HOCHSCHULE RAVENSBURG-WEINGARTEN

BACHELORARBEIT

Vergleich von Workflow-Management-Systemen unter Verwendung der 'Anwendbarkeit von Software-Engineering-Konzepten' als Vergleichskriterium

Verfasser:
Bianca-Maria Braun
28857

Gutachter:
Prof. Dr. rer. nat. Marius Hofmeister
Philipp Hehnle M.Sc. (bamero AG)

Thesis zur Erlangung des akademischen Grades
Bachelor of Science
im Studiengang
Angewandte Informatik
der Fakultät Elektrotechnik/Informatik

18. März 2021

HOCHSCHULE RAVENSBURG-WEINGARTEN

Zusammenfassung

Fakultät Elektrotechnik und Informatik
Angewandte Informatik

Bachelor of Science

Vergleich von Workflow-Management-Systemen unter Verwendung der 'Anwendbarkeit von Software-Engineering-Konzepten' als Vergleichskriterium

von Bianca-Maria Braun

Die Nachfrage an Individualsoftware zur Digitalisierung von Geschäftsprozessen steigt stetig. Um häufig verwendete Funktionen in der Geschäftsprozessdigitalisierung nicht ständig neu entwickeln zu müssen, werden bei der Digitalisierung von Geschäftsprozessen Workflow-Management-Systeme eingesetzt. Daher ist es sinnvoll zu erforschen, wie gut sich gängige Konzepte des Software-Engineerings in die Entwicklung mit Workflow-Management-Systemen integrieren lassen. Durch diese Paradigmen können eine hohe Qualität der Software und eine kontinuierliche Bereitstellung an den Kunden erreicht werden.
In dieser Arbeit wird untersucht, wie gut sich Prinzipien und Vorgehensweisen des modernen Software-Engineerings in die Softwareentwicklung mit Workflow-Management-Systemen integrieren lassen. Um dies detailliert zu untersuchen, wurde ein Prozess in verschiedenen Workflow-Management-Systemen implementiert und Software-Engineering-Konzepte auf diese Implementierung angewandt. Die Auswertung zeigt, dass es große Unterschiede in den Ansätzen der einzelnen Workflow-Management-Systeme gibt. Anhand dieser Auswertung wurde eine Entscheidungshilfe in Form einer Tabelle entworfen, um auf aufkommende Projektanforderungen mit dem richtigen System reagieren zu können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

BPM	**B**usiness **P**rocess **M**anagement
BPMN	**B**usiness **P**rocess **M**odel and **N**otation
UML	**U**nified **M**odeling **L**anguage
XML	E**X**tensible **M**arkup **L**anguage
REST	**RE**presentational **S**tate **T**ransfer
API	**A**pplication **P**rogramming **I**nterface
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
YAML	**Y**AML **A**in't **M**arkup **L**anguage
JSON	**J**ava**S**cript **O**bject **N**otation

Glossar

Annotation Annotationen enthalten Metadaten. Diese Annotationen werden zu Java-Programmelementen hinzugefügt und während des Build-Vorgangs oder zur Laufzeit ausgewertet.[1]. 43, 52

API API ist die Abkürzung für Application Programming Interface. Dabei handelt es sich um eine Softwareschnittstelle, auf die ein Anwendungsprogramm verweisen kann, um Zugriff auf spezielle Dienste zu erhalten.[2]. 9, 13, 14, 28, 36, 53

Application-Server Auf einem Application-Server, auch Anwendungsserver genannt, kann Software bereitgestellt werden. Der Server bietet einen Konfigurationsbereich um die Anwendung je nach Umgebung anzupassen ohne den Code dafür ändern zu müssen.[3]. 27, 28

Archetype Unter einem Archetype versteht man in der Softwareentwicklung ein vorgefertigtes Projektkonstrukt mit allen nötigen Dateien.[4]. 28, 29, 45, 46

deploy Unter deployen oder einem Deployment versteht man die Bereitstellung von Software.[5]. 3, 28–31, 36, 46, 48

Docker Docker ist ein System, welches zur Virtualisierung auf der Anwendungsebene dient. Docker-Container werden üblicherweise zur Bereitstellung spezifischer Umgebungen zum Testen von Software oder auch für das Ausführen von Software im Produktivbetrieb verwendet.[6]. 9, 40, 41, 43, 44, 48

Fork Fork kommt aus dem Englischen und bedeutet Gabel beziehungsweise Gabelung. In der Softwareentwicklung versteht man darunter das Kopieren des Quellcodes einer Software, um eine eigenständige Version davon zu erstellen, welche komplett losgelöst vom Ursprungsprojekt gepflegt wird.[7] . 13, 14, 22

Jenkins In Jenkins können JUnit-Testfälle beim Einchecken neuer Änderungen in die Versionsverwaltung ausgeführt werden. Nach erfolgreichem Abschluss dieser Tests kann Jenkins die Applikation Bauen und auf einer definierten Umgebung bereitstellen.[8]. 46, 48, 50, 52

JSON Die JavaScript-Object-Notation (kurz JSON) ist ein Datenformat um JSON-Objekte als für Menschen lesbare Liste von Name-/Wert-Paaren darzustellen. Das JSON-Format ist dabei sprachunabhängig verwendbar.[9]. 10, 28, 30, 40, 42, 51, 53

Microservice Microservices sind kleinstmögliche Anwendungen, welche untereinander kommunizieren, um einen größeren Anwendungsfall abzudecken. Dies trägt dazu bei, dass jede dieser Anwendungen bestmöglich skalierbar ist.[10]. 15, 30, 31, 40, 41, 48, 49

Node.js Node ist eine Open Source-Umgebung, welche das serverseitige Ausführen von JavaScript ermöglicht. Dadurch wird die Möglichkeit geboten, dieselbe Sprache für client- und serverseitige Programmierung zu verwenden.[11]. 31, 35, 42, 48

REST REST ist ein Architekturparadigma für den Entwurf von APIs. Dieses Paradigma befasst sich mit den Ressourcen eines Systems und der Art und Weise, wie die Zustände dieser Ressourcen über HTTP adressiert und übertragen werden.[12]. 13, 15, 28, 30, 36, 40, 46, 48, 51, 54, 55

Vue.js Vue.js ist ein progressives JavaScript-Framework für die Erstellung von Benutzeroberflächen.[13]. 23, 31, 32, 42

XML XML ist eine Beschreibungssprache, die es ermöglicht, eigene Vokabulare mit benutzerdefinierten Tag-Namen und Regeln einzuführen.[9] . 6, 9, 10, 29, 32, 37, 45, 46, 48, 51

Vorwort

Die vorliegende Bachelorarbeit entstand im Rahmen meines Studiums der angewandten Informatik an der Hochschule Ravensburg-Weingarten. Die Forschungsfrage wurde in Zusammenarbeit mit der in Konstanz ansässigen bamero AG verfasst. Ich habe dieses Thema gewählt, um zum Abschluss meines Studiums noch einen mir bisher unbekannten Themenbereich zu erkunden.
Ich möchte mich an dieser Stelle bei der bamero AG, insbesondere meinem Betreuer Philipp Hehnle M.Sc., für die Unterstützung in der Anfertigung dieser Arbeit bedanken. Ein besonderer Dank gilt ebenfalls Herrn Prof. Dr. rer. nat. Hofmeister, für die wissenschaftliche Betreuung dieser Arbeit.
Auch danke ich meiner Familie, die mich während des gesamten Studiums unterstützt hat.

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Konstanz, den 18.03.2021 B.-M. Braun

Ort, Datum Unterschrift

Kapitel 1

Einleitung

In diesem Kapitel wird die Motivation der Themenwahl erläutert. Außerdem wird dargestellt, warum diese vorgenommene Untersuchung relevant ist.

1.1 Motivation

Abläufe (im folgenden auch Prozesse genannt) im Betrieb sind oft komplex und beinhalten mehrere Parteien. Das Prozessmanagement setzt an diesem Punkt an und versucht, diese Prozesse zu dokumentieren, strukturieren, vereinheitlichen und zu vereinfachen.[14]
Diese Arbeit wurde in Zusammenarbeit mit der in Konstanz ansässigen bamero AG verfasst. Das Unternehmen beschäftigt sich seit 2013 mit der Digitalisierung von Geschäftsprozessen in Unternehmen. Unter dem Leitgedanken „Jeden Tag verschwenden Menschen ihre Zeit mit veralteten und ineffizienten Prozessen.“ unterstützt die bamero AG Unternehmen bei der Analyse von Geschäftsprozessen und bildet diese als individuelle Softwarelösungen ab.

Ein Mittel zur Digitalisierung und Verbesserung eines Prozesses ist ein Workflow-Management-System. Mithilfe dieser Systeme lässt sich die Effizienz der Abläufe, insbesondere derer, an denen mehrere Parteien beteiligt sind, enorm steigern.[15]
Werden Prozesse mit Workflow-Management-Systemen als Workflow abgebildet, lässt sich die Digitalisierung effizient gestalten. Workflow-Management-Systeme stellen Funktionen zur Verfügung, die bei der Prozessdigitalisierung ansonsten ständig neu entwickelt werden müssten (Rechteverwaltung, Aufgabenverwaltung). Eine weitere Eigenschaft von Workflow-Management-Systemen ist es, modellierte Prozesse ausführen zu können.[16] Damit ist es möglich, dem Auftraggeber die fachliche Logik visuell darzustellen, was bei reinem Quellcode nicht möglich ist. Dadurch verringert sich der Gap zwischen Fachbereich und Entwicklung.

