhanden sind oder eine Zeitüberschreitung auftritt. Die Minimierung der Testzeit ist hierbei jedoch Aufgabe der verantwortlichen Projektleiter und Tester.

Aufgrund der langen Laufzeit ist es sinnvoll, die manuellen Tests nur bei wichtigen Releases anzuwenden. Die lange Ausführungsdauer der Pipeline wird von manchen Personen eventuell als Störung im Arbeitsfluss empfunden. Außerdem entstehen weitere Kosten, wenn jede neue Version, auch wenn sie nur wenige Änderungen hat, aufs neue manuell getestet werden muss.

4.3 Herausforderungen in der Umsetzung

4.3.1 Zuordnung der Versionen und Varianten des Testsystems

Zwischen den Versionen eines Testsystems und den Versionen im SystemTestPortal muss eine eindeutige Beziehung hergestellt werden. Das SystemTestPortal bietet dafür System-Under-Test-Versionen. Damit festgelegt ist, welcher Testfall überhaupt bei einer Version ausgeführt werden kann, muss die Anwendbarkeit der Testfälle definiert werden. Dadurch wissen die Projektmitglieder, dass z.B. der Testfall „Änderung der Sprache“ nur ab Version „2.0“ getestet werden kann, da es vor dieser Version die Option, die Sprache zu ändern, noch nicht gab.

Analog zu den Versionen verhalten sich die Varianten eines Testsystems. Ein Testfall kann beispielsweise nur auf das Produkt für den Kunden XY angewendet werden, da sich dieser Kunde eine spezielle Funktion gewünscht hat. Die Standardvariante besitzt die Funktion jedoch nicht. Somit ist der Testfall bei der Ausführung auf die spezielle Kundenvariante beschränkt.

Für die Analyse der Tests wird die zu testende Version und Variante benötigt. Diese müssen vom CI/CD-System bereitgestellt werden. Je nach Umsetzungsart unterscheidet sich die Bereitstellung.

Versionen und Varianten bei Webhooks

Die Version des Testsystems wird bei den Webhooks über den ausgelösten HTTP POST-Request gesendet. Eine Variante kann hierbei nicht abgerufen werden. Diese muss im SystemTestPortal vordefiniert sein und kann auch nicht dynamisch je nach Request angepasst werden.

Ein Beispiel für den Request-Body bei einem Push-Ereignis von GitLab [31] ist in 4.1 zu sehen. Einige irrelevante Informationen wurden aufgrund der Länge entfernt. Aus dem Parameter „checkout_sha“ kann die zu analysierende Version ausgelesen werden. Der Aufbau der Message-Bodies von Merge-Request-Events oder Tag-Events unterscheidet sich zu dem der Push-Events. Je nach Typ des Events müssen somit andere Parameter ausgelesen werden. Bei einem Merge-Request-Event wird der Name des Merge-Requests ausgelesen, bei einem Tag-Event der Name des Tags.

Die Versionen im SystemTestPortal müssen somit für eine eindeutige Zuordnung wie im Folgenden beschrieben benannt werden:

- Push-Events:
Der Commit-Hash des letzten Commits, der gepusht wird.
- Merge-Request-Events:
Der Name des Merge-Requests.
- Tag-Events:
Der Name des Tags. Für einen Tag mit dem Namen „v1.0.0“ wird somit eine Version „v1.0.0“ im SystemTestPortal angelegt.

Listing 4.1 Beispiel für den Message-Body eines Webhook-Requests

```
{
  "object_kind": "push",
  "before": "95790 bf891e76fee5e1747ab589903a6a1f80f22 ",
  "after": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
  "ref": "refs/heads/master",
  "checkout_sha": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
  "user_id": 4,
  "user_name": "John Smith",
  "user_username": "jsmith",
  ...
},
"repository": {
  ...
},
"commits": [
  {...},
  {...}
],
"total_commits_count": 4
}
```

Durch Abgleichen der gegebenen Version und der Version im SystemTestPortal werden nur die auf diese Version anwendbaren Tests analysiert.

Versionen und Varianten bei einer Webapplikation

Analog zu den Versionen und Varianten bei Webhooks, können sie bei Verwendung eines eigenen Webapplikation auch in JavaScript Object Notation (JSON) im Body der Anfrage gesendet werden. Die Version wird auch durch das CI-System vorgegeben. Die Variante des zu testenden Systems muss vor der Ausführung der Pipelines konfiguriert werden. Damit ist die Variante jedoch für jede Ausführung gleich. Manche CI-Systeme erlauben das Definieren von Variablen für eine spezifische Ausführung der Pipeline. Das ist ein Vorteil gegenüber den Webhooks.

4.3.2 Analyse der Tests

Wie wird festgelegt, welche Tests analysiert werden?

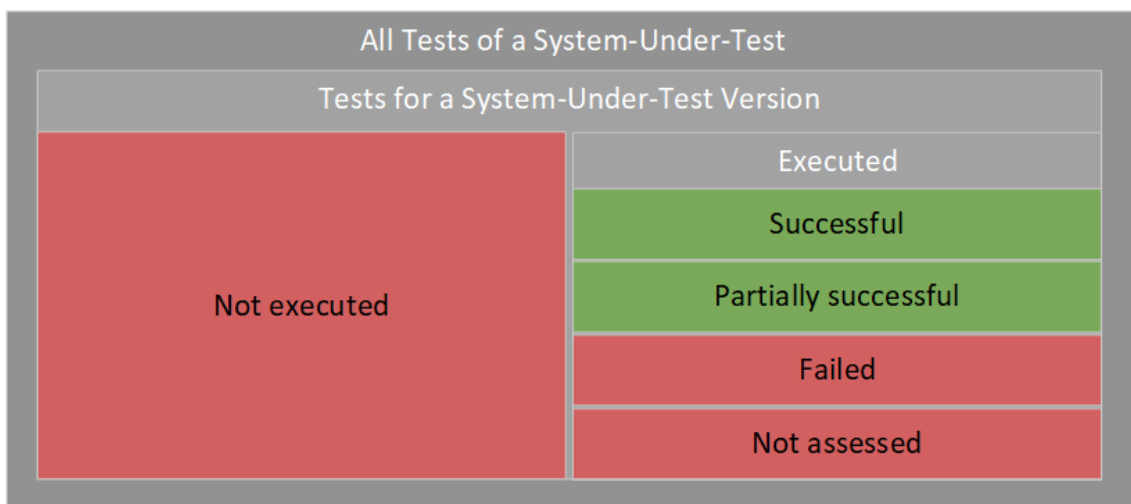
Da es, besonders in großen Systemen, unnötig wäre, alle Testfälle für jede neue Version des Testsystems auszuführen, ist eine Evaluation, welche Tests ausgeführt werden müssen, sinnvoll. Zusätzlich dazu sind einige Testfälle für bestimmte Versionen nicht ausführbar, da die zu testende Funktionalität in diesem Systemausbau noch nicht existierte.

Praktisch wäre eine automatische Evaluation. Dabei müssten jedoch die Änderungen im Testsystem auf die im SystemTestPortal vorhandenen Testfälle abgebildet werden. Das ist in der Praxis jedoch (zumindest automatisiert) nahezu unmöglich. GitLab beispielsweise

stellt bei Veröffentlichung einer neuen Version nur die geänderten Dateien zur Verfügung. In jedem Testfall im SystemTestPortal müssten also die Dateien verlinkt sein, die durch diesen getestet werden. Das ist für die Test-Manager ein großer Wartungsaufwand und somit für die Praxis nicht relevant.

Statt einer vollautomatischen Evaluation, welche Tests für eine Version zwingend ausgeführt werden müssen, kann man auch den Test-Managern eine größere Verantwortung übergeben. Im Detail heißt das, dass diese vor der eigentlichen Ausführung der Tests den zu testenden Testfällen die korrekte Version des Testsystems zuweisen müssen. Für die Evaluation des Ergebnisses der Tests werden dann nur die Testfälle analysiert, die für diese Version geeignet sind.

Definition des erfolgreichen Tests einer Version



- Successful, Partially successful = Test passed
- Not executed, Failed, Not assessed = Test failed

Abbildung 4.3: Evaluation der Testergebnisse des SystemTestPortals

Während der Integration muss das SystemTestPortal selbstständig entscheiden, ob eine Version des Testsystems erfolgreich getestet wurde. Im Abschnitt 3.5 wurde bereits auf mögliche Metriken zur Evaluation eingegangen. Dabei gab es für Tests jedoch nur die Möglichkeit, bestanden oder nicht bestanden zu sein. Das SystemTestPortal bietet auch die Möglichkeit, Tests außerdem als „Bestanden mit Kommentar“ oder „Nicht bewertet“ zu markieren.

Da die CI/CD-Pipeline nur die binären Optionen „Bestanden“ und „Nicht bestanden“ verarbeiten kann, müssen die Ergebnisse der Testausführungen im SystemTestPortal auf diese zwei Optionen abgebildet werden. Abbildung 4.3 zeigt eine mögliche Interpretation der Testergebnisse. Hierbei sind die grauen Felder nicht berücksichtigte Ergebnisse. Das

sind Ergebnisse von Tests, die nicht auf die zu evaluierende System-Under-Test Version anwendbar sind. Grundsätzlich wird für jede Version und jeden Test nur das Ergebnis der neuesten Testausführung ausgewertet.

Die Tests für die aktuelle Version teilen sich auf in „Nicht ausgeführt“ und „Ausgeführt“. Alle nicht ausgeführten Tests einer System-Under-Test Version werden als fehlgeschlagen angesehen. Wenn nicht ausgeführte Tests als erfolgreich angesehen werden, würde für eine nicht getestete Version eine positive Testabdeckung angezeigt werden. Das ist irreführend.

Von den ausgeführten Tests werden natürlich die bestandenen (auch bestanden mit Kommentar) und die nicht bestandenen jeweils als solche angesehen. Nicht bewertete Tests werden ebenso als fehlgeschlagen angesehen, da kein positives Resultat ersichtlich ist.

Nach dieser Einteilung hat man nun nur noch bestandene und nicht bestandene Tests. Damit die Qualität des Testsystems möglichst hoch bleibt, ist definiert, dass alle Tests bestanden sein müssen. Nur wenn alle Tests als bestanden eingestuft sind, wird eine positive Rückmeldung zum CI/CD-System gesendet und die CI/CD-Pipeline wird fortgesetzt. Eine mögliche Erweiterung wäre eine einstellbare, prozentuale Grenze der bestandenen Tests.

Zuständigkeit für das manuelle Testen

Beim Testen sind verschiedene Personen beteiligt. Test-Manager beispielsweise koordinieren die Durchführung der Tests, die dann von den Testern gemacht wird. Diese Rollen müssen im SystemTestPortal verfügbar sein, um die Zuständigkeit von bestimmten Aufgaben definieren zu können.

Jedes Mitglied eines Projekts muss dafür eine Rolle zugewiesen bekommen. Durch die Rollenzuweisung kann das SystemTestPortal automatisch festlegen, dass die Test-Manager Verwaltungsaufgaben übernehmen sollen und die Tester die eigentlichen Tests ausführen.

Neben dem Tester und dem Test-Manager kann es noch Vorgesetzte geben. Diese sind ausschließlich an den Ergebnissen interessiert und nicht an Details der Ausführung.

Bei der Umsetzung muss berücksichtigt werden, dass manuell weitere Rollen angelegt werden können. Die Rollen unterscheiden sich neben dem Namen auch in den Berechtigungen. Der Test-Manager kann zum Beispiel Tests editieren, wohingegen der Tester diese nur ausführen darf.

4.4 Sicherheitsbedingte Entwurfbeschränkungen

Sichtbarkeit der Ergebnisse

Damit nicht jeder die Ergebnisse eines Testsystems abrufen kann, kann ein geheimes Token mitgesendet werden. Dieses sollte im SystemTestPortal generiert und im CI/CD-System hinzugefügt werden. Webhooks in GitLab unterstützen das Mitsenden von Tokens [32]. Bei Verwendung einer eigenen Applikation zur Kommunikation zwischen Pipeline und

dem SystemTestPortal kann das Token per Parameter übergeben werden und im Body der Anfrage mitgeschickt werden. Zur Generierung des Tokens kann das crypto/rand-Package von Golang verwendet werden [20].

Wenn ein HTTP POST-Request gesendet wird, überprüft das SystemTestPortal ob das im Request mitgesendete Token dem im SystemTestPortal hinterlegten Token entspricht. Sind die Tokens nicht gleich, wird ein Status 403 (Forbidden) zurückgesendet.

4.5 Aktueller Stand der Integration manueller Tests

Vorhandene Testmanagement-Tools legen im Bereich der Integration in externe Systeme mehr Wert auf die Ausführung automatisierter Tests und das zentrale Anzeigen aller Testergebnisse (automatisiert und manuell) in ihrem System. Das Integrieren und Ausführen der manuellen Tests während der Integrationspipeline ist in keinem der bekannten Testmanagement-Tools umgesetzt.

Der in dieser Arbeit entwickelte Ansatz zur Integration von manuellen Tests in das Continuous Deployment bindet nicht nur Auswertung der vorhandenen Testergebnisse in die Pipeline ein. Vielmehr bietet es die Möglichkeit, die eigentliche Ausführung der manuellen Tests während der Ausführung der Pipeline durchzuführen und viele Verwaltungsaufgaben des manuellen Testens zu automatisieren.

Im Vergleich zum manuellen Testen ohne die Integration entsteht dadurch eine Zeit- und somit Kostenersparnis und eine Steigerung der Testqualität. Die Entwickler profitieren von einem schnellen, konsistenten und detaillierten Feedback der manuellen Tests. Das Feedback wird automatisch bereitgestellt und der Kommunikationsaufwand für Tester und Entwickler sinkt, ohne einen Qualitätsverlust bemängeln zu müssen.

5 Integration in das Continuous Deployment

In diesem Kapitel wird die entstandene Integration vorgestellt. Diese wurde im Rahmen der Arbeit primär für GitLab ausgelegt, da die Entwicklung des SystemTestPortals ebenso in GitLab gemacht wird. Dabei wurde jedoch die Erweiterung der Integration in andere Systeme mit bedacht und eine möglichst generisch einsetzbare Anwendung geschaffen. Die Konfiguration der Integration für andere Systeme kann dennoch vom hier Beschriebenen abweichen.

5.1 Grundsätzliche Funktionsweise

Abbildung 5.1 zeigt die Systemarchitektur. Der Entwickler veröffentlicht eine neue Version und löst dadurch die CI/CD-Pipeline aus. Diese führt Aufgaben wie das Bauen, automatische Testen und Deployen in eine Testumgebung aus. Danach werden die manuellen Tests ausgeführt. In einem Docker-Container wird eine Webapplikation gestartet, die für die Kommunikation mit dem SystemTestPortal zuständig ist. Sie sendet direkt nach dem Start eine Anfrage an das SystemTestPortal. Das SystemTestPortal verwaltet die zu testende Version und ist für die Ausführung der manuellen Tests zuständig.

Damit das SystemTestPortal nicht alle aktiven Pipelines verwalten muss, ist jede einzelne aktuell laufende Applikation in der CI/CD-Pipeline selbst verantwortlich für das Aktualisieren des Zustands der Tests. Jede aktive Webapplikation sendet eine Anfrage für eine bestimmte Version und Variante an das SystemTestPortal. Je nach Fortschritt der Testausführung reagiert das SystemTestPortal und sendet eine Antwort zurück. Die Webapplikation sendet nach einiger Zeit erneut eine Anfrage. Dies geschieht so lange, bis das SystemTestPortal die Ergebnisse der ausgeführten Tests zurücksendet. Diese Art der Kommunikation bezeichnet man als „Polling“ [22]. Allgemein gilt: Beim Polling wird der Status eines externen Gerätes (das SystemTestPortal) von einem Client-Programm abgerufen (die Webapplikation). Es wird meistens bei Eingabe/Ausgabe zwischen Computer und der Umgebung verwendet.

Sind alle Tests ausgeführt, bereitet die Webapplikation das Feedback für die Entwickler vor. Beim Beenden der Webapplikation und des Docker-Containers wird dieses Feedback an die Pipeline gesendet, wo der Entwickler die Testergebnisse abrufen kann. Nach erfolgreicher Ausführung der Tests kann das Testsystem in die Produktion ausgeliefert werden.

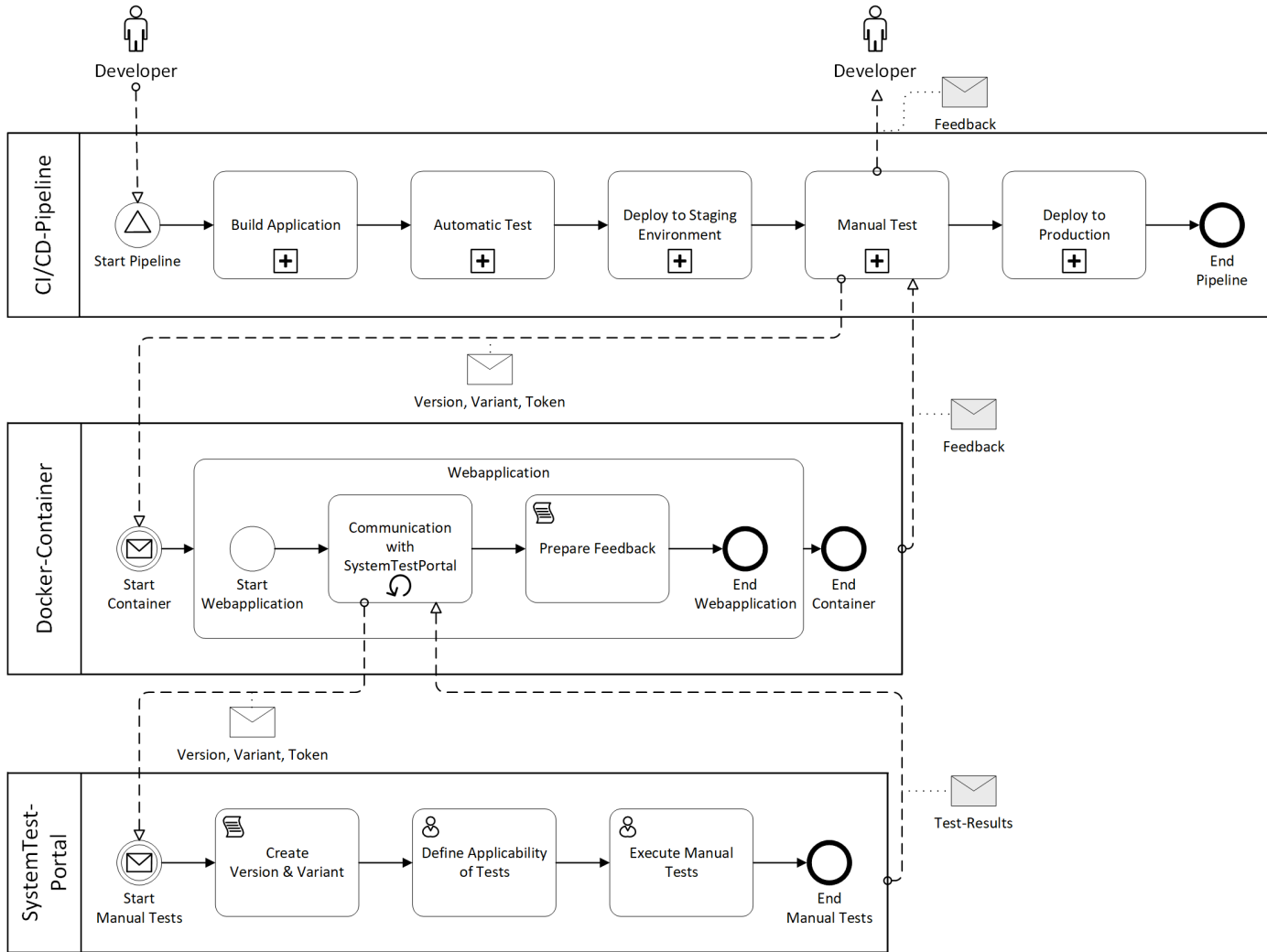


Abbildung 5.1: Systemarchitektur der Integration manuellen Testens

5.2 Erweiterungen im SystemTestPortal

Um die Integration in das SystemTestPortal möglich zu machen, wurde es im Zuge dieser Arbeit neben der eigentlichen Schnittstelle um einige Funktionalitäten erweitert.

5.2.1 Rollen in einem Projekt

In der Entwicklung eines Systems gibt es viele Personen, die unterschiedliche Rollen einnehmen. Die Rollen unterscheiden sich in den Interessen, Verantwortungen und Berechtigungen. Das können unter anderem Manager eines Unternehmens, Entwickler oder Tester sein. Damit jeder Nutzer des SystemTestPortals das für ihn optimale Benutzungsgedühl hat, können in einem Projekt Rollen erstellt und verwaltet werden.

Diese Rollen bestimmen vorerst nur die Berechtigungen eines Nutzers in einem Projekt. Es ist jedoch im weiteren Verlauf der Entwicklung auch geplant, die angezeigten Informationen je nach Rolle anzupassen. Manager haben ein anderes Interesse an den Tests als Tester. Sie wollen primär eine Übersicht über die Ergebnisse haben. Tester hingegen sind an den Tests interessiert, die sie ausführen sollen. Für sie ist ein Gesamtbild der Ausführungen weniger interessant.

Anfangs hatte jede Rolle unveränderliche Berechtigungen. Ein Tester durfte ausschließlich Tests ausführen. Ein Test-Manager durfte zusätzlich Tests anlegen und editieren. Für eine höhere Flexibilität wurde anschließend durch das Team des Studienprojekts das manuelle Anpassen der Berechtigungen einer Rolle entwickelt. Auswählbare Berechtigungen gehen hierbei von der Ausführung der Tests bis hin zu Einstellungen des Projekts. In Abbildung 5.2 sind die Berechtigungen eines Testers zu sehen. Er darf ausschließlich Tests ausführen.

Tester Delete

Execution	Test Cases	Test Sequences	Members	Settings
<input checked="" type="checkbox"/> Execute	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Duplicate <input type="checkbox"/> Assign	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Duplicate <input type="checkbox"/> Assign	<input type="checkbox"/> Edit	<input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Edit Perm.

Abbildung 5.2: Berechtigungseinstellungen einer Rolle

Die verfügbaren Rollen werden für die Integration verwendet, um automatisiert die richtigen Benachrichtigungen und Aufgaben für den manuellen Test eines Systems an die entsprechenden Projektmitglieder zu liefern. Die Verwaltung der Tests ist Aufgabe

der Test-Manager. Die Ausführung der Tests betrifft nur die Tester eines Projekts. Nach dem Ausführen werden aber nur die Manager und Vorgesetzten über die Testergebnisse benachrichtigt, da dies für die Tester nicht relevant ist.

5.2.2 Aufgabenliste für Benutzer

Die Nutzer des SystemTestPortals müssen über die Tests, die sie ausführen sollen, benachrichtigt werden. Um eine umfassende Übersicht über alle noch offenen Zuweisungen zu haben, wurde das SystemTestPortal um eine Aufgabenliste erweitert. Jeder Nutzer hat seine eigene Aufgabenliste, die er über die Navigationsleiste erreichen kann.

In der Aufgabenliste sind alle noch zu bearbeitenden Aufgaben aufgelistet. Eine Testzuweisung ist in Abbildung 5.3 zu sehen.

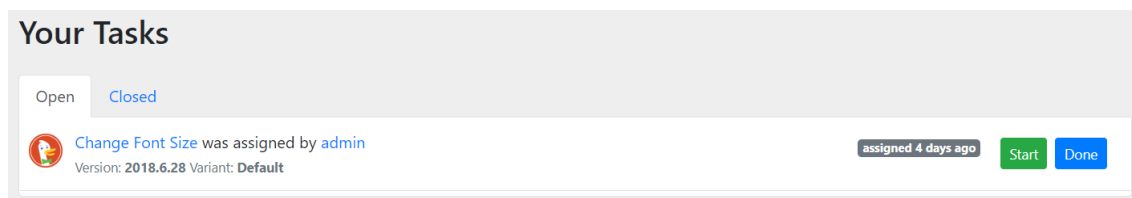


Abbildung 5.3: Aufgabe für Tester mit Version und Variante

Tests können von jedem Projektmitglied zugewiesen werden, das die entsprechenden Berechtigungen hat. Das sind in den Standardrollen eines Projekts der Test-Manager und Supervisor.