Diese Systeme existieren in vielen verschiedenen Varianten. Es gibt Baukastensysteme, die ohne Programmierung also lediglich mit Modellen auskommen, aber auch Systeme, die sich mit Eigenentwicklungen kombinieren lassen.[17]
Bei einfachen Prozessen bieten diese Baukastensysteme Vorteile, da auch Nicht-Entwickler Prozesse digitalisieren können und die gesamte Logik als Modell abgebildet wird. Der Vorteil eines Baukastensystems ist aber zugleich auch sein Nachteil. In Baukastensystemen können Modelle nur schwer komplexe Logiken abbilden, insbesondere wenn diese Modelle ausführbar sein sollen. Wenn es darum geht, große und komplexe Geschäftsprozesse zu digitalisieren, stößt man mit Baukastensystemen an Grenzen und Individualentwicklung ist unausweichlich. Dass

die Digitalisierung von großen und komplexen Prozessen klassische Software Entwicklung darstellt, wird oft unterschätzt, da oftmals das Bild der Baukastensysteme die Digitalisierung von Prozessen prägt.[16]
Daraus folgt, dass auch in Workflow-Management-Systemen, die grundsätzlich Individualentwicklung zulassen, Konzepte und Prinzipien, die sich im klassischen Software-Engineering längst als Standard etabliert haben, nicht anwendbar sind.
Die Anwendbarkeit von Software-Engineering-Konzepten in großen Softwareprojekten ist jedoch essenziell für eine hohe Qualität der entwickelten Software[5]. Beispiele für anwendbare Konzepte sind die Nutzung einer Versionsverwaltung, um die Arbeit im Team zu vereinfachen[18]. Außerdem wird mit Continuous Integration und Continuous Delivery durch das automatisierte Ausführen von Tests, das automatische Bauen und deployen das Risiko minimiert, dass fehlerhafte Software in das Produktivsystem gelangt[19][20].

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist es, einen Vergleich verschiedener Workflow-Management-Systeme zu erstellen. Die Grundlage dafür bieten Kriterien, um zu überprüfen, wie gut sich Software-Engineering-Konzepte auf diese Systeme anwenden lassen. Im Rahmen der vorliegenden Arbeit wird eine Marktanalyse durchgeführt und eine Auswahl der zu untersuchenden Systeme getroffen. Anschließend werden Kriterien ausgearbeitet und diese auf die einzelnen Systeme angewandt. Die Arbeit soll auf folgende Frage eine Antwort finden:

- Inwieweit lassen sich klassische Software-Engineering-Konzepte auf die Entwicklung mit unterschiedlichen Workflow-Management-Systemen anwenden?

Im Rahmen dieser Untersuchung wird überprüft, wie gut die ausgewählten Workflow-Management-Systeme eine Integration von Software-Engineering-Konzepten unterstützen. Wie in Abschnitt 1.1 beschrieben, kann es in großen Projekten mit mehreren Entwicklern ohne diese Konzepte schnell unübersichtlich werden, was sich in mangelnder Qualität der ausgelieferten Software widerspiegeln kann. Durch Versionsverwaltung und vordefinierte Tests können Fehler minimiert und lauffähiger Code sichergestellt werden[5]. Ziel soll sein, eine Auswahlmatrix in Form einer Tabelle zu erstellen, um so das passende Workflow-Management-System in Bezug auf die benötigten Software-Engineering-Konzepte für ein anstehendes Projekt auswählen zu können.

1.3 Aufbau

Zu Beginn der Arbeit werden die Grundlagen in den Bereichen Workflow-Management und Software-Engineering geklärt. Anschließend wird der Markt an gängigen Workflow-Management-Systemen analysiert. Im nächsten Kapitel wird ein Kriterienkatalog anhand gängiger Software-Engineering-Konzepte erstellt. Außerdem werden Kriterien zum Vergleich der Systemeigenschaften entworfen. In der Implementierung wird in jedem der anhand der Marktanalyse ausgewählten Workflow-Management-Systeme ein Beispielprozess implementiert. Anschließend werden die Systeme auf Grundlage des erarbeiteten Kriterienkataloges verglichen. Anhand dieser Auswertung wird eine Tabelle zur zukünftigen Entscheidungsfindung entworfen.

Kapitel 2

Grundlagen

2.1 Geschäftsprozessmanagement

In diesem Kapitel werden die Grundlagen des Geschäftsprozessmanagements erläutert. Es wird darauf eingegangen, was Prozesse im Geschäftsleben sind und wie diese mit Hilfe von Workflow-Management-Systemen digitalisiert werden können.

2.1.1 Geschäftsprozess

Nach Arndt in seinem Buch „Grundlagen der Prozessoptimierung“[21] wird ein Geschäftsprozess als „eine von einer Kundenanforderung ausgelöste Folge von Aktivitäten, um das vom Kunden gewünschte Ergebnis zu erstellen“ definiert. Dies bedeutet, dass ein Geschäftsprozess aus einer Vielzahl von Aktivitäten besteht, die klar strukturiert sind[22]. Geschäftsprozesse überschreiten häufig Abteilungsgrenzen, wodurch Schnittstellen entstehen. Außerdem kann ein Geschäftsprozess in kleinere Teilprozesse untergliedert werden.[23][24]

2.1.2 Geschäftsprozessoptimierung

Bei der Geschäftsprozessoptimierung spricht man eigentlich nicht von Optimierung, da selten ein Optimum erreicht wird, sondern eher von einer Verbesserung. In dieser Phase des Geschäftsprozessmanagements werden bestehende Prozesse kritisch hinterfragt, verändert oder neugestaltet. Die Entwicklung von Maßnahmen zur Erreichung von zu Beginn festgelegten Zielen ist von Bedeutung. Während der Umsetzung dieser Maßnahmen muss der Erfolg kontinuierlich geprüft werden.[21] Typisch für die Prozessoptimierung ist nach Gadatsch[23] eine gründliche Ist-Analyse der Situation mit anschließender Optimierung und IT-gestützter Umsetzung.

2.1.3 Abgrenzung Prozess und Workflow

Als Workflow versteht man nach Gadatsch[23] einen formal beschriebenen, ganz oder teilweise automatisierten Geschäftsprozess. Ein Workflow beinhaltet zeitliche, fachliche und ressourcenbezogene Spezifikationen, die für eine automatische Steuerung des Arbeitsablaufes erforderlich sind. Die einzelnen Arbeitsschritte werden von Mitarbeitern oder Anwendungsprogrammen ausgeführt. Es ist wichtig, dass sich der Geschäftsprozess mit dem „Was“ soll getan werden, und der Workflow mit dem „Wie“ etwas umgesetzt wird, befasst.

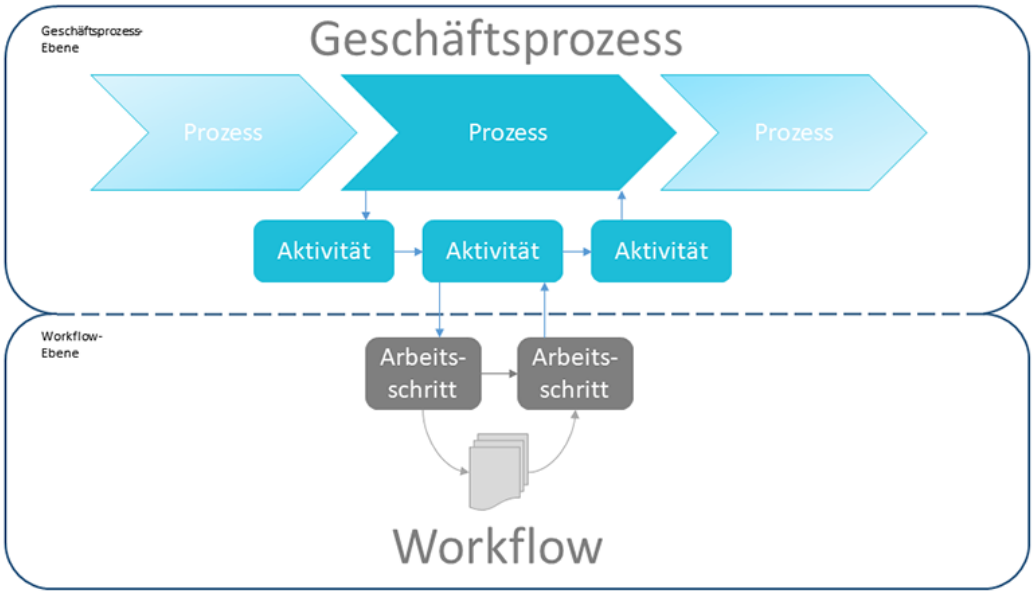

ABBILDUNG 2.1: Zusammenhang zwischen Geschäftsprozess und Workflow[25]

Dies wird durch Abbildung 2.1 verdeutlicht. Die Geschäftsprozess-Ebene definiert den Ablauf von Aktivitäten. Die Workflow-Ebene steuert das automatisierte Abarbeiten dieser Aktivitäten.

2.1.4 Workflow-Management

Das Workflow-Management beschäftigt sich mit der Umsetzung von Geschäftsprozesszielen und der Digitalisierung dieser Geschäftsprozesse[23]. Zu diesem Zweck bietet das Workflow-Management Methoden und Werkzeuge zur digitalen Überwachung und Analyse der Arbeitsabläufe. Einige Ziele des Workflow-Managements sind beispielsweise eine bessere Kundenzufriedenheit, die Verkürzung der Durchlaufzeit von Prozessen, die Verbesserung der Prozessqualität und eine permanente Qualitätssicherung.[26]

2.1.5 Workflow-Management-Systeme

Ein Workflow-Management-System ist eine Software zur Unterstützung des Workflow-Managements. Diese Software stellt dem Nutzer eine Applikation zur Verfügung, in der die einzelnen Arbeitsschritte und Aufgaben in einen Ablauf integriert sind.[27][26]
Wichtig ist, dass der abgebildete Geschäftsprozess sich vorab modellieren lässt. Vor der Modellierung eines Geschäftsprozesses sollte eine Prozessanalyse durchgeführt werden, um Aufgaben und beteiligte Personen zu kennen und so den Ablauf präzise beschreiben zu können.[28]
Wird der Geschäftsprozess in einem Workflow-Management-System abgebildet, spricht man von einem Workflow. (siehe Unterabschnitt 2.1.3)

Nutzen durch den Einsatz von Workflow-Management-Systemen

Für das Unternehmen entstehen einige Vorteile durch den Einsatz von Workflow-Management-Systemen. So werden die Mitarbeiter von Routineaufgaben entlastet,

die nun von dem digitalisierten Geschäftsprozess automatisch übernommen werden. Außerdem kann sich auch die Durchlaufzeit der Abläufe verringern und Mehrfacheingaben vermieden werden. Zusätzlich bietet ein Workflow-Management-System dem Controlling die Möglichkeit, detaillierte Analysewerte des Ablaufes, wie beispielsweise die durchschnittliche Bearbeitungsdauer eines Antrages, einzusehen.[29] Dies führt laut zur Mühlen und Hansmann[26] dazu, dass der Geschäftsprozess transparent wird, was bedeutet, es ist jederzeit möglich, Auskunft über den Zustand einer laufenden Instanz zu erhalten. So wird ebenfalls eine höhere Transparenz der Arbeitsbelastung erreicht. Über das Rollenkonzept kann jeder mit entsprechender Berechtigung die Aufgaben der zu vertretenden Person einsehen und bearbeiten. Auch die Liege- und Transportzeit wird durch die Digitalisierung minimiert. So können Dokumente nun elektronisch übergeben werden und sind nicht auf den Postweg oder das menschliche Weitergeben angewiesen. Diese Verkürzung der Durchlaufzeit ist direkt messbar.[26]

Standards bei der Modellierung von Workflows

Im Bereich Prozessdigitalisierung ist es wichtig, dass alle am Projekt beteiligten verstehen, wie der Geschäftsprozess aufgebaut und strukturiert ist. Um dieses umfassende Verständnis zu erreichen, wurden Standards geschaffen. Diese Standards sollen für alle Beteiligten klar verständlich sein, vom Business Analyst, der das initiale Prozessmodell entwirft, bis zum Entwickler, der die Funktionalität des Workflows implementiert.[30]
Chinosi und Trombetta geben in „BPMN: An introduction to the standard“[30] einen Überblick über Standards zur Modellierung von Workflows. Einer dieser Standards ist nach Chinosi und Trombetta der seit 2004 für die Modellierung von Workflows verwendete BPMN-Standard. 2005 wurde dieser als offizieller Standard der Object Management Group (OMG) anerkannt. Dabei handelt es sich um eine graphische Notation. 2011 wurde dieser Standard durch die neue Version BPMN 2.0 revolutioniert, welche auf XML basiert. Durch diese Unterstützung von XML wird aus der graphischen Notation eine für Maschinen verständliche, ausführbare Beschreibungssprache.[30]

2.2 Software-Engineering-Konzepte

2.2.1 Software-Engineering

Um zu verstehen, wieso es sinnvoll ist, in der Softwareentwicklung nach definierten Konzepten vorzugehen, muss klargestellt werden, wo die Herausforderungen bei der Entwicklung von Software liegen.

Balzert beschreibt in seinem Werk *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*[31] Charakteristika von Software, die zu beachten sind. So ist diese ein immaterielles Produkt, was zu Inkonsistenzen zwischen dem laufenden Code und der zugehörigen Dokumentation führen kann. Außerdem muss bei der Wartung von Software ein Umdenken stattfinden, denn es können nicht einfach Verschleißteile ausgetauscht werden. So fällt die Wartung wesentlich komplexer aus. Auch muss man sich über den Alterungsprozess von Software bewusst sein. Die Technik befindet sich im stetigen Wandel, was dazu führen könnte, dass Software auf einmal nicht mehr lauffähig ist.[31]

Die Softwareentwicklung hat in den letzten Jahren an zunehmender Bedeutung gewonnen. Die Anforderungen sind höher und auch die Nachfrage an intelligenten Softwarelösungen steigt stetig.[32]
Software-Engineering, oft auch als Softwaretechnik bezeichnet, wird von Balzert [31, p. 17] wie folgt definiert: „Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen. Zielorientiert bedeutet die Berücksichtigung z.B. von Kosten, Zeit, Qualität."
Diese Definition beschreibt die Charakteristika des Software-Engineerings. Mit dieser Definition ist gemeint, dass es bei der Softwareentwicklung im Sinne des Software-Engineerings immer klar definierte Vorgehensweisen und Ziele, die es zu erreichen gilt, gibt.[31]

2.2.2 Software-Engineering-Konzepte

Im modernen Software-Engineering wird nach Perez, Wang und Casale[33] immer öfter das sogenannten „DevOps-Konzept", was kurz für Development Operations Konzept steht, eingesetzt. Dieses Konzept versucht eine Brücke zwischen Entwicklungsebene und der operativen Ebene zu schaffen.
Auftraggeber für Softwareprojekte erwarten immer häufiger eine schnelle Auslieferung der Software sowie ein unkompliziertes Einspielen von Weiterentwicklungen und Änderungen. Deshalb beschäftigt sich das DevOps-Paradigma vorrangig mit der Automatisierung von Entwicklungsprozessen. Wichtige Aspekte sind dabei das Abrufen des in der Versionsverwaltung eingecheckten Codes, das automatisierte Bauen von Anwendungen und das Ausführen von Testfällen.[34] Beispiele für Mechanismen, um diese Aspekte in der Softwareentwicklung umzusetzen, sind Continuous Integration, Continuous Delivery oder das Nutzen einer Versionsverwaltung. Diese dienen der Qualitätssicherung entwickelter Software und der kontinuierlichen Testung und Auslieferung von Individualsoftware.[5]

Kapitel 3

Marktanalyse etablierter Workflow-Management-Systeme

Workflow-Management-Systeme existieren als kostenlose Open Source Software oder als bezahlte Enterprise Anwendungen. In diesem Kapitel werden Open Source Workflow-Management-Systeme, die sich am Markt etabliert haben, untersucht.

3.1 Auswahl der zu untersuchenden Systeme

Für die Auswahl an relevanten Open Source Systemen wurde eine Liste bekannter Workflow-Management-Systeme auf GitHub[35] herangezogen. Hier sind die BPM Suites und die vollwertigen Produkte (full fledged Products) von Interesse, da diese einen großen Funktionsumfang aufweisen. Die Auswahl fand Anhang dieser Liste von Wahnon[35], den Interessen des Unternehmens (Abschnitt 1.1) und anhand folgender Kriterien statt:

- Mit dem System kann individuelle Softwareentwicklung, im Bezug auf die Digitalisierung von Geschäftsprozessen, betrieben werden (Kein No-Code-Ansatz) (K1)
- Die Programmierung der Workflow-Logik erfolgt mittels Java (K2)
- Das System verwendet keinen Cloud-basierten Ansatz (K3)

3.1.1 BPM Suites

Kriterium / **System**	K1	K2	K3
Activiti	✓	✓	✓
Activiti Cloud	✓	✓	✗
Bonita	✓	✓	✓
Flowable	✓	✓	✓
jBPM	✓	✓	✓

TABELLE 3.1: Tabellarische Auswertung der Kriterien für die BPM Suites aus der Liste von Wahnon[35]

Activiti Cloud basiert auf der Verwendung in der Cloud und erfüllt daher Kriterium K3 nicht[35].

3.1.2 Vollwertige Systeme

Kriterium / **System**	K1	K2	K3
Airflow	✓	✗	✓
N8n-io	✗	✗	✓
Argo	✗	✗	✗
Prefect	✓	✗	✓
Cadence	✓	✓	✓
StackStorm	✓	✗	✓
RunDeck	✓	✗	✓
Azkaban	✗	✗	✓
MassTransit	✓	✗	✓
Conductor	✓	✓	✗
easy-rules	✗	✓	✓
Brigade	✓	✗	✗
Camunda	✓	✓	✓
Zeebe	✓	✓	✓
elsa-workflows	✓	✗	✓

TABELLE 3.2: Tabellarische Auswertung der Kriterien für die vollwertigen Produkte aus der Liste von Wahnon[35]

Wie in Tabelle 3.2 dargestellt, erfüllen nicht alle Systeme die aufgestellten Kriterien. Apache Airflow ist eine auf der Programmierung mit Python basierende Plattform, um Workflows programmatisch zu Steuern[36]. Somit ist die Programmierung mittels Java nicht möglich. N8n-io verfolgt einen No-Code-Ansatz und bietet so keine Grundlage für die individuelle Softwareentwicklung sowie die Programmierung mit Java[37]. Argo wurde für den Einsatz in Kubernetes konzipiert und definiert die Workflows in YAML-Dateien, welche für Docker-Container verwendet werden[38]. Somit ist eine individuelle Entwicklung von Software sowie die Programmierung mittels Java nicht möglich. Prefect nutzt die Programmierung der Workflows durch Python[39]. StackStorm dient zur Verbindung verschiedener Anwendungen und bietet keinen Client in Java an[40]. RunDeck definiert Workflows in XML oder YAML, bietet aber auch die Möglichkeit, Workflows über ein CLI-Tool oder eine API zu definieren[41]. Die Workflow-Logik kann dabei nicht in Java geschrieben werden. Azkaban ist ein auf Batch basierter Job-Scheduler für Hadoop-Jobs und somit nicht für die individuelle Softwareentwicklung und Programmierung mittels Java geeignet[35]. Im Workflow-Management-System MassTransit wird .Net für die Programmierung verwendet[35]. Conductor ist eine Cloud-basierte Anwendung[35]. Die Java-Rules-Engine easy-rules definiert Regeln und Aktionen. Diese Aktionen werden abhängig von den Regeln ausgeführt[42]. Somit bildet dieses System keine gezielten Abläufe ab, sondern lediglich Handlungsanleitungen auf bestimmte Ereignisse. Brigade

steuert automatisierte, mit JavaScript beschriebene Aufgaben in der Cloud[43]. Somit wird der Workflow nicht über Java mit Logik versehen. Außerdem verfolgt dieses System einen Cloud-basierten Ansatz. Elsa-Workflows verwendet für die Programmierung der Funktionalität von Workflows C#, JSON, YAML oder XML, bietet jedoch keine Möglichkeit für die Programmierung mit Java[44].