Ausführung eines Tests mit vordefinierter Version und Variante

Bei der Zuweisung muss die konkrete Version und Variante des Testsystems, die getestet werden soll, ausgewählt werden. Somit muss der Tester sich nicht die gewünschte Version und Variante merken und dann in der Testausführung manuell auswählen. Das vereinfacht die Durchführung für den Tester und verringert das Fehlerrisiko. Über den „Start“-Button kann der Test mit der vordefinierten Version und Variante gestartet werden.

5.3 Kommunikation zwischen dem CI/CD-System und dem SystemTestPortal

Für die Kommunikation zwischen dem CI/CD-System und dem SystemTestPortal wurde eine für diesen Zweck konzipierte Webapplikation entwickelt. Sie erhält beim Starten entsprechende Parameter, die sie für die Erstellung und Sendung der Anfrage an das SystemTestPortal benötigt. Wenn das SystemTestPortal eine Antwort sendet, verarbeitet die Webapplikation diese und erstellt daraus das Feedback für den Entwickler.

Die Applikation ist in Golang geschrieben und in GitLab unter der Adresse gitlab.com/schneisn/stp-ci-integration veröffentlicht. Golang bietet eine weitreichende Unterstützung zur schnellen Entwicklung von Webanwendungen. Zudem ist das SystemTestPortal in derselben Sprache entwickelt und zukünftige Änderungen an der Applikation fallen den Entwicklern des SystemTestPortals leichter.

5.3.1 Docker-Image mit der Webapplikation

Um die Webapplikation in viele CI/CD-Systeme einbinden zu können, wird sie mit Hilfe von Docker bereitgestellt.

Ein Docker-Image ist eine Datei, bestehend aus mehreren Schichten, die genutzt wird um Code in einem Docker Container auszuführen [9]. In einem Docker-Image werden alle Programmbibliotheken, Werkzeuge und weiteren Dateien und sonstige Abhängigkeiten (Dependencies) für den auszuführenden Code zusammengestellt. Wenn ein Docker-Image gestartet wird, entstehen daraus eine oder mehrere Instanzen des Containers.

Nutzer von Docker können die Images in privaten oder öffentlichen Repositories (Docker-Registries) speichern. Von dort aus können Container erzeugt werden, Images getestet oder mit anderen Nutzern geteilt werden. Das Docker-Image der Webapplikation ist verfügbar in der Docker-Registry von GitLab unter registry.gitlab.com/schneisn/stp-ci-integration[4].

Konfiguration des Docker-Images

Listing 5.1 zeigt die Konfiguration für das Docker-Image der Webapplikation. Das Image „frolvad/apline-glibc“s fungiert als Basis für dieses Image und stellt Werkzeuge zur Verfügung, damit das kompilierte Programm der Webapplikation in der CI/CD-Pipeline funktioniert [7].

Der ADD Befehl fügt vom lokalen Dateisystem die definierte Datei hinzu. Das ist die ausführbare Webapplikation. Diese wird in den „/app/“-Ordner hinzugefügt. Um die Größe des Images auf ein Minimum zu reduzieren, wird nur der kompilierte Programmcode hinzugefügt. Alternativ könnte der Sourcecode eingefügt und erst beim Ausführen des Docker-Images kompiliert werden. Dadurch wären weitere Hilfsprogramme notwendig und die Größe des Images würde von unter 20 MegaByte (MB) auf über 400 MB steigen. Je größer das Image ist, desto länger braucht die CI/CD-Pipeline um das Image herunterzuladen.

Die Webapplikation empfängt Anfragen auf Port 4200. Im Docker-Image muss extra spezifiziert werden, welche Ports für die enthaltenen Programme offengelegt werden.

Damit die Webapplikation auch wirklich ausgeführt werden kann, muss „Bash“ in Zeile 15 hinzugefügt werden. Bash ist eine Befehlssprache für Linux und macOS und wird hier verwendet, um die Webapplikation mit dem Command (CMD) Befehl zu starten. Bevor damit Befehle ausgeführt werden können, muss im ENTRYPOINT, also dem Einstiegspunkt des Docker-Containers, Bash gestartet werden.

Listing 5.1 Konfiguration des Docker-Images für die Webapplikation

```
1 FROM frovlad/ alpine - glibc
2
3 LABEL version="1.0"
4 LABEL description="Image for integrating the SystemTestPortal into the
   CI/CD-Pipeline"
5
6 WORKDIR /app
7
8 # Only add binary
9 ADD stp-ci- integration /app/
10
11 # App runs on port 4200
12 EXPOSE 4200
13
14 # Add bash to image
15 RUN apk add --no-cache bash
16
17 ENTRYPOINT ["/bin/bash", "-c"]
18
19 # The flags url, version and token will be overridden in the respective
   projects
20 CMD stp-ci- integration -- url=$stp-url -- version=$stp-version
   -- variant=$stp-variant -- token=$stp-token
```

Vorteile der Verwendung von Docker

Wie weiter oben schon gesehen, sind Docker-Images extrem leichtgewichtig. Zusätzlich sind sie einfach zu handhaben, da man nicht unzählige Programme und Umgebungen manuell installieren muss, um ein Programm auszuführen. Dazu lassen sich vorhandene Docker-Images nutzen, die nur eingebunden werden müssen.

Docker kann in jedem CI/CD-System verwendet werden und ermöglicht somit die einfache Einbindung des SystemTestPortals in die Pipeline anderer Systeme.

5.3.2 Konfiguration der Pipeline

Die Pipeline des CI/CD-Systems muss entsprechend konfiguriert werden, um die Stage für das manuelle Testen einzubinden. In diesem Abschnitt werden die Faktoren für eine funktionierende Konfiguration erklärt. Anhand von GitLab wird eine Beispielkonfiguration gezeigt.

Benötigte Parameter für die Webapplikation

Damit eine gewisse Flexibilität gegeben ist, kann die Webapplikation mit verschiedenen Parametern gestartet werden. Diese werden im CI/CD-System durch manuell festgelegte oder automatisch gesetzte Variablen definiert.

Die Parameter, die für ein Projekt gleichbleibend sind und manuell gesetzt werden, umfassen z.B.:

- Die Projekt-URL im SystemTestPortal
- Das Token um auf das Projekt zugreifen zu können
- Die Wiederholungsrate der Anfragen
- Die Variante des Testsystems.

Theoretisch kann sich die Variante des Testsystems, welches getestet werden soll, je nach Durchführung der Continuous Integration/Continuous Deployment-Pipeline ändern. Die Variante ist jedoch ein Parameter, der nicht automatisch durch GitLab gesetzt werden kann. Somit muss dieser Parameter vor der Ausführung einer Pipeline manuell definiert werden.

Der Name der Version ist für jede Pipeline unterschiedlich. Dieser kann in der Pipeline über die vordefinierten Umgebungsvariablen [11] abgerufen werden, die vom CI/CD-System bereit gestellt werden. Setzt ein Entwickler einen neuen Tag mit dem Namen „1.2.0“, wird der Pipeline, die durch Setzen des Tags ausgelöst wird, dieser Name übergeben. Die Variable um diesen Namen abzurufen ist „CI_COMMIT_TAG“.

Statische Parameter können in GitLab über die CI/CD-Variablen gesetzt werden. Die CI/CD-Variablen sind erreichbar in den Projekteinstellungen unter Variablen. Diese Variablen können in der Stage einer Pipeline abgerufen werden und einem Programm als Parameter übergeben werden.

Für eine funktionierende Kommunikation zwischen dem CI/CD-System und dem SystemTestPortal werden manuell definierte Variablen mit den folgenden Namen und Werten benötigt:

- **STP_URL**: Die benötigte URL zum Projekt im SystemTestPortal. Diese kann im SystemTestPortal in den Projekteinstellungen abgerufen werden.
- **STP_TOKEN**: Das benötigte Token um auf das Projekt im SystemTestPortal zugreifen zu können. Dieses kann in den Projekteinstellungen abgerufen werden.
- **STP_VARIANT**: Die kundenspezifische Variante des Testsystems.

Sicherheitsaspekte des generierten Zugriffstokens

Das im SystemTestPortal hinterlegte Zugriffstoken ist ein 16-stelliger, zufällig generierter String. Er wird bei Erstellen des Projektes mit dem „crypto/rand“-Package von Golang erzeugt [20]. Bei einer eingehenden Anfrage wird das hinterlegte Token mit dem in der Anfrage mitgesendeten Token verglichen. Nur wenn diese übereinstimmen wird die Anfrage weiter bearbeitet.

Besonders bei sicherheitskritischen Systemen dürfen keine Informationen über aufgetretene Sicherheitslücken an unbefugte Personen gelangen. Selbst bei nicht-sicherheitskritischen Systemen besteht die Gefahr, dass durch Sicherheitslücken, die

beim Testen einer neuen Version entdeckt worden sind, Daten bei älteren Versionen unerlaubt abgerufen werden. Die Sicherheitslücke kann bei älteren Versionen ebenso vorhanden sein, jedoch wurde sie dort beim Testen nicht entdeckt und nicht behoben.

Beispielkonfiguration der Pipeline in GitLab

Listing 5.2 Konfiguration der GitLab CI-Pipeline

```
1  stages:
2    # Build, Test, Deploy to Test-Server
3    - manual-test
4    # Deploy to Production
5
6  manual-test:
7    image: registry.gitlab.com/schneisn/stp-ci-integration:latest
8    stage: manual-test
9    script:
10     - cp /app/stp-ci-integration $CI_PROJECT_DIR/stp-ci-integration
11     - cd $CI_PROJECT_DIR/
12     - ./stp-ci-integration --url="$STP_URL" --version="$CI_COMMIT_TAG"
13       --variant="$STP_VARIANT" --token="$STP_TOKEN" --time="300s"
14
15  only:
16    - tags
17
18  artifacts:
19    when: always
20    paths:
21     - testresults.zip
```

In Listing 5.2 wird die Stage ausschließlich für Tags ausgeführt. Sollten manuelle Tests für jeden Commit oder Merge-Request ausgeführt werden, muss der „only“-Tag angepasst werden. Zusätzlich muss die vordefinierte Variable „CI_COMMIT_TAG“ geändert werden, da diese nur im Falle eines Tags einen Wert liefert. Weitere Variablen und deren Anwendung sind unter [11] zu finden.

Mit dem Tag „artifacts“ können Dateien definiert werden, die persistent zu speichern sind. Näheres dazu ist in Abschnitt 5.5 zu finden.

Eine in der Praxis eingesetzte Pipeline würde weitere Stages wie das Kompilieren und Testen der Applikation umfassen. Zusätzlich dazu ist es hilfreich, die Applikation vor dem manuellen Testen auf einen Test-Server zu deployen, damit alle Tester in der selben Umgebung testen. Den manuellen Tests folgt das Deployment in die Produktion. Somit agieren die manuellen Tests als zusätzliche Qualitätssicherungsstufe zwischen den automatischen Tests und der Auslieferung in die Produktion.

Das „Image“ definiert das Docker-Image, welches die Webapplikation zur Kommunikation mit dem SystemTestPortal enthält. Das Docker-Image definiert den Docker-Container, der nach Beginn der Stage gestartet wird. Im „Script“-Teil wird die Webapplikation im Container gestartet und die benötigten Parameter werden übergeben. Die Webapplikation wird ausgeführt, bis sie sich selbst beendet oder bis ein Timeout durch die Einstellungen

der Pipeline auftritt. Dieser ist in GitLab standardmäßig auf eine Stunde eingestellt. Da manuelle Tests meistens mehr Zeit benötigen, muss dieser Timeout in den Projekteinstellungen auf eine angemessene Zeit erhöht werden.

5.3.3 Webapplikation zur Kommunikation mit dem SystemTestPortal

Der Webapplikation unterliegt der im folgenden vorgestellte Algorithmus, um Anfragen an das SystemTestPortal zu senden und sich je nach Rückmeldung entsprechend zu verhalten. Mit der Methode des Pollings wird hierbei der Status der Tests wiederholt abgefragt, bis ein positives oder negatives Testergebnis zurückgemeldet wird [22]. Um ein unendliches Warten bei fehlgeschlagenen Anfragen zu vermeiden, wird die Webapplikation automatisch nach zehn Fehlversuchen mit einer negativen Rückmeldung beendet.

Algorithmus 5.1 Algorithmus der Webapplikation zur Bestimmung des Testzustandes

```
procedure REQUESTTESTRESULTS(version, variant, token)
  allowedFailures ← 10
  while allowedFailures > 0 do
    response ← SENDREQUEST(version, variant, token)
    if error sending request then
      allowedFailures ← allowedFailures – 1
    else if response = InternalServerError (500) then
      allowedFailures ← allowedFailures – 1
    else if response != StatusNotFound (404) then
      if response = StatusOK (200) then
        Exit with positive Result
      else
        Exit with negative Result
      end if
    end if
    Wait for N seconds
  end while
end procedure
```

Algorithmus 5.1 beschreibt den Algorithmus in Pseudocode. Die Anzahl der erlaubten Fehlversuche wurden auf zehn festgelegt. Dieser Parameter könnte ebenso wie die anderen manuell gesetzt werden, aufgrund der geringen Relevanz für den Nutzer wurde darauf jedoch verzichtet.

Nach dem Starten der Webapplikation wird ein Request-Body (Payload) in JSON erzeugt. Dieser Body enthält Informationen wie Version, Variante und das Token des STP-Projekts. Listing 5.3 zeigt die Informationen des Requests an das Projekt „DuckDuckGo.com“ für die Version 1.0.0.

Zukünftig kann der Body mit beliebigen Variablen erweitert werden. Der Request ist vom Typ „POST“, da diese Art der Requests einen Body haben kann. Der Request wird an die in der Variable „STP_URL“ gesetzte Adresse gesendet. Das SystemTestPortal verarbeitet die Anfrage und sendet eine Antwort zurück.

Listing 5.3 Anfrage an das STP mit den benötigten Informationen im Message-Body

```
1      POST /demo/DuckDuckGo.com/ci HTTP/1.1
2      Host: stp-host.com
3      User-Agent: Go-http-client /1.1
4      Content-Length: 85
5      Content-Type: application/json
6      Accept-Encoding: gzip
7      X-Forwarded-For: 35.237.201.217
8
9      {"version": "v1.0.0",
       "variant": "Default", "token": "5fc9d9141bb582bf98518f794625f7d8"}
```

5.3.4 Statuscodes der Antwort

Wie in Algorithmus 5.1 zu sehen ist, wird der Status der Antwort als Zustand der Tests interpretiert. Es gibt vier verschiedene Statuscodes.

- Status Not Found (404): Die Tests sind noch nicht komplett ausgeführt
- Status Internal-Server-Error (500): Bei der Analyse im SystemTestPortal ist ein Fehler aufgetreten
- Status OK (200): Alle Tests für die angefragte Version und Variante wurden erfolgreich ausgeführt
- Status Bad Request (400): Dieser und alle anderen, nicht genannten Status-Codes werden als fehlgeschlagene Tests interpretiert.

Der Webserver interpretiert die Statuscodes und protokolliert den aktuellen Zustand. Dieses Protokoll ist in GitLab in der Ansicht des aktuellen Jobs zu sehen. Ein Beispiel für die Aufzeichnung ist in Abbildung 5.6 zu sehen.

Bei Status 404 oder 500 sendet der Webserver eine erneute Anfrage an das SystemTestPortal nach einer gewissen Wartezeit.

Bei Status 200 beendet sich die Applikation mit Exitcode 0. Das bedeutet eine erfolgreiche Ausführung des Programms. Jeder andere Exitcode wird als fehlerhafte Beendigung eines Programms interpretiert. Durch Beenden der Applikation wird der Docker-Container ebenso mit Exitcode 0 beendet und die CI/CD-Pipeline deutet das als erfolgreiche Ausführung der Stage.

Im Gegensatz dazu bedeutet der Statuscode 400, dass die Tests nicht erfolgreich waren. Der Server und Docker-Container werden mit Exitcode 1 geschlossen und die Stage wird als fehlgeschlagen markiert.

5.4 Schnittstelle für die Integration im SystemTestPortal

5.4.1 Automatisierte Zuweisung der Tests

In Abbildung 5.4 ist ein Überblick über die Aktionen des SystemTestPortals beim Testen einer neuen Version gegeben. Bevor eine Version für eine Systemvariante von den Testern getestet werden kann, müssen dafür Vorkehrungen getroffen werden. Das umfasst das Erstellen der Testversion und der Variante und das Bestimmen, welche Testfälle für die neue Version ausgeführt werden können.

Der entsprechende Algorithmus, der bei jeder Anfrage dynamisch entscheidet, welche Aktion als nächstes ausgeführt werden muss, ist in vereinfachter Form in Algorithmus 5.2 dargestellt.

Algorithmus 5.2 Algorithmus zur Evaluation der auszuführenden Aktion

```
1: procedure TESTASSIGNMENT
2:   if TESTSAREASSIGNEDTOTESTERS then
3:     Return
4:   end if
5:   if VERSIONALREADYEXISTS then
6:     if APPLICABILITYISSET then
7:       CREATETASKFORTESTERS
8:     else
9:       ASSIGNVERSIONTOTESTS
10:      CREATETASKFORMANAGERS
11:    end if
12:  else
13:    CREATEVERSION
14:    ASSIGNVERSIONTOTESTS
15:    CREATETASKFORMANAGERS
16:  end if
17: end procedure
```

Die erste Bedingung, die überprüft wird, ist in Zeile 2. Dort wird überprüft, ob die Tests schon den Testern zugewiesen sind. Ist dies der Fall, müssen keine weiteren Aufgaben zugewiesen werden. Es wird darauf gewartet, dass die Tester alle Tests ausführen.

Wenn es noch keine Aufgaben für die Tester gibt, wird mit „VersionAlreadyExists“ überprüft, ob die Version und Variante schon im System existieren, da diese benötigt werden, um die Aufgaben überhaupt erstellen zu können.

Existieren diese nicht, werden sie in Zeile 10 erstellt und müssen nun den korrekten Testfällen zugeordnet werden. Da eine dauerhaft zuverlässige automatische Ermittlung der zu testenden Testfälle nicht möglich ist, wird hierbei Verantwortung an die Test-Manager übergeben. Der Grund, weswegen die automatische Zuweisung nicht möglich ist, wurde in Abschnitt 4.3.2 erläutert.

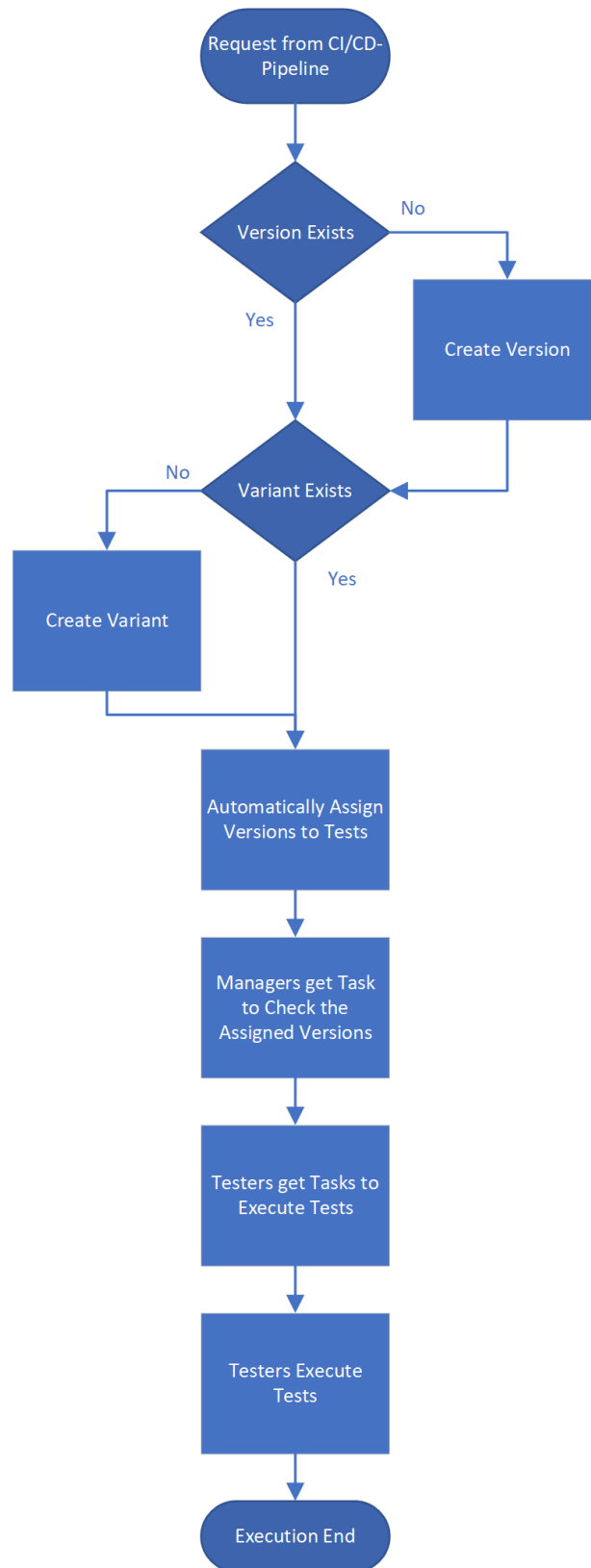


Abbildung 5.4: Ablauf des manuellen Testens einer neuen Version im SystemTestPortal

Zuerst erhielten die Test-Manager die Aufgabe, die angefragte Version/Variante jedem Testfall, der für diese Version/Variante des Testsystems ausgeführt werden kann, einzeln zuzuweisen. Das ist jedoch mit großem Aufwand verbunden, besonders wenn die Anzahl der Testfälle steigt.

Deswegen wurde die Zuweisung weiterentwickelt und die neue Version wird per Standard den Testfällen zugewiesen, die auch anwendbar auf die vorherige Version waren (Zeile 14). Hierbei wird angenommen, dass die Versionen alphabetisch sortiert sind. In einer zukünftigen Version des SystemTestPortals ist angedacht, manuelles Sortieren der Versionen zu ermöglichen. Nun müssen die Test-Manager nur noch überprüfen, ob die Version den korrekten Testfällen zugewiesen ist und eventuell eine geringe Anzahl an Testfällen anpassen (Abbildung 5.5). Sollte die Version bereits existieren, aber ist noch keinem Test zugewiesen, wird sie ebenso automatisch zugewiesen und die Test-Manager erhalten die Aufgabe, die Anwendbarkeit der Tests zu überprüfen (Zeile 9 und 10). Das verhindert, dass, wenn die Version im SystemTestPortal schon manuell erstellt wurde, trotzdem die Anwendbarkeit aller Tests manuell aktualisiert werden muss.

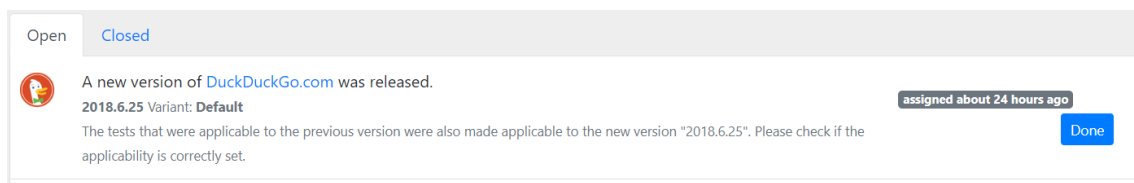


Abbildung 5.5: Aufgabe für Manager mit Version und Variante

Nachdem mindestens ein Test-Manager entschieden hat, dass diese Aufgabe erledigt ist und es wirklich Tests gibt, denen die Version und Variante zugewiesen ist, können die Aufgaben für die Tester erstellt werden. Beide Bedingungen müssen erfüllt sein, damit Tester mit dem Testen der neuen Version beginnen können. Es kann vorkommen, dass ein fauler Test-Manager die Aufgabe schon als erledigt setzt und damit die Test-Suite zum Testen freigibt. In Wirklichkeit gibt es aber noch gar keine Testfälle die getestet werden können, da die Version noch keinem Test zugewiesen ist. Diese Bedingung wird in Zeile 6 durch „ApplicabilityIsSet“ überprüft.