Die Workflow-Management-Systeme jBPM, Activiti, Flowable, Camunda, Zeebe, Bonita und Cadence erfüllen alle Kriterien und werden deshalb für die Marktanalyse ausgewählt.

3.2 jBPM

jBPM ist ein von Red Hat JBoss entwickeltes Open Source Projekt[45]. Das auf Java basierende Workflow-Management-System zeichnet sich durch seine Flexibilität und Leichtgewichtigkeit aus[46].

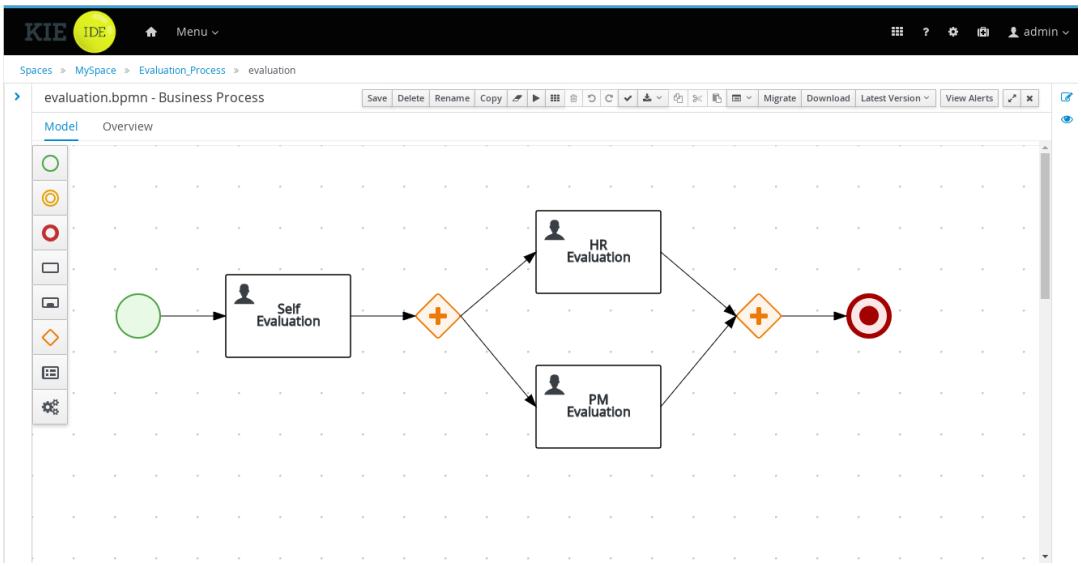

ABBILDUNG 3.1: Modellierung von Workflows in jBPM[47]

Der Funktionsumfang von jBPM wird in dem Artikel „A context-aware and workflow-based framework for pervasive environments“ von Avenoğlu und Eren[48] beschrieben. Ein webbasierter Process-Designer(Abbildung 3.1) kann zum Modellieren von Workflows verwendet werden. Die Core-Engine bildet das Herzstück des Projektes und dient dem Ausführen von Workflows. Sie ist flexibel einsetzbar und kann in das Projekt integriert oder als Service bereitgestellt werden. Der Human Task Service ist optional und kümmert sich um manuelle Aufgaben, sollten Endnutzer in den Workflow involviert sein.[48]

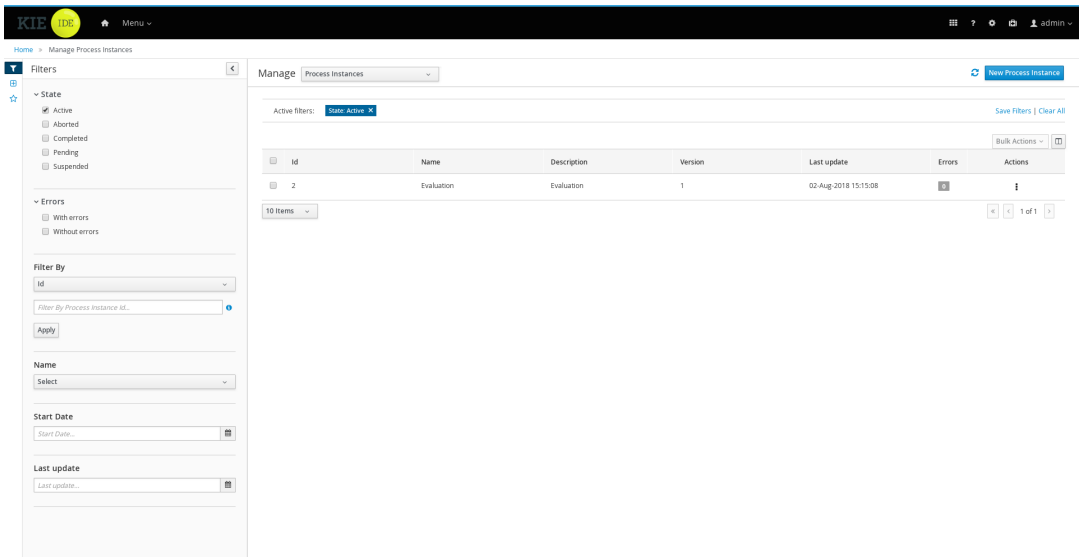

ABBILDUNG 3.2: Verwaltung der Workflow-Instanzen in jBPM[49]

Für die Überwachung und das Management des Workflows und der Workflow-Instanzen kann die Management-Konsole(Abbildung 3.2) verwendet werden.[48]

3.3 Activiti

Das Workflow-Management-System Activiti wurde 2010 von der 2005 gegründeten Alfresco Software Inc. ins Leben gerufen[45]. Die Entwicklung von Activiti wurde durch ehemalige jBPM Entwickler geleitet[50].

Die technischen Komponenten von Activiti beinhalten einen webbasierten Modeler zur graphischen Modellierung von Workflows[51].

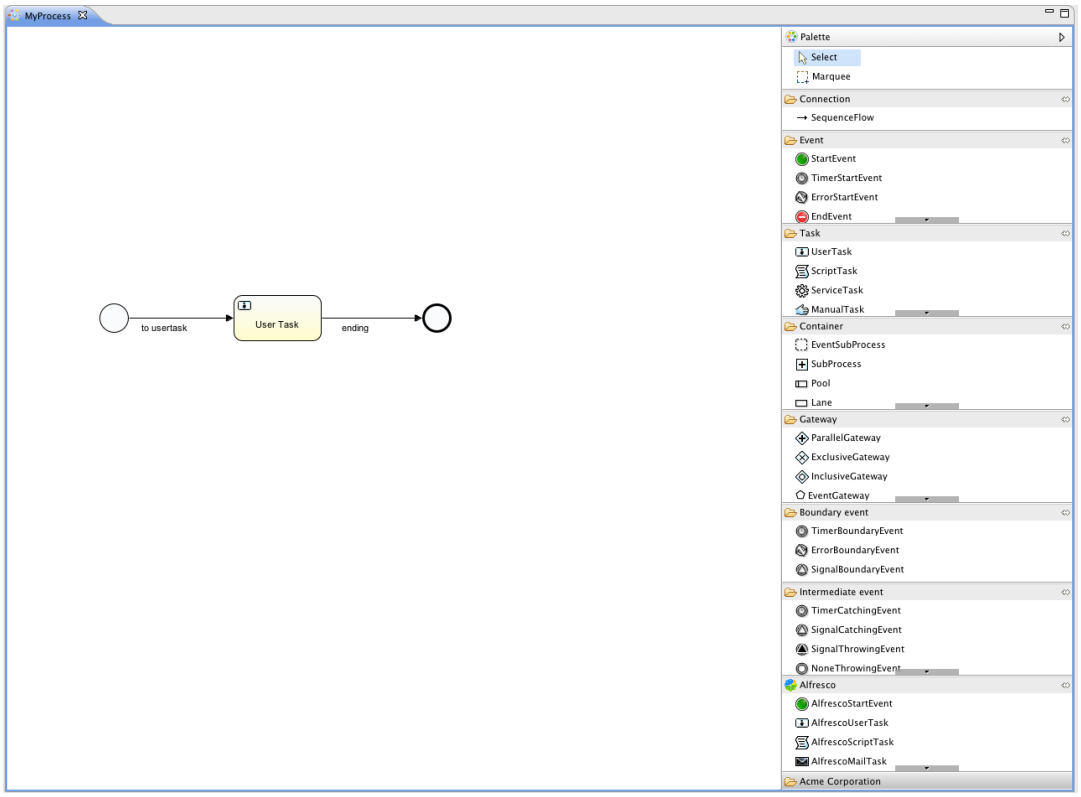

ABBILDUNG 3.3: Modellierung von Workflows in Activiti[52]

Ebenfalls enthalten ist der Activiti Designer, welcher als Eclipse-Plugin verfügbar ist (Abbildung 3.3). Er dient dazu, den modellierten Workflow mit Logik zu versehen. Auch bietet der Designer die Möglichkeit, Workflows graphisch zu modellieren.[51]