Die Aufgaben für die Tester enthalten die konkrete Version und Variante, die getestet werden soll (Abbildung 5.3). Die Ausführung des Tests mit der ausgewählten Version und Variante kann über den Button „Start“ begonnen werden. Nach der Ausführung wird die Aufgabe automatisch als erledigt markiert.

5.4.2 Analyse der Testergebnisse

Algorithmus 5.3 bietet einen Einblick in die Analyse der Testergebnisse. Je nach Ergebnis wird ein anderer Statuscode zurückgegeben.

Solange noch nicht alle Tests einer Version ausgeführt sind, wird Status 404 zurückgegeben. Während dem Iterieren über alle Tests wird die Gesamtanzahl Tests und die Anzahl der bestandenen Tests gezählt. Ist ein Test nicht auf die Version und Variante anwendbar,

Algorithmus 5.3 Analyse der Testergebnisse

```
1: procedure ANALYSEPROTOCOLS
2:   if NOTALLTESTSEXECUTED then
3:     Return Status Not Found (404)
4:   end if
5:   passedTests ← 0
6:   totalTests ← 0
7:   for all Tests do
8:     if TESTISNOTAPPLICABLE then
9:       Skip Test
10:    end if
11:    totalTests ← totalTests + 1
12:    if Testresult = Passed or Testresult = Partially Passed then
13:      passedTests ← passedTests + 1
14:    end if
15:  end for
16:  if totalTests ≤ 0 then
17:    Return Status Not Found (404)
18:  end if
19:  if TESTSNOTPASSED(totalTests, passedTests) then
20:    Return Status Bad Request (400)
21:  end if
22:  Return Status OK (200)
23: end procedure
```

wird dieser ignoriert. Ist das Testresultat der letzten Ausführung eines Tests „Bestanden“ oder „Teilweise bestanden“, wird der Test als bestanden gewertet. Dies wird für jeden Test wiederholt. Sollte trotz automatischer Zuweisung der Tests kein Test auf die Version/Variante anwendbar sein, wird ein Status 404 zurückgegeben, da diese Systemversion immer noch getestet werden muss (Zeile 16/17).

Je nach Metrik werden die Tests nun bewertet. Um eine hohe Qualität sicherzustellen, ist die Metrik für diese Arbeit so gewählt, dass kein Test fehlgeschlagen sein darf. Weitere Metriken sind in Abschnitt 3.5 vorgestellt. Ist die Version des Testsystems nicht erfolgreich getestet worden, wird ein Status 400 zurückgegeben und die Stage in der Pipeline schlägt fehl.

Bei erfolgreicher Analyse wird ein Status 200 gemeldet. Die Stage in der Pipeline wird erfolgreich beendet und die nächste Stage wird fortgesetzt.

5.5 Ausführliches Feedback für die Entwickler

Neben den Testprotokollen, die für Tester, Manager und Vorgesetzte im SystemTestPortal zur Verfügung stehen, müssen Entwickler über den Ausgang der Tests benachrichtigt werden. In der agilen Entwicklung ist ein schnelles Feedback essentiell. Das Feedback sollte möglichst detailliert sein, ohne unnötige Informationen zu enthalten.

Sobald alle Test ausgeführt wurden, erzeugt das SystemTestPortal ein ZIP-Archiv mit allen Testprotokollen und fügt dieses der Antwort bei. Dieses Archiv wird von der Webapplikation automatisch gespeichert und wird durch die Konfiguration der Stage durch den Tag „artifacts“ als Artefakt, also eine Datei, die auch nach dem Beenden der Stage erhalten bleibt, gespeichert.

Entwickler bekommen dadurch einen weiteren Einblick in die Testausführung und müssen nicht extra auf das SystemTestPortal zugreifen, um detaillierte Informationen zu erhalten. Abbildung 5.6 zeigt eine erfolgreiche Ausführung der manuellen Tests. Nach dem Beenden wird das ZIP-Archiv hochgeladen.

```

$ ./stp-ci-integration --url="$STP_URL" --version="$CI_COMMIT_TAG" --variant="$STP_VARIANT" --token="$STP_TOKEN" --time="15s"
2018/08/29 11:07:40 not all tests executed yet
2018/08/29 11:07:55 not all tests executed yet
2018/08/29 11:08:11 not all tests executed yet
2018/08/29 11:08:26 tests were succesfull
$ cp testresults.zip $CI_PROJECT_DIR
Uploading artifacts...
testresults.zip: found 1 matching files
Uploading artifacts to coordinator... ok          id=93012344 responseStatus=201 Created token=Lz5MsWaz
Job succeeded

```

Abbildung 5.6: Ausführung der GitLab-Pipeline mit erfolgreichen Tests

Die Artefakte einer Pipeline können nachträglich eingesehen und heruntergeladen werden (Abbildung 5.7). Artefakte bleiben dauerhaft erhalten. Wenn sie nach einiger Zeit gelöscht werden sollen, kann dies in der Pipelinekonfiguration definiert werden.

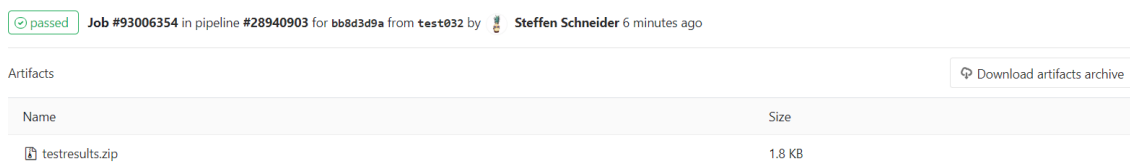


Abbildung 5.7: ZIP-Archiv mit Testprotokollen in GitLab als Artefakt

5.6 Benachrichtigung der Entwickler bei fehlgeschlagenen Tests

Die in Abschnitt 5.5 beschriebenen Testprotokolle helfen den Entwicklern, aufgetretene Fehler zu einer Testversion zu suchen. Die Benachrichtigung über fehlgeschlagene Tests an die Entwickler ist, auch unabhängig von der Ausführung einer Pipeline, wichtig. Testet ein Tester eine ältere Version, die schon ausgeliefert ist, und findet dort einen Fehler, muss dieser Fehler an die Entwickler weitergeleitet werden.

Die Benachrichtigungen können auf verschiedenen Wegen zu den Entwicklern gelangen. Abgesehen von den (unzuverlässigen) Varianten der mündlichen oder auf dem Schriftweg vorgenommenen Überlieferung gibt es elektronische Wege, die für Entwickler einen geringeren Aufwand bedeuten und somit mit höherer Wahrscheinlichkeit Beachtung finden.

Email und SMS-Verkehr könnte zwar automatisiert werden, der Entwickler müsste den Report jedoch selbst in das Issue-Tracking-System übernehmen. Die direkte Integration mit den Issue-Tracking-Systemen ist besser. GitLabs API bietet Zugriff auf die Erstellung eines Issues. Über URL-Parameter werden die gewünschten Attribute konfiguriert. Ein Beispiel für eine Anfrage ist in Listing 5.4 zu sehen.

Listing 5.4 Anfrage an GitLab zur Erstellung eines neuen Issues

```
curl --request POST --header "PRIVATE-TOKEN: 9f0as93-klf23SD3-7dOw32s"
https://gitlab.example.com/api/v4/projects/4/issues?title=Test%20failed&labels=bug
```

Um Zugriff auf die API zu bekommen, benötigt man, analog zum Zugriff auf das Projekt im SystemTestPortal, ein Token. Das Token wird in den Benutzereinstellungen von GitLab konfiguriert und muss dann im SystemTestPortal gespeichert werden (Abbildung 5.8).

Gitlab Integration

Integrate the SystemTestPortal into your GitLab project to automatically create issues when a test fails.

Activate GitLab Integration

Project ID

The project id (or url-encoded path) of your GitLab project.


Project Token

The token to access your project. See [the GitLab documentation](#) for more information.

Abbildung 5.8: Einstellungen des SystemTestPortals für die Benachrichtigung

Der Name des Issues soll einen schnellen Überblick über den Fehler geben. Dafür wird der Kommentar des fehlgeschlagenen Tests verwendet. Ein Beispiel für den Namen ist: „Test failed: Die Farbe der Schaltfläche ist blau statt grün“. Weitere Details sind in der Beschreibung des Issues zu finden. In der Beschreibung sind alle Informationen, die auch in den Protokollen der Tests zu finden sind. Dazu gehen verwendete Version, Variante sowie die Ergebnisse und Kommentare zu den einzelnen Schritten. Ein Beispiel für ein erstelltes Issue ist in Abbildung 5.9 zu sehen.

Angenommen, ein Test wird für eine Testversion öfters ausgeführt und jedes Mal als fehlgeschlagen bewertet, würde für jede Ausführung ein neues Issue erzeugt werden. Für die Entwickler wäre diese Redundanz nervig, da sie bei jedem Issue überprüfen

Open Opened 2 days ago by  **Steffen Schneider** [Close issue](#) [New issue](#)

Test failed: Font size did not change

Test "Change Font Size" failed
 SUT-Version: 2018.5.1
 SUT-Variant: Default
 Executed on 2018-09-23 10:53:22.4886796 +0000 UTC

Tester: admin
 Comment from the tester: *Font size did not change*

Preconditions

Test steps

- 1: Click on the menu-button (The three horizontal lines at the top right corner).
 Result: *Partially successful*
 Comment: *Sidebar is too small*
- 2: Select "other Settings".
 Result: *Pass*
 Observed behavior: *List is there*
- 3: Click "Appearance" in the list on the right of the headline "Settings".
 Result: *Pass*
- 4: Select "Largest" in the drop down menu on the right of "Font Size".
 Result: *Fail*
 Observed behavior: *Font siez does not change*
- 5: Scroll to the end of the settings list and click "Save and Exit".
 Result: *Pass*

This issue was automatically created by the SystemTestPortal.

Abbildung 5.9: Automatisch erstelltest Issue eines Testprotokolls zu fehlgeschlagenem Test

müssten, ob der Fehler schon aufgenommen wurde. Aus diesem Grund speichert das SystemTestPortal, ob für einen Testfall und eine Version/Variante schon ein Issue erstellt wurde. Existiert bereits ein Eintrag in der Datenbank, wird kein Fehlerreport gesendet.

5.7 Nutzen der Integration manueller Tests

Die Integration von manuellen Tests in das Continuous Deployment bietet mit dem hier vorgestellten Ansatz mehrere Vorteile gegenüber den losgelösten manuellen Tests.

Die Kommunikation zwischen dem Tester und Entwickler wird sowohl vereinfacht als auch verbessert, da Benachrichtigungen und Informationen automatisch und somit schneller ausgetauscht werden. Durch die verbesserte Testzuweisung mit auswählbarer Version

und der neuen Aufgabenliste im SystemTestPortal wissen Tester zeitnah, welche Tests sie ausführen sollen. Informationen wie die Testergebnisse werden für die Entwickler einheitlich präsentiert. Dadurch entfällt die Einarbeitungszeit in unterschiedliche Darstellungen von Testprotokollen.

Durch eine starke Kohäsion der neuen Stage in der Pipeline ist die Einbindung der manuellen Tests kompatibel mit den in der Pipeline definierten automatischen Tests. Die manuellen Tests können selbst in Projekten ohne automatische Tests verwendet werden, da sie vollständig unabhängig von anderen Schritten der Integration und des Deployments sind.

Durch die Integration steigt die Testqualität im gesamten Entwicklungsprozess einer Applikation, besonders durch die Reduzierung des Verwaltungsaufwandes für die manuellen Tests.

6 Evaluation der Lösung

Damit die Integration des manuellen Testens von dritten Personen bewertet werden konnte, wurde diese mit ausgewählten Personen durchgeführt. Dabei wurde besonders auf Merkmale wie die Benutzerfreundlichkeit und einen angemessenen Arbeitsfluss geachtet. Für eine geeignete Messung der Verbesserung der Softwarequalität müsste eine Langzeitstudie durchgeführt werden. Dies übersteigt jedoch den zeitlichen Rahmen dieser Arbeit. Die mögliche Durchführung der Langzeitstudie ist in Abschnitt 6.5 vorgestellt.