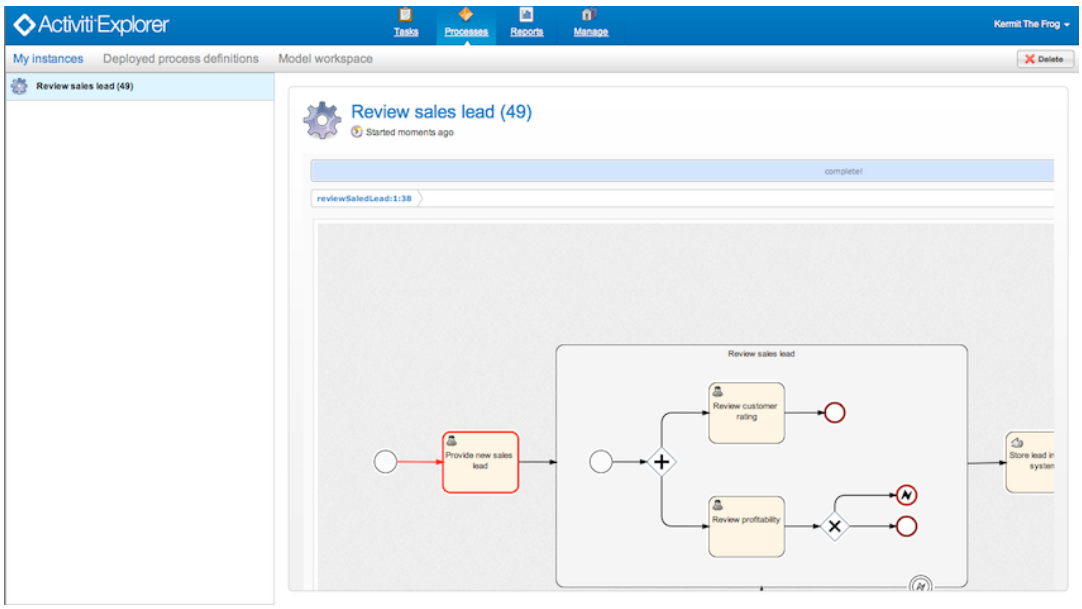

ABBILDUNG 3.4: Verwaltung der Workflow-Instanzen in Activiti[53]

Der Activiti Explorer ist webbasiert und enthält Management- und Monitoring

Funktionen. Hier können Tasks, User und Workflows verwaltet werden (Abbildung 3.4). Zusätzlich sind im Explorer auch Analysetools enthalten. Die Process-Engine bildet die Logik von Activiti. Sie ist für das Starten von Workflows, aber auch für das Ausführen der Tasks zuständig. Activiti bietet REST-APIs, um auf die Engine zuzugreifen.[51]

3.4 Flowable

Flowable wurde durch das gleichnamige Unternehmen „Flowable“ von Activiti geforked[54]. Dieses System besteht aus einer Reihe von Webanwendungen, welche für verschiedene Bereiche zuständig sind[55].

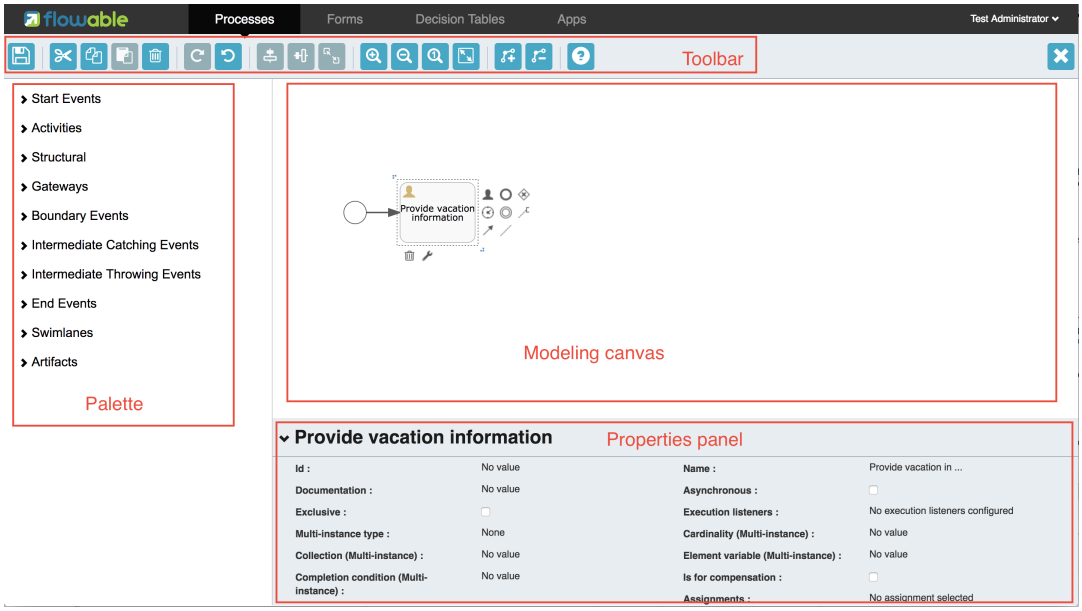

ABBILDUNG 3.5: Modellierung von Workflows in Flowable[56]

Die technische Seite von Flowable bildet die Process-Engine API ab. Mithilfe dieser lassen sich alle Funktionen des Systems verwenden. Für Flowable gibt es eine Webanwendung. Diese Webanwendung enthält den Modeler (Abbildung 3.5), welcher die graphische Modellierung von Workflows ermöglicht.[57]

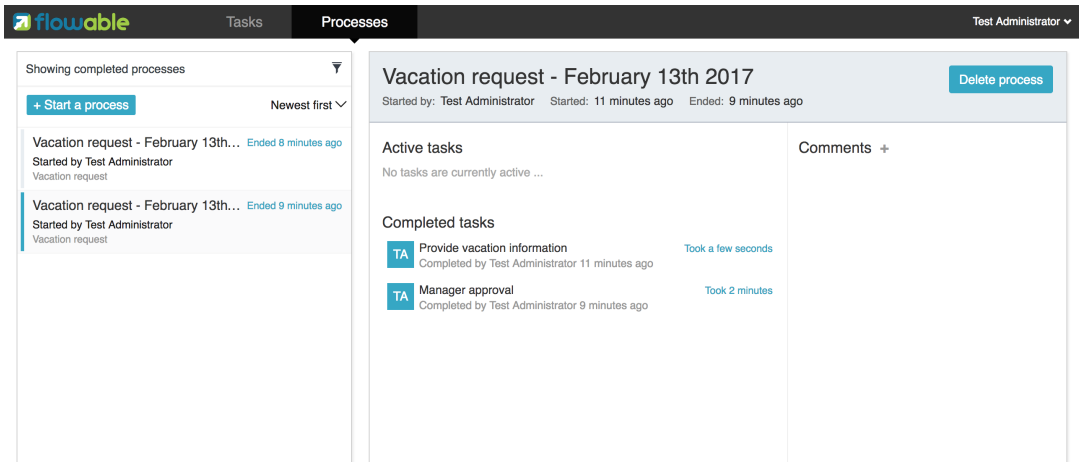

ABBILDUNG 3.6: Verwaltung der Workflow-Instanzen in Flowable[58]

Ebenfalls in der Webanwendung enthalten ist Flowable Task. Dieser Bereich bietet Funktionen zum Steuern und Anzeigen von Workflow-Instanzen und Aufgaben (Abbildung 3.6). Über eine Admin-Oberfläche können Workflow-Instanzen, Aufgaben und Workflow-spezifische Daten geändert werden.[57]

3.5 Camunda BPM

Das auf Java basierende Framework zur Prozessautomatisierung Camunda BPM wurde von der Camunda Services GmbH im Jahr 2013 von Activiti geforked[59]. Das in Berlin ansässige Unternehmen wurde 2008 gegründet[45]. Durch eine serviceorientierte API wird Java-Anwendungen in Camunda die direkte Möglichkeit zur Interaktion mit der Prozess-Engine geboten[46].

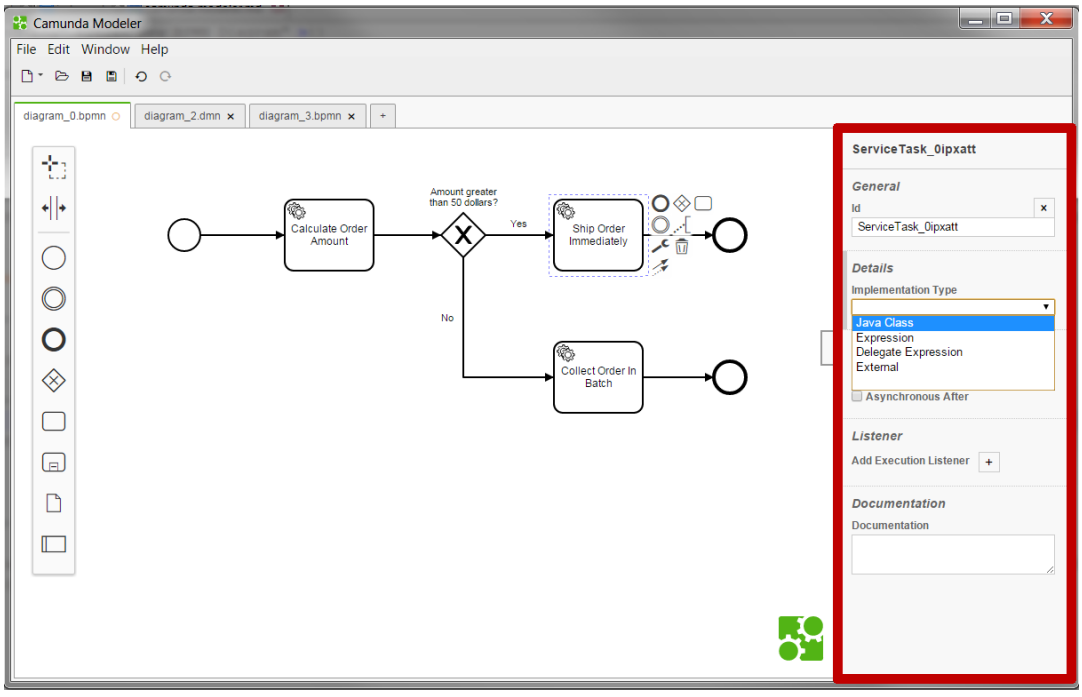

ABBILDUNG 3.7: Modellierung von Workflows in Camunda[60]