6.1 Auswahl der Probanden

Die Evaluation wurde mit zwei Personen durchgeführt. Beide Personen waren an der Entwicklung der SystemTestPortals beteiligt und haben schon mit den verwendeten Systemen gearbeitet. Ebenso haben sie Erfahrung in der längeren Entwicklung im Team. Der Zielgruppe der Integration sind beide Systeme ebenso bekannt. Aus diesem Grund eignen sie sich als Probanden für die Evaluation.

6.2 Aufbau der Evaluation

Um die Integration in Ruhe testen zu können, waren die Probanden einzeln in einem Raum. Außer dem Proband war nur der Instruktor anwesend. Dies entspricht einer kreativen Arbeitsatmosphäre, welche bei einer realen Durchführung ebenso gegeben sein sollte.

Als Arbeitsgerät wurde ein Laptop mit Internetzugang verwendet. Der Internetzugang wurde für den Zugriff auf GitLab gebraucht. Weitere Geräte wurden für die Evaluation nicht benötigt.

6.3 Ablauf des Experimentes

Die Evaluation der Integration wurde in drei Schritten ausgeführt. Nach einer kurzen Einführung in das System führten die Probanden Schritt für Schritt Aufgaben durch, welche auch beim Veröffentlichen und Testen einer neuen Softwareversion in der Praxis anfallen würden. Dabei sind verschiedene Stakeholder wie Entwickler, Tester und Test-Manager beteiligt. Um einen Überblick über die Aufgaben jedes Stakeholders zu

bekommen, nahmen die Probanden nacheinander diese Rollen an. Somit konnten sie aus der Sicht jedes Stakeholders reflektieren und den Arbeitsfluss des manuellen Testens in der CI/CD bewerten.

Um einen realen Hintergrund zu schaffen, war die Aufgabe das Testen der noch unveröffentlichten Version 1.7.0 des SystemTestPortals.

6.3.1 Konfiguration der Pipeline

Die Konfiguration der CI/CD-Pipeline wird von den Entwicklern durchgeführt, weswegen der Proband zuerst aus Sicht eines Entwicklers agierte.

Die Aufgabe dabei war, aus der Dokumentation der Integration die Konfiguration für die CI/CD-Pipeline in ein GitLab-Projekt zu übernehmen. Das GitLab-Projekt repräsentiert das zu testende System (hierbei das SystemTestPortal). Um die Verständlichkeit der CI-Konfiguration für den Endnutzer zu überprüfen, sollten die Probanden die Konfiguration erklären. Die Erfahrung mit der CI/CD von GitLab hat aufgrund der spezifischen Syntax jedoch einen großen Einfluss auf das Verständnis der Konfiguration.

6.3.2 Veröffentlichung einer neuen Version

Um die Veröffentlichung einer neuen Version zu simulieren, sollten die Probanden einen neuen Tag in GitLab erstellen. Der Name des Tags sollte den Namen der zu testenden Version tragen, also 1.7.0.

Nach dem Setzen des Tags wird durch GitLab automatisch eine Pipeline gestartet. In der Stage des manuellen Testens wird der Webserver gestartet und kommuniziert mit dem SystemTestPortal.

6.3.3 Durchführung der manuellen Tests

Im SystemTestPortal wird automatisch die neue Version 1.7.0 erstellt. Nun soll der Proband als Test-Manager agieren. Dazu meldet er sich im SystemTestPortal als Test-Manager an. In der Aufgabenliste seines Profils sieht der Nutzer, dass er die Version den korrekten Testfällen zuweisen soll. Warum dies nicht möglich ist, wurde in Abschnitt 4.3.2 erklärt.

Nachdem diese Aufgabe als erledigt markiert wurde, wechselte der Proband die Rolle zum Tester.

Als Tester sollte er nacheinander die zugewiesenen Tests durchführen. Diese konnte er in der Aufgabenliste seines Profils abrufen und von dort aus direkt für die gegebene Version starten.

6.3.4 Protokolle in GitLab abrufen

Nachdem alle Tests ausgeführt wurden, beendete sich die Stage in der Pipeline automatisch und die Protokolle der Testausführungen wurden als Artefakte, also nach dem Beenden der Pipeline verfügbare Dateien, bereit gestellt. Der Proband sollte diese herunterladen und einsehen.

6.3.5 Beantwortung des Fragebogens

Dem praktischen Teil folgte die Befragung der Probanden. Der Fragebogen enthielt Skalenfragen und offene Fragen. Elemente wie die Komplexität der Konfiguration konnten auf einer Skala gemessen werden. Fragen wie z.B. nach Verbesserungsmöglichkeiten können jedoch durch freien Text besser beantwortet werden, da jeder Proband andere Möglichkeiten sieht.

6.4 Ergebnisse der Evaluation

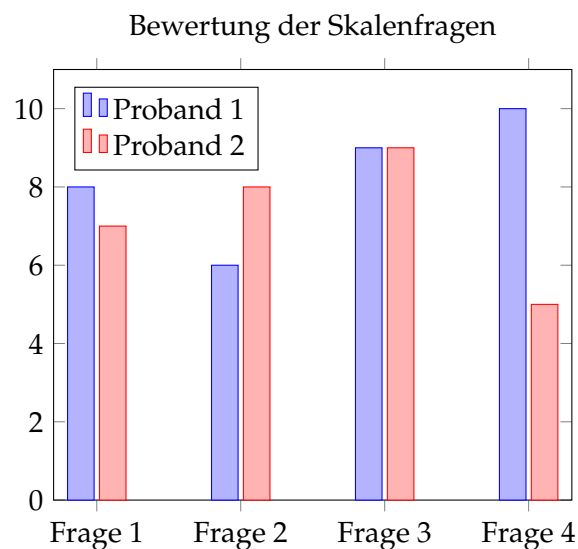


Abbildung 6.1: Bewertung der Fragen zur Benutzerfreundlichkeit

In Abbildung 6.1 sind die Einschätzungen der Probanden visualisiert. Jede Frage konnte auf einer Skala von Null bis Zehn bewertet werden.

Frage 1: Wie einfach war die Einbindung und Konfiguration des SystemTestPortals in die Continuous Integration/Continuous Deployment?

Die Frage nach der Komplexität der Konfiguration ist beeinflusst von der vorhandenen Erfahrung mit der GitLab-Ci, aber auch von Ähnlichkeiten zu der Integration anderer Systeme.

Laut der Probanden sei die Integration „vergleichbar zu anderen Systemen“. Andere Systeme verwenden für ihre Integration ebenso die Projektvariablen, die in der CI/CD-Pipeline abgerufen werden.

Frage 2: Für wie sinnvoll bewerten Sie den Arbeitsfluss (Workflow) der Integration?

Als größtes Problem bei der Integration wurde die Zuweisung der Version und Variante zu den anwendbaren Testfällen genannt. Die Zuweisung sei bei häufigen manuellen Tests zu aufwändig, da man jeden Testfall einzeln editieren und die neue Version/Variante auswählen muss.

Für eine bessere Benutzbarkeit kann ein Dialog direkt in der Aufgabenliste geöffnet werden. In diesem Dialog können die Versionen und Varianten von allen Testfällen auf einmal editiert werden. Die Darstellung dieses Dialogs muss den Nutzern jedoch verdeutlichen, wie die Versionen, Varianten und die Testfälle zusammenhängen. Die Darstellung davon ist auch ein Problem im aktuellen SystemTestPortal.

Alternativ zum manuellen Definieren der Anwendbarkeit wurde vorgeschlagen, automatisch allen Testfällen, die bereits für die vorherige Version ausführbar waren, die neue Version und Variante zuzuweisen. Dieser Ansatz wurde ergänzt und ist in der Vorstellung der Integration beschrieben.

Frage 3: Wie wahrscheinlich ist es, dass die Integration in der Entwicklung von Projekten verwendet wird, die auch das SystemTestPortal verwenden?

Beide Probanden halten es für sehr wahrscheinlich, dass Projekte, die das SystemTestPortal für manuelle Tests benutzen, auch die Integration in die CI/CD verwenden. Für den Einsatz ist aber besonders die Qualität der Integration und die des SystemTestPortals von Bedeutung. Behindert der Einsatz des SystemTestPortals den Entwicklungsfortschritt, wird es natürlich nicht mehr eingesetzt.

Frage 4: Wie stark verbessert das zusätzliche manuelle Testen die Qualität des Testsystems?

Qualität wurde in dieser Frage als Abwesenheit von Fehlern, aber auch als Erfüllung von nicht-funktionalen Anforderungen wie Barrierefreiheit, Benutzerfreundlichkeit oder Robustheit gesehen. Die abweichenden Werte bei der Bewertung sind durch das Argument gegeben, dass es sehr schwer abschätzbar ist.

6.4.1 Weitere Anmerkungen zur Integration

Freigabe der Testsuite

Die Freigabe der Testsuite zum Testen der neuen Version wird aktuell durch das Fertigstellen der Aufgabe des Test-Managers geregelt. Diese Aufgabe sieht jedoch nur der Test-Manager selbst. Tester beispielsweise haben keinen Einblick darauf, wie viele der Tests schon für die neue Version aktualisiert worden sind. Somit können sie nicht schon mit dem Testen beginnen, bevor der Test-Manager alle Tests aktualisiert hat.

Die Freigabe der Testsuite kann auf unterschiedliche Weise realisiert werden.

Freigabe jedes einzelnen Tests

Wenn eine neue Version im SystemTestPortal erstellt wird, wird der Status aller Testfälle auf „in Arbeit“ gesetzt. Dieser Status deutet darauf hin, dass der Testfall erst überprüft und gegebenenfalls aktualisiert werden muss. Betrachtet ein Test-Manager diesen Testfall als fertig, wird der Status auf „Ausführbar“ oä. gesetzt. Jetzt können Tester diesen Test ohne Bedenken ausführen, da sie wissen, dass er auf dem neuesten Stand ist.

Vorteil hierbei ist, dass mehrere Test-Manager in einem Team leicht zusammen arbeiten können. Jeder Test-Manager bearbeitet einen anderen Testfall und setzt diesen auf „Ausführbar“, sobald er damit fertig ist. Die anderen Test-Manager sehen sofort, dass sie diesen Testfall nicht mehr bearbeiten müssen.

Im Vergleich zur anderen Methode ist es potenziell jedoch ein größerer Aufwand, da jeder Testfall einzeln angeschaut werden muss. Um diesen Aufwand zu minimieren, könnte ein Dialog angeboten werden, bei dem durch Auswählen in einer Liste mehrere Testfälle kollektiv auf „Ausführbar“ gesetzt werden. In der Liste sollten jedoch auch Details wie die einzelnen Schritte oder Vorbedingungen sichtbar sein, da diese sich für die neue Version ändern könnten.

Freigabe der Version

Anstatt den Status jedes einzelnen Testfalls anzupassen, kann auch der Status einer Version angepasst werden. Nachdem eine neue Version erstellt wurde, müssen alle Testfälle manuell überprüft und aktualisiert werden. Sieht ein Test-Manager diese Version als zugewiesen an, kann er die Version auf „Ausführbar“ setzen.

Bei dieser Methode müssen sich Test-Manager mehr absprechen, da während dem Aktualisieren der Testfälle nicht ersichtlich ist, ob ein Testfall schon für die neueste Version angepasst worden ist oder nicht. Auch für einen einzelnen Test-Manager kann dies negativ sein, wenn er einen Testfall nicht komplett aktualisiert hat und unterbrochen wird. Beim Fortsetzen weiß er eventuell nicht mehr, dass er diesen Testfall noch fertig bearbeiten muss und die Tester stoßen beim Ausführen auf Probleme.

Detaillierte Rückmeldung für Entwickler

Des Weiteren sind detaillierte Rückmeldungen für die Entwickler interessant. Zusätzlich zur Meldung, dass noch nicht alle Tests ausgeführt wurden, kann z.B. die Prozentzahl der ausgeführten Tests angezeigt werden.

Im Falle eines internen Fehlers im SystemTestPortal kann die Fehlermeldung an die Entwickler weitergeleitet werden. Diese haben ein größeres Verständnis für den aufgetretenen Fehler und können diesen eventuell beheben, um die Integration erfolgreich fortzusetzen.

6.5 Weiterführende Studie zur Untersuchung der Auswirkung auf die Qualität

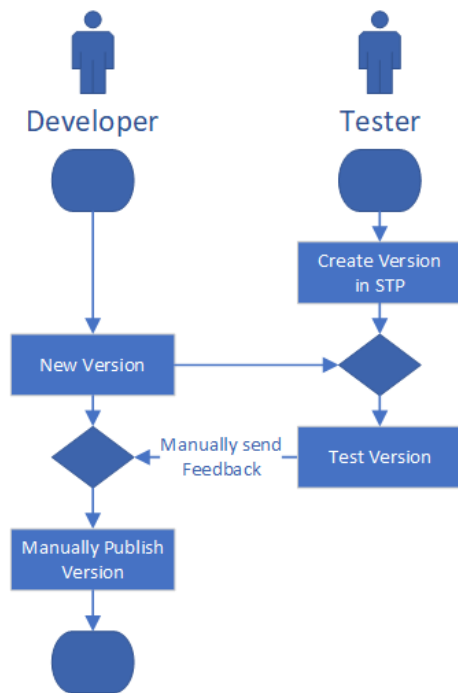
Um eine Änderung der Qualität konkret messen zu können, wäre eine Langzeitstudie sinnvoll. Diese Studie ist nach dem Within-Subject-Design aufgebaut, d.h. eine Gruppe testet alle Zustände. Die Gruppe ist hierbei ein Team an Entwicklern und Testern, die ein Testsystem entwickeln. Als Zustand gibt es die Entwicklung mit und ohne die Integration. Die Verwendung der Integration ist auch die unabhängige Variable. Änderungen des Ergebnisses können auf diese unabhängige Variable zurückgeführt werden.

In Abbildung 6.2 ist ein Vergleich der verschiedenen Arbeitsflüsse visualisiert. Ohne die Integration müssen die Tester und Entwickler viele Schritte manuell ausführen und auf die andere Partei warten. Auch die Rückmeldung der Testergebnisse muss manuell erfolgen. Bei der Verwendung der Integration werden viele Schritte automatisch ausgeführt und es entsteht ein kontinuierlicher Ablauf.

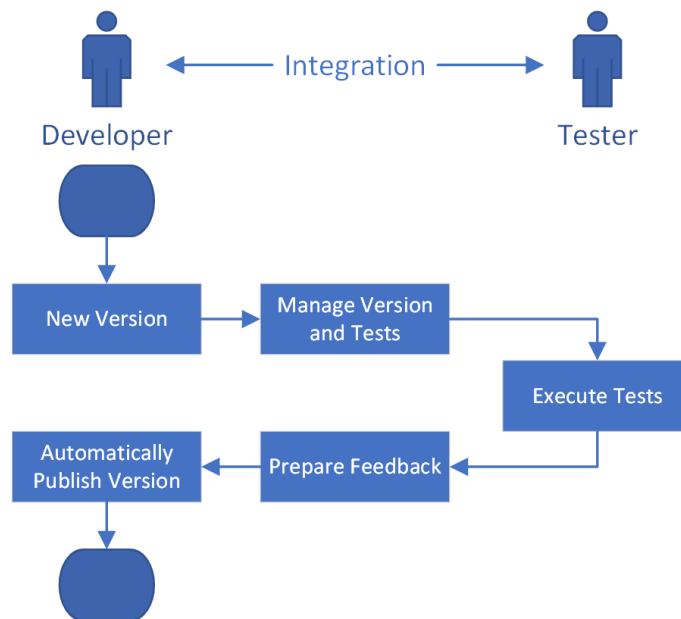
Als abhängige Variablen würden z.B. die Anzahl der Bugs vor und nach dem Einführen der Integration verglichen werden.

Außerdem kann das Nutzerfeedback zum Testsystem vor und nach der Integration verglichen werden. Das ist jedoch sehr subjektiv und nicht durch quantitative Methoden messbar.

Im Gegensatz zu einem Between-Subjects-Design (oder auch Between-Group-Design) hat die Verwendung einer Gruppe den Vorteil, dass maßgebliche Störvariablen wie die unterschiedliche Qualifizierung der Versuchspersonen reduziert werden.



(a) Ablauf des Testens ohne Integration



(b) Ablauf des Testens mit Integration

Abbildung 6.2: Vergleich des Ablaufs mit und ohne Integration

6.6 Probleme und deren Lösung während der Ausführung der Integration

Während dem Testen der Integration mit den Probanden traten unerwartete Probleme auf. Diese hingen mit den zwei Systemen zusammen, die integriert wurden. Das SystemTestPortal und GitLab.

In GitLab wurde die Pipeline aufgrund eines Fehlers von GitLab beendet [18]. Dieses Problem wird eventuell in einer späteren Version von GitLab behoben. Sollte dieser Fehler auftreten, kann die Pipeline erneut manuell gestartet werden und die manuellen Tests werden weiter ausgeführt.

```
$ ./stp-ci-integration --url="$STP_URL" --version="$CI_COMMIT_TAG" --variant="$STP_VARIANT" --token="$STP_TOKEN" --time="30s"
2018/09/04 14:06:18 not all tests executed yet
2018/09/04 14:06:48 not all tests executed yet
2018/09/04 14:07:18 not all tests executed yet
Pulling docker image sha256:29a456dab9652b6456d2b9f87d7b743a05c0c2ea4ac0fc7e2ce06e2e0168a180 ...
ERROR: Job failed: exit code 137
```

Abbildung 6.3: Abbruch der Pipeline aufgrund eines Pipeline-Fehlers

Besonders hinderlich waren Bugs im SystemTestPortal, die den Arbeitsfluss behinderten. Beispielsweise wurde beim Auswählen der Versionen und Varianten für einen Testfall die Version als ausgewählt angezeigt, aber nicht gespeichert. Nutzer müssen sich auf die Konformität der angezeigten und gespeicherten Daten verlassen können. Fehler wie diese können durch manuelle Tests schnell entdeckt werden und Entwickler können in den Protokollen herausfinden, in welchem Testschritt der Fehler auftrat.

Zusätzlich zu den Mängeln in der Benutzbarkeit trat ein interner Fehler bei der Erstellung der Testprotokolle auf. Das hatte zur Folge, dass der Docker-Container nach zehn erfolglosen Versuchen, die vorhandenen Ergebnisse abzurufen, beendet und die CI-Pipeline abgebrochen wurde.

Dadurch ist es nicht möglich, eine neue Version des Testsystems zu veröffentlichen, auch wenn eigentlich alle Tests erfolgreich abgeschlossen waren.

Statt einer Fehlermeldung aufgrund der Protokolle könnte der Status der Tests zurückgegeben werden, ohne die Testprotokolle an die Antwort des SystemTestPortals anzuhängen. Dann muss der Entwickler aber benachrichtigt werden, aus welchem Grund keine Protokolle verfügbar sind.

7 Herausforderungen und Ergebnisse der Umsetzung

In diesem Kapitel wird auf Herausforderungen während des Prozesses der Erstellung der Arbeit eingegangen. Um eine bessere Transparenz gegenüber der Arbeit des parallelen Studienprojektes herzustellen, wird hier auch beschrieben, welche Funktionen des SystemTestPortals im Zuge dieser Arbeit entstanden sind.

7.1 Herausforderungen bei der Entwicklung der Integration

7.1.1 Recherche und Besonderheit der Integration

Der erste Schritt dieser Arbeit umfasste das Recherchieren von ähnlichen Themen. Dabei wurde zuerst nach Applikationen gesucht, die, wie das SystemTestPortal, Unterstützung bei der Erstellung und Durchführung manueller Systemtests bieten. Der Horizont dieser Applikationen ist jedoch begrenzt, da sich viele Programme dem Trend hingeben und ausschließlich auf automatisiertes Testen ohne Einbezug von manuellen Tests beschränkt sind. Viele dieser Applikationen unterstützen neben den manuell ausführbaren Tests die Einbindung automatisierter Testskripte. Verwaltungsprogramme von automatisierten Tests integrieren im Gegensatz dazu jedoch keine manuellen Tests.

Aus diesem Grund ist diese Art der Integration einer Manuellen-Systemtest-Applikation in das Continuous Deployment bisher einzigartig. Das kann für das SystemTestPortal ein hilfreicher Vorteil gegenüber Konkurrenzsystemen sein. Auf der anderen Seite gab es während des Entwurfes nicht die Möglichkeit, negative Gesichtspunkte bereits existierender Systeme herauszuarbeiten und diese besser umzusetzen.

7.1.2 Einwirkungsfaktoren auf die Umsetzung

Nach der Recherche begann die Erstellung des Entwurfs mit einem groben Umriss, welche Funktionalitäten für eine angemessene Integration in dieser Arbeit umgesetzt werden können. Die Anwendungsfälle wurden anschließend evaluiert um zu entscheiden, wie diese umgesetzt werden können und welche Umsetzungsart die meisten Vorteile bietet. Im Vordergrund stand dabei eine simple Integration für Endnutzer, wobei dennoch eine hohe Flexibilität bezüglich der Einstellungen und eine leichte Erweiterung gegeben sein sollten. Aus diesem Grund wurde letztendlich auch eine eigene Applikation entwickelt statt Webhooks zu nutzen.

Um den Datenverkehr zu minimieren, sollte die Webapplikation in der Pipeline anfangs mit einer einzigen Anfrage an die Instanz des SystemTestPortals arbeiten. Das SystemTestPortal speicherte diese Anfrage und meldete erst bei vollständigen Tests das Ergebnis zurück. Aufgrund der Tatsache, dass Applikationen in einer Pipeline keine Anfragen entgegen nehmen können, wurde die Webapplikation so umgesetzt, dass sie wiederholt Anfragen an das SystemTestPortal sendet.

Die Konfiguration der Pipeline gestaltete sich als eine Herausforderung, da in GitLab beispielsweise im Abschnitt der Stage, in der das Skript definiert wurde, intern andere Pfade verwendet wurden als für die Speicherung der Artefakte. Bei der Definition des Pfades der zu speichernden Artefakte hatte man jedoch keinen Zugriff auf den Pfad, in dem das Skript ausgeführt wurde. Deswegen musste zuerst die Webapplikation im Dateisystem verschoben werden, damit die Testprotokolle, die als Artefakte gespeichert werden, von GitLab gefunden werden können.

Die Optimierung der Größe des Docker-Images bedurfte ebenso vieler Anpassungen und Tests. Anstelle des rohen Sourcecodes, der dann bei jeder Ausführung der Pipeline erst kompiliert werden musste, bot sich die direkte Bereitstellung der Binärdatei der Webapplikation an. Dafür mussten jedoch die korrekten benötigten Programme und Befehle in der Konfiguration des Docker-Images bereit gestellt werden.

7.2 Ergebnisse der Umsetzung

Um einen transparenten Überblick zu schaffen, zeigt Tabelle 7.1 eine Übersicht über alle Entwicklungen, Implementierungen und sonstige Ergänzungen, die im Rahmen dieser Arbeit entstanden sind.

Umgesetzte Funktion	Aufgabe
Testzuweisung mit auswählbarer Version und Variante	Zuweisung von Tests an Nutzer mit Auswahl der Testversion und Variante
Aufgabenliste für Nutzer im STP	Benachrichtigung und Auflistung von offenen Testdurchführungen mit vordefinierter Version und Variante
Projekttrollen im STP	Spezifizierung der Rolle von Projektmitgliedern zur automatischen Zuweisung der Aufgaben an die Mitglieder
Automatische Erstellung der Version und Variante	Erzeugung der zu testenden Version/Variante zur Reduzierung des Verwaltungsaufwandes
Automatische Zuweisung der Anwendbarkeit	Zuweisung der Testversion zu den Testfällen zur Definition der Anwendbarkeit
Automatische Zuweisung der Tests	Zuweisung der Tests einer Version zu den Testern im Projekt
Analyse der Testergebnisse	Ermittlung des Ergebnisses für die Pipeline
Bereitstellung der Testprotokolle in einem ZIP-Archiv	Detaillierte Informationen für die Entwickler zu der Testausführung
Automatische Benachrichtigung der Entwickler bei fehlgeschlagenen Tests	Schnelle Rückmeldung von Fehlern zur Planung in der Entwicklung
Webapplikation zur Kommunikation mit dem STP	<ul style="list-style-type: none"> • Verarbeitung der Eingabeparameter • Erzeugung der Anfrage an das STP • Empfangen und Verarbeiten der Antwort vom STP • Vorbereitung und Auslieferung des Feedback für die Entwickler
Konfiguration des Docker-Images	<ul style="list-style-type: none"> • Bereitstellung der ausführbaren Webapplikation • Einfache Einbindung in die CI/CD-Pipeline
Konfiguration der CI/CD-Pipeline	<ul style="list-style-type: none"> • Beispielkonfiguration der Pipeline in GitLab • Führt manuelle Tests bei Erstellung einer neuen Version aus • Speichert Testprotokolle als Artefakt

Tabelle 7.1: Übersicht über die umgesetzten Funktionen

Damit ist die Grundlage geschaffen um ein Zusammenwirken von automatischen Testläufen und kreativen manuellen Tests zu verwalten und zu koordinieren.