Mit dem Camunda Modeler (Abbildung 3.7) können Workflows und DMN-Entscheidungstabellen modelliert werden. Es existieren zwei verschiedene Engines. Die BPMN-Workflow-Engine dient dem Workflow-Management und bietet die Möglichkeit zur Nutzung als REST-Service oder zur Einbettung in Java-Anwendungen.[61]
Die DMN-Engine wird genutzt, um fachlich definierte Entscheidungstabellen auszuführen. Sie ist in die Workflow-Engine integriert, kann jedoch auch als Standalone mittels REST-Aufrufen oder durch die Einbettung in eine Java-Anwendung verwendet werden. Die Tasklist ist eine Webanwendung mit der Möglichkeit, durch Workflows zugewiesene Aufgaben als Endanwender zu erledigen.[45]

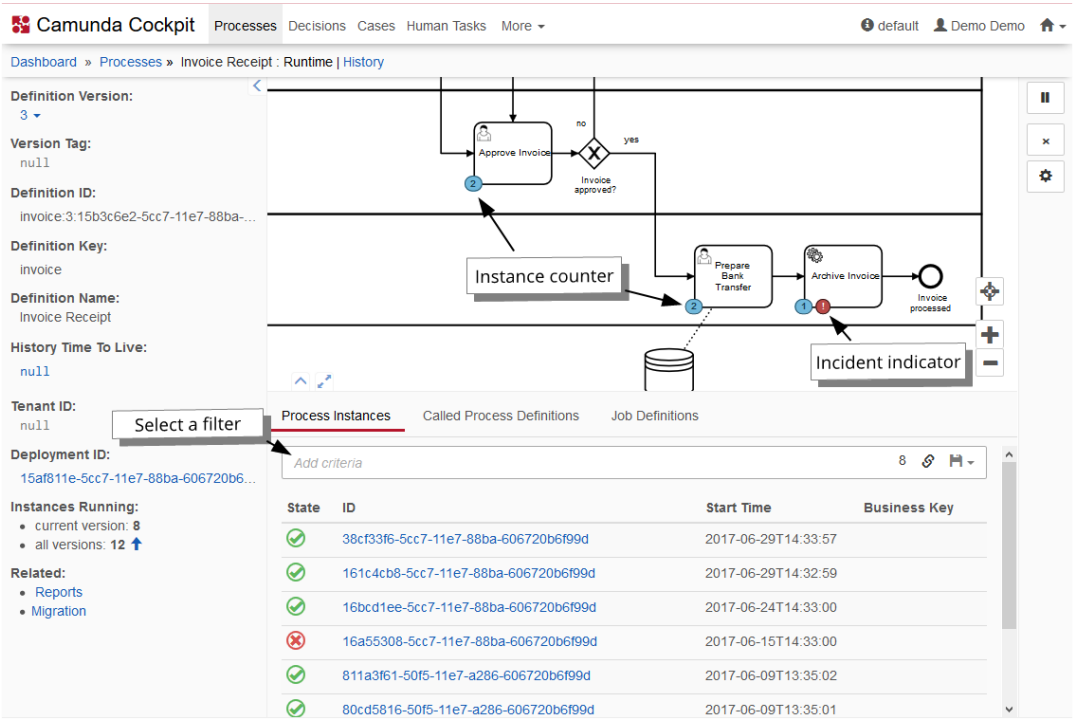

ABBILDUNG 3.8: Verwaltung der Workflow-Instanzen in Camunda[62]

Das Cockpit bietet die Möglichkeit, die Workflow-Instanzen einzusehen und zu verwalten (Abbildung 3.8). Zudem können Reports entwickelt werden, aber auch Performance-Ziele überwacht und Engstellen im Workflow erkannt werden.[45]

3.6 Zeebe

Zeebe ist eine von der Camunda Services GmbH entwickelte ereignisgesteuerte Workflow-Engine zur Orchestrierung von Microservices[63].
Zeebe wird nicht direkt als Open Source deklariert, da es über eine eigene Community-Lizenz verfügt. Das System wird in *Zeebe License Overview and FAQ*[64] als „*Source available*“ bezeichnet. Die Community-Lizenz wird laut diesem Artikel verwendet, um die Nutzung von Zeebe als Workflow-Service im Cloud-Bereich einzuschränken.

In dem Artikel „A Survey on Service Composition Languages“ von Nikoo, Babur und Brand[61] wird der Funktionsumfang wie folgt beschrieben:

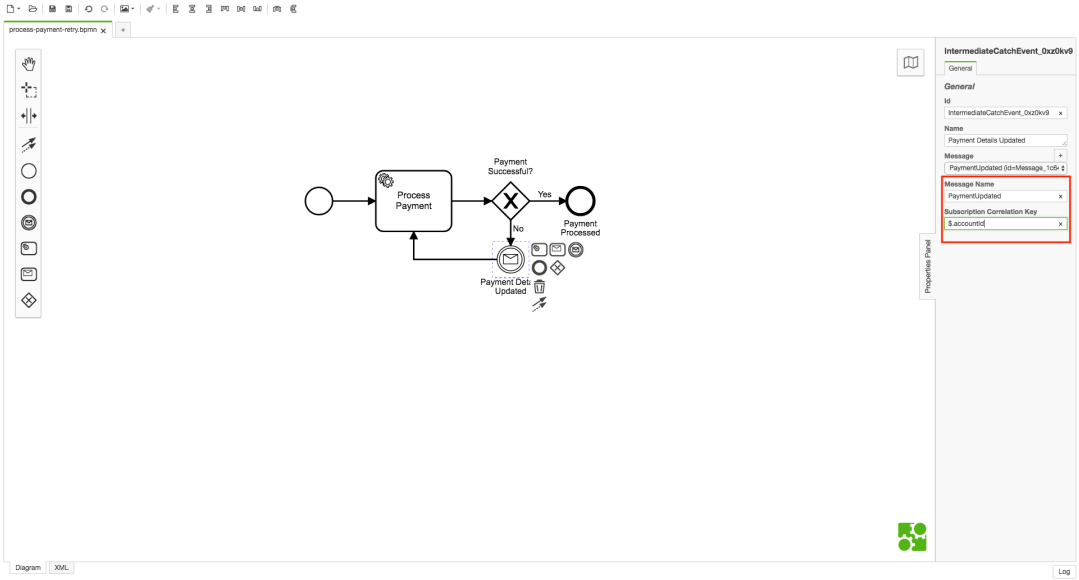

ABBILDUNG 3.9: Modellierung von Workflows in Zeebe[65]

Für die Modellierung von Workflows steht eine graphische Modellierungssoftware zur Verfügung (Abbildung 3.9).

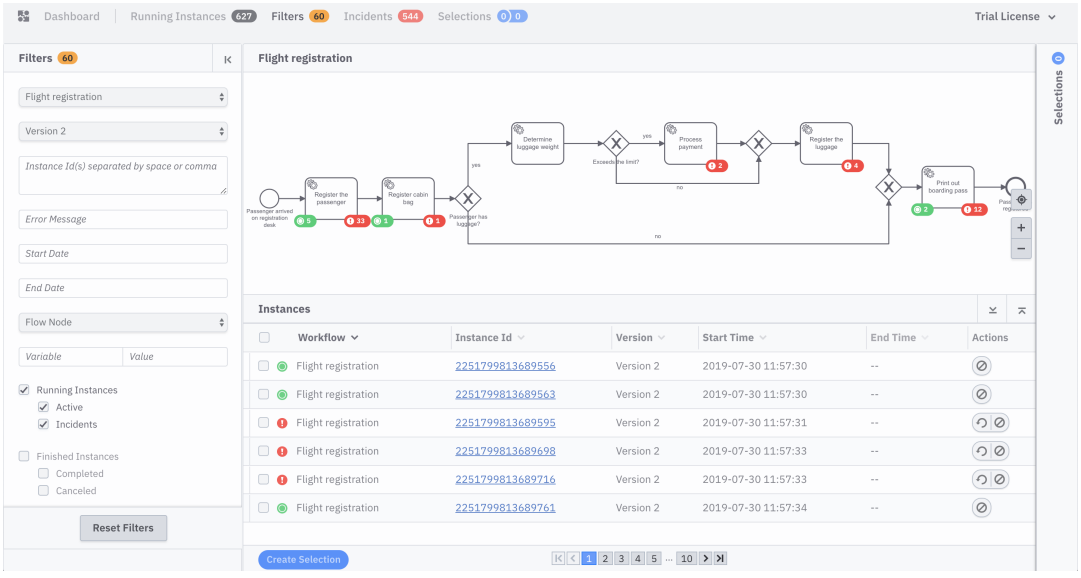

ABBILDUNG 3.10: Verwaltung der Workflow-Instanzen in Zeebe[66]

Außerdem wird eine Weboberfläche mit Funktionen des Monitorings und der Datenanalyse bereitgestellt (Abbildung 3.10). Zudem bietet Zeebe auch eine eigene Engine für die Ausführung von Workflows an.[61]

3.7 Bonita

Bonita wurde erstmals 2001 als Open Source Projekt veröffentlicht. 2009 gründete sich dann Bonitasoft um die Entwicklung weiterzuführen.[67] Diese Lösung für

das Digitalisieren von Geschäftsprozessen zeichnet sich durch die Möglichkeit aus, schnell und unkompliziert auf Prozessen basierende Anwendungen zu erstellen. Bonita richtet sich an Geschäftsleute, welche ohne Expertenunterstützung Geschäftsprozesse digitalisieren möchten.[46]

„Bonita BPM: an open–source BPM–based application development platform to build adaptable business applications“[67] beschreibt die drei wesentlichen Funktionen, das BPM Studio, die BPM-Engine und das BPM Portal.

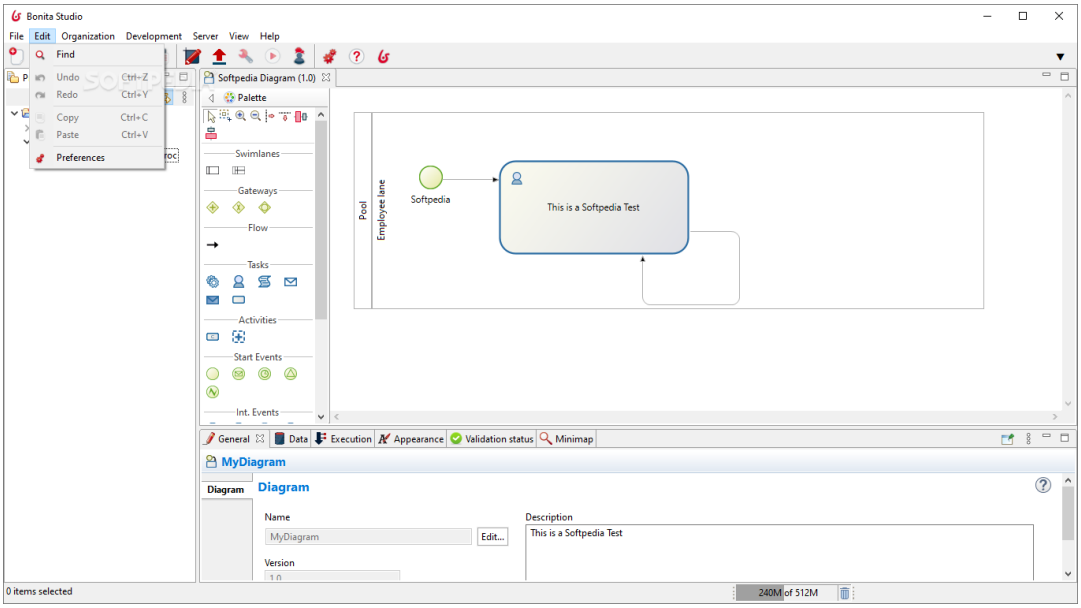

ABBILDUNG 3.11: Modellierung von Workflows in Bonita Studio[68]

Im BPM Studio können Workflows graphisch modelliert und mit Funktion versehen werden (Abbildung 3.11). Die Engine ist für das Ausführen großer Workflows mit hoher Nachfrage und großem Transaktionsvolumen konzipiert.[67]

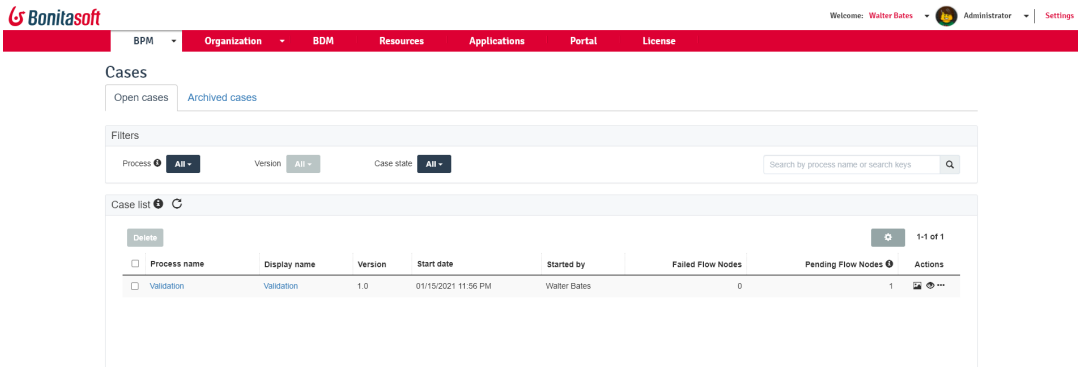

ABBILDUNG 3.12: Verwaltung der Workflow-Instanzen in Bonita[69]

Ebenfalls enthalten ist eine anpassbare Weboberfläche, das BPM Portal, für die Endnutzer. In dieser Oberfläche finden sich auch das Fehlermanagement und Analysefunktionen. Hier können Workflows und deren Instanzen eingesehen werden (Abbildung 3.12).[45]

3.8 Cadence

Cadence ist eine von Uber Technologies Inc. entwickelte Orchestrierungs-Engine. Dieses System trennt Orchestrierungs- und Implementierungslogik durch Aktivitäten und Workflows.[63]

Der Artikel „Cadence — The only workflow orchestrator you will ever need“[70] bietet eine Übersicht über die wesentlichen Bestandteile der Cadence-Engine. Diese sind der Cadence-Service, der Workflow-Starter und die Workflow- und Activity-Worker.
Den Hauptteil der Engine bildet der Cadence-Service, dieser koordiniert alle Abläufe. Dieser Service steuert die sogenannte Worker und weist ihnen Aufgaben zu. Die Funktionalität der Worker wird per Code geschrieben und ausgeführt, sobald dem Worker eine Aufgabe zugewiesen wird. Zuerst muss ein Workflow von einem Workflow-Starter gestartet werden. Anschließend gibt es zwei Worker, den Workflow-Worker und den Activity-Worker, die auf Aufgaben reagieren. Die Hauptaufgabe des Workflow-Workers ist es, Aktivitäten zu koordinieren. Der Workflow-Worker hat dabei volle Kontrolle darüber, in welcher Reihenfolge die Aufgaben ausgeführt werden. Der Activity-Worker implementiert eine einzelne genau definierte Aktion und dient zur Anbindung von Drittsystemen. Cadence bietet keine Oberfläche zur graphischen Modellierung von Workflows. Die Workflows werden per Code definiert.[70]

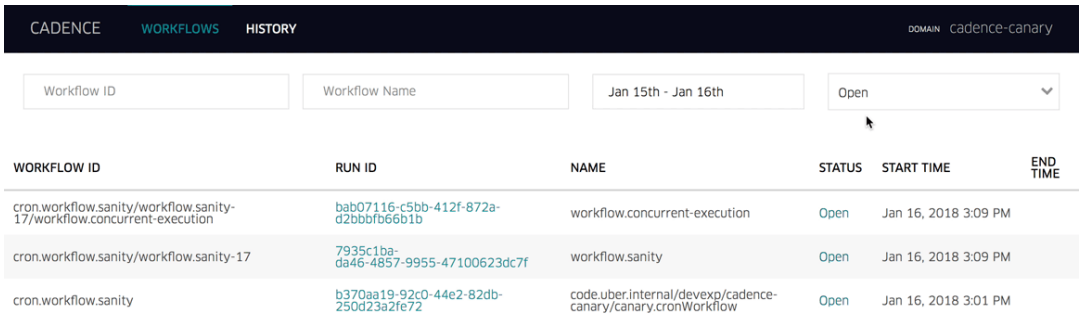

ABBILDUNG 3.13: Verwaltung der Workflow-Instanzen in Cadence[71]

Zur Verwaltung der Workflow-Instanzen wird eine Benutzeroberfläche (Abbildung 3.13) angeboten[72].

Kapitel 4

Kriterienkatalog zur Überprüfung der Anwendbarkeit von Software-Engineering-Konzepten auf die Entwicklung mit Workflow-Management-Systemen

In diesem Kapitel werden die zu untersuchenden Kriterien und deren Nutzen in der Softwareentwicklung erläutert.

4.1 Entwicklungswerkzeuge

In diesem Absatz werden alle Kriterien zusammengefasst, die sich mit der Entwicklung von Software beschäftigen.

4.1.1 Versionsverwaltung

Die Versionsverwaltung bietet einem Team die Möglichkeit, gemeinsam und gleichzeitig an einem Projekt zu arbeiten, ohne dabei die Arbeit des anderen zu stören. Außerdem werden in einer Versionsverwaltung frühere Versionen der Dateien hinterlegt, sodass jederzeit zu einer Vorgängerversion zurückgekehrt werden kann.[18] Mit diesem Vergleichskriterium wird die Möglichkeit untersucht, für den Quellcode eines Workflows auf Basis des zu untersuchenden Workflow-Management-Systems eine Versionsverwaltung zu nutzen. Gibt es Systeme, die proprietäre Formate verwenden, könnte es zu Problemen beim Zusammenführen verschiedener Versionen kommen. Es können nicht oder nur schwer lösbare Konflikte bei dieser Zusammenführung auftreten.

4.1.2 Build-Management-Tools

Der Unterschied zum nativen Kompilieren besteht darin, dass Build-Management-Tools das Bauen, Testen und Packen von Anwendungen vereinfachen[4]. Ein Mittel dazu sind inkrementelle Builds. Dabei wird der aktuelle Status des Quellcodes mit den vorherigen Builds verglichen, um Quellcode-Änderungen nur mit den nötigsten Befehlen anzuwenden. Außerdem können durch Build-Management-Tools, Bibliotheken und Abhängigkeiten im Projekt verwaltet werden. So müssen die einzelnen

Abhängigkeiten nicht manuell heruntergeladen und in das Projekt eingepflegt werden. Zudem wird die Möglichkeit geboten, die Build-Tools durch Plugins benutzerdefiniert anzupassen und zu konfigurieren.[4]
Mit diesem Kriterium wird geprüft, ob sich Build-Management-Tools bei der Softwareentwicklung auf Basis des zu untersuchenden Workflow-Management-Systems anwenden lassen.

4.1.3 Testabdeckung

Tests prüfen die Funktion einer Anwendung und dienen der Qualitätssicherung von Software. Es werden Testfälle mit vordefinierten Eingabe-Variablen erstellt und die Anwendung damit durchlaufen. Kann eine möglichst hohe Testabdeckung des Codes erreicht werden, so wird die Wahrscheinlichkeit für Fehlfunktionen minimiert.[73] In den einzelnen Systemen muss geprüft werden, wie gut sich solche Tests integrieren lassen. Auch spielt die Testabdeckung eine Rolle. So wird ebenfalls betrachtet, ob es Code gibt, welcher nicht mit Testfällen abgedeckt werden kann.

4.1.4 Continuous Integration/Continuous Delivery

Unter Continuous Integration, oft auch mit CI abgekürzt, versteht man nach Dalton[19] das automatisierte Bauen und Testen von in der Versionsverwaltung eingecheckter Software. Im ersten Schritt wird überprüft, ob der eingecheckte Code erfolgreich kompiliert werden kann. Besonders bei Projekten, in denen mehrere Softwarekomponenten zu einer Applikation zusammengefasst werden müssen, kann CI durch automatisiertes Kompilieren Fehler feststellen. Außerdem kann die Ausführung von in Unterabschnitt 4.1.3 beschriebenen Tests definiert werden, welche beim Einchecken von Code-Änderungen die neue Softwareversion inhaltlich auf Fehler überprüfen. Der Erfolg oder Misserfolg dieser Tests ist jederzeit einsehbar.[19]
Continuous Delivery (kurz CD) knüpft an das automatisierte Bauen der Applikation an. Die Änderungen werden nach erfolgreicher Testung automatisiert in eine definierte Umgebung eingepflegt und können so dem Kunden zur Verfügung gestellt werden.[20]
Mit diesem Kriterium wird geprüft, ob sich ein Workflow-Management-System zusammen mit CI/CD nutzen lässt und so die Entwicklung von qualitativ hochwertiger Individualsoftware erleichtert und verbessert wird.

4.2 Systemeigenschaften

Dieser Unterpunkt befasst sich mit den systemspezifischen Aspekten von Workflow-Management-Systemen.

4.2.1 Kompatible Standards

Mit diesem Kriterium wird überprüft, ob die Workflow-Management-Systeme den in Abschnitt 2.1.5 erläuterten BPMN-Standard zur Modellierung von Workflows unterstützen oder ob die Systeme proprietäre Formate einsetzen. Außerdem soll geprüft werden, ob die Programmierung über klassische Programmiersprachen wie beispielsweise Java möglich ist oder ob das System eigene Sprachen verwendet. Wird kein Standard angewandt, müsste beim Wechsel des Workflow-Management-Systems eine neue Art der Modellierung oder eine neue Programmiersprache erlernt

werden. Außerdem ist es möglich, unter Einsatz eines Standards für die Modellierung das Workflow-Management-System für die Implementierung zu wechseln, da das Model denselben Standard unterstützt.

4.2.2 Entwicklungsfreiheit

Mit diesem Kriterium soll untersucht werden, ob festgelegte Tools wie beispielsweise bestimmte Entwicklungsumgebungen für die Modellierung und Programmierung der Workflow-Management-Systeme verwendet werden müssen. Benötigen die Systeme nicht bekannte Tools, müssten sich Entwickler erst in diese einarbeiten.

4.2.3 Anbindung an Webapplikation

Für die Entwicklung von Individualsoftware ist es oft notwendig, dem Kunden eine auf seine Bedürfnisse zugeschnittene Webapplikation zur Verfügung stellen zu können.
Mit diesem Kriterium soll untersucht werden, wie gut sich die Workflow-Management-Systeme in eine Webapplikation integrieren lassen. Es wird geprüft, inwieweit es möglich ist, auf Workflow-Instanzen aus einer Webapplikation zuzugreifen und diese zu steuern. Hierunter fällt unter anderem das Starten von Workflow-Instanzen und das Abschließen von Aufgaben.

4.2.4 Typsicherheit der Workflow-Instanzvariablen

Wird ein Workflow-Management-System eingesetzt, beinhaltet der Workflow häufig interne Variablen. Diese sind in der Workflow-Instanz enthalten und können gesetzt oder ausgelesen werden. Bietet ein System keine Typsicherheit, besitzen die Workflow-Instanzvariablen zur Laufzeit keine feste Typzuweisung. Dies kann zu Problemen im Programmablauf führen, da keine Sicherheit besteht, welchen Typ die aktuell verwendete Variable besitzt.[74] Es soll überprüft werden, inwieweit die Typsicherheit der Workflow-Instanzvariablen in den einzelnen Systemen gewährleistet ist.

Kapitel 5

Anwendung der Software-Engineering-Konzepte auf die Entwicklung mit Workflow-Management-Systemen

In diesem Kapitel wird ein Beispielworkflow in ausgewählten Workflow-Management-Systemen implementiert. Diese Implementierung bildet die Grundlage für das Überprüfen der Vergleichskriterien.
In dieser Arbeit werden verschiedene Ansätze bezüglich des Aufbaus der Workflow-Management-Systeme untersucht. Daher wurden die Workflow-Management-Systeme Zeebe, Bonita und Cadence aufgrund ihrer unterschiedlichen Ansätze für die Untersuchung der Kriterien ausgewählt. Da Flowable und Camunda von Activiti geforked wurden[54][59] und die Entwicklung von Activiti durch ehemalige jBPM-Entwickler geleitet wurde[50] verfolgen diese Systeme ähnliche Ansätze. Camunda hat sich seit dem Fork von Activiti schnell weiterentwickelt[59] und wird daher aus dieser Gruppe von ähnlichen Systemen neben Zeebe, Bonita und Cadence für die Implementierung ausgewählt.

5.1 Beispielapplikation

Zur Überprüfung der Kriterien wurde eine in diesem Kapitel beschriebene Beispielapplikation in den ausgewählten Workflow-Management-Systemen implementiert.

5.1.1 Workflow

Für die Implementierung eines Beispielworkflows wurde der Geschäftsprozess einer Projektvermittlung verwendet.

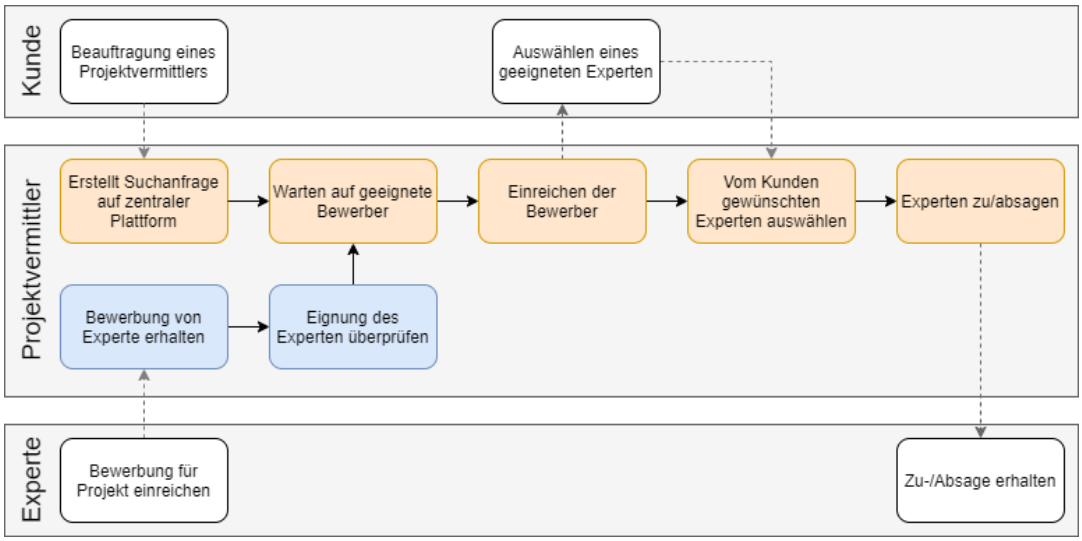

ABBILDUNG 5.1: Grundlegender Geschäftsprozess für die Implementierung

Bei dem in Abbildung 5.1 dargestellten Geschäftsprozess gibt es drei Beteiligte: Einen Kunden, welcher sich eine Softwarelösung wünscht, einen Projektvermittler und Experten.

Der Kunde kann einen Projektvermittler beauftragen, sein gewünschtes Projekt auszuschreiben. Diese Ausschreibung ist öffentlich sichtbar und Selbstständige oder Unternehmen, welche gerade freie Kapazitäten haben und die vom Kunden gewünschten Anforderungen mitbringen, können sich nun auf das Projekt als Experte bewerben. Jeder Bewerber muss vom Projektvermittler genehmigt werden, so wird sichergestellt, dass die benötigten Projektanforderungen erfüllt sind.
Wurden genug potenzielle Experten ausgewählt, kann der Projektvermittler die Bewerbungsfrist für das Projekt schließen und eine Liste der ausgewählten Bewerber an den Kunden übergeben. Dieser wählt nun den für ihn zum Projekt passenden Experten aus der Vorauswahl des Projektvermittlers aus und teilt dem Vermittler den gewählten Experten mit. Nach Auswahl des vom Kunden gewählten Experten durch den Projektvermittler werden automatisiert Zu- und Absagen an die Bewerber versandt.
Der in Abbildung 5.1 dargestellte Geschäftsprozess wurde in zwei Teilprozesse „Projektvermittlungsprozess“(Orange) und „Bewerberprüfungsprozess“(Blau) aufgeteilt. Durch das Anlegen eines neuen Projektes wird der Projektvermittlungsprozess gestartet. Bei Eingang einer neuen Bewerbung wird eine Instanz des Bewerberprüfungsprozesses eröffnet.

5.1.2 Individuelle Nutzeroberfläche

Bei der Implementierung wird eine eigene Tasklist in Form einer Vue.js-Webapplikation verwendet. Die Tasklist ist eine Art Übersichtsseite, auf der die aktuell offenen Aufgaben angezeigt werden. Hauptsächlich Anwendung findet dies, wenn Nutzer in das Abarbeiten von Aufgaben involviert sind. Einige Anbieter von Workflow-Management-Systemen bieten eigene Weboberflächen mit einer Standard-Tasklist an. Um individuelle Softwareentwicklung betreiben zu können, ist es jedoch wichtig, dass vom System die Möglichkeit zur Anbindung einer eigenen Tasklist (in Form einer Webapplikation) zur Verfügung gestellt wird. So können

Portale erstellt werden, welche speziell auf die Wünsche und Bedürfnisse des Kunden angepasst sind.
Essenziell für die optimale Nutzung der individuellen Benutzeroberfläche ist es, dass neue Prozessinstanzen erstellt und Aufgaben abgearbeitet werden können. Außerdem muss ein