8 Zusammenfassung und Ausblick

Diese Arbeit hat einen Ansatz zur Integration von manuellem Testen in die agile Entwicklung und den kontinuierlichen Releaseprozess vorgestellt und für den Einsatz in die Praxis umgesetzt. Das SystemTestPortal ist die Basis für die Ausführung der manuellen Tests und spielt eine große Rolle in der Benutzerakzeptanz der Integration. Die Umsetzung zielt auf eine leichte Einbindung des SystemTestPortals in die CI/CD-Pipeline ab, ohne Abstriche in der Flexibilität machen zu müssen. Neben der automatischen Benachrichtigung von Test-Managern und Testern im SystemTestPortal bietet die Integration umfangreiches und schnelles Feedback für die Entwickler an, um diese bei der Fehlersuche zu unterstützen. In der Evaluation des entwickelten Ansatzes wurde die Benutzerfreundlichkeit und der Arbeitsfluss untersucht und eine weiterführende Langzeitstudie zur Untersuchung der Qualitätsänderungen durch die Anwendung der Integration vorgestellt. Für das SystemTestPortal steigt die Konkurrenzfähigkeit durch die Einbindung in den kontinuierlichen Entwicklungsprozess.

8.1 Erweiterungsmöglichkeiten und Ausblick

Der in dieser Arbeit entwickelte Ansatz bietet Möglichkeiten zur Einbindung weiterer Personengruppen oder Erweiterung der bereitgestellten Informationen und Einstellungsmöglichkeiten. Diese werden im Folgenden beschrieben.

8.1.1 Aktuelle Meldung des Testfortschritts

In der Detailansicht einer aktuell ausgeführten Stage (dabei spricht man von einem Job) gibt die Webapplikation Rückmeldung, ob die Tests noch ausgeführt werden oder beendet sind. Darin können auch weitere Informationen wie der prozentuale Fortschritt der Testausführung bereitgestellt werden. Diese Informationen können vom SystemTestPortal zu dessen Antwort beigefügt werden.

8.1.2 Einstellung der Akzeptanzkriterien für erfolgreiche Tests

Anstelle des aktuell definierten Kriterium für eine erfolgreiche Rückmeldung an die Pipeline können die Projekteinstellungen im SystemTestPortal um weitere auswählbare Kriterien wie in Abschnitt 3.5 beschrieben erweitert werden. Je nach Entwicklungsteam wollen die Entwickler eine andere Definition von erfolgreichen Tests. Anstelle ausschließlich erfolgreicher Tests kann dadurch nur eine bestimmte Prozentzahl erfolgreicher Tests gefordert werden.

8.1.3 Benachrichtigung weiterer Stakeholder

Zusätzlich zu einer Rückmeldung der Testergebnisse für die CI/CD-Pipeline ist ein detailliertes Protokoll der Tests interessant für Personen, die nicht am manuellen Testvorgang teilgenommen haben wie Kunden oder Vorgesetzte. Dies führt das manuelle Testen und das Continuous Development weiter zusammen. Von Vorteil wäre hier die Verfügbarkeit des Testprotokolls ohne Zugriff auf das SystemTestPortal und das CI/CD-System zu benötigen. Für eine schnelle und für die Empfänger komfortable Übermittlung des Reports eignet sich die Sendung per E-Mail.

Die Benachrichtigungen über die Testergebnisse sollten möglichst zeitnah gesendet werden. Am besten ist hierbei das sofortige Senden nachdem die Testergebnisse der neuen Version ausgewertet sind. Im Report sollte ein umfangreiches Bild der Testergebnisse zu finden sein. Für einen schnellen Überblick ist eine Übersichtsseite mit Diagrammen der gesamten Ergebnisse von Vorteil.

8.2 Weiterführende Forschungsthemen

Die Integration von manuellem Testen in die kontinuierliche Entwicklung und die Automatisierung einiger Teile des manuellen Testprozesses zeigte Herausforderungen auf, die nicht in dieser Arbeit absolut zufriedenstellend gelöst werden konnten.

8.2.1 Ermittlung der Anwendbarkeit von manuellen Tests

In der Evaluation dieses Ansatzes zur Integration von manuellen Tests wurde die Zuweisung der Anwendbarkeit von Testfällen zur neuen Version als Schwachpunkt des Prozesses erkannt. Sie erfordert ein Eingreifen und Überprüfen durch Test-Manager und verzögert das Testen. Um die korrekten Testfälle zu bestimmen, die auf eine Systemversion anwendbar sind, muss herausgefunden werden, welcher Anwendungsfall durch einen Testfall getestet wird und ab welcher Version dieser Anwendungsfall unterstützt wird.

8.2.2 Vollständige Testabdeckung durch manuelle Tests

Automatisierte Tests streben oftmals eine möglichst hohe Testabdeckung an. Jeder Anwendungsfall eines Systems soll mit verschiedenen Eingaben getestet werden. Besonders wichtig sind die kritischen Anwendungsfälle. Sie werden von den Nutzern am häufigsten ausgeführt und müssen reibungslos funktionieren. Um die Testabdeckung von manuellen Tests bestimmen zu können, müssen die Definitionen der Anwendungsfälle auf die Testfälle abgebildet werden. Dafür ist eine generische Lösung zur Anwendung mit unterschiedlichen Systemen von Vorteil.

8.2.3 Definition der Qualität von Testfällen

Die manuellen Tests werden nicht nur von Menschen ausgeführt, sondern auch von Menschen erstellt. Aus diesem Grund unterliegt die Erstellung den gleichen Herausforderungen wie die Ausführung. Persönliche Unterschiede wirken sich auf das Design der Testfälle aus. Das reicht vom Detailgrad bis zur Verständlichkeit des Tests. Um konsistente Testfälle mit einer hohen Qualität zu schaffen, muss diese objektiv gemessen werden können. Eine hohe Qualität spiegelt sich beispielsweise in der Dauer der Testausführung und der daraus folgenden Kostenersparnis wieder.

8.2.4 Automatische Generierung von manuellen Tests

Die Erstellung und Wartung von Anleitungen zum manuellen Testen ist das größte Aufgabengebiet der Test-Manager. Um diese Arbeit zu erleichtern, wäre eine automatische Erstellung von manuellen Testfällen interessant. Diese müssten aus den Anforderungen des Systems abgeleitet werden. Schwierig hierbei ist die korrekte Ableitung der Testschritte, da sich der Arbeitsablauf zum Testen eines Anwendungsfalls von Version zu Version unterscheiden kann.

Literaturverzeichnis

- [1] L. Ash. *The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests*. Wiley technology publishing : timely, practical, reliable. Wiley, 2003. ISBN: 9780471430216. URL: books.google.de/books?id=tUYo644-SYcC (zitiert auf S. 26).
- [2] G. Booch. *Object-oriented Analysis and Design with Applications (2Nd Ed.)* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994. ISBN: 0-8053-5340-2 (zitiert auf S. 21).
- [3] *Comparison of Continuous Integration Software*. URL: en.wikipedia.org/wiki/Comparison_of_continuous_integration_software (zitiert auf S. 26).
- [4] *Docker Registry*. URL: docs.docker.com/registry/ (zitiert auf S. 47).
- [5] *DuckDuckGo Search Engine*. URL: duckduckgo.com/ (zitiert auf S. 23).
- [6] A. Eck, F. Uebernickel, W. Brenner. „Fit for Continuous Integration: How Organizations Assimilate an Agile Practice“. In: *20th Americas Conference on Information Systems, AMCIS 2014*. Aug. 2014, S. 11 (zitiert auf S. 30).
- [7] *frolvlad/alpine-glibc*. URL: hub.docker.com/r/frolvlad/alpine-glibc/ (zitiert auf S. 47).
- [8] *Generic Webhook Trigger Plugin*. URL: wiki.jenkins.io/display/JENKINS/Generic+Webhook+Trigger+Plugin (zitiert auf S. 28).
- [9] *Get started, Part 1: Orientation and Setup*. URL: docs.docker.com/get-started/#images-and-containers (zitiert auf S. 47).
- [10] *Getting started with Review Apps*. URL: docs.gitlab.com/ee/ci/review_apps/ (zitiert auf S. 28).
- [11] *GitLab CI/CD Variables*. URL: docs.gitlab.com/ee/ci/variables/ (zitiert auf S. 49, 50).
- [12] *GitLab Pipelines*. URL: docs.gitlab.com/ee/ci/img/pipelines.png (zitiert auf S. 28).
- [13] *GNU General Public License v3.0*. URL: choosealicense.com/licenses/gpl-3.0/ (zitiert auf S. 22).
- [14] *Important Software Test Metrics and Measurements – Explained with Examples and Graphs*. URL: softwaretestinghelp.com/software-test-metrics-and-measurements/#Examples_of_Software_Testing_Metrics (zitiert auf S. 29).
- [15] *Introduction to pipelines and jobs*. URL: docs.gitlab.com/ee/ci/pipelines.html#pipelines (zitiert auf S. 34).
- [16] *Jenkins Plugins*. URL: plugins.jenkins.io/ (zitiert auf S. 27).
- [17] D. F. Jez Humble. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. (Zitiert auf S. 29).
- [18] *Job failed: exit code 137*. 14. Juli 2017. URL: gitlab.com/gitlab-com/infrastructure/issues/2289 (zitiert auf S. 68).

- [19] *Mainfram DevOps*. URL: compuware.com/lifecycle-overview/ (zitiert auf S. 27).
- [20] *Package rand*. URL: golang.org/pkg/crypto/rand/ (zitiert auf S. 41, 49).
- [21] A. G. Paul M. Duvall Stephen M. Matyas. *Continuous Integration: Improving Software Quality and Reducing Risk (Addison-Wesley Signature Series (Fowler))*. (Zitiert auf S. 29).
- [22] *Polling*. URL: enacademic.com/dic.nsf/enwiki/2208811 (zitiert auf S. 43, 51).
- [23] *Review Apps*. URL: about.gitlab.com/features/review-apps/ (zitiert auf S. 36).
- [24] M. Shahin, M. A. Babar, L. Zhu. „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices“. In: *IEEE Access* 5 (2017), S. 3909–3943. DOI: [10.1109/ACCESS.2017.2685629](https://doi.org/10.1109/ACCESS.2017.2685629) (zitiert auf S. 30).
- [25] *Software Testing Metrics: Complete Tutorial*. URL: guru99.com/software-testing-metrics-complete-tutorial.html (zitiert auf S. 29).
- [26] D. Ståhl, J. Bosch. „Modeling continuous integration practice differences in industry software development“. In: 87 (Jan. 2014), S. 48–59 (zitiert auf S. 30).
- [27] *SystemTestPortal*. URL: gitlab.com/stp-team/systemtestportal-webapp (zitiert auf S. 22, 28).
- [28] *SystemTestPortal Homepage*. URL: systemtestportal.org/ (zitiert auf S. 22).
- [29] *Top 15 Best Test Management Tools in 2018 (Our Reviews)*. 3. Juli 2018. URL: softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/ (zitiert auf S. 26).
- [30] *Webhooks*. URL: developer.github.com/webhooks/ (zitiert auf S. 28).
- [31] *Webhooks*. URL: docs.gitlab.com/ee/user/project/integrations/webhooks.html (zitiert auf S. 28, 37).
- [32] *Webhooks Secret Token*. URL: docs.gitlab.com/ce/user/project/integrations/webhooks.html#secret-token (zitiert auf S. 40).
- [33] *What is a WebHook?* URL: webhooks.pbworks.com/w/page/13385124/FrontPage (zitiert auf S. 28).

Alle URLs wurden zuletzt am 10. 10. 2018 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift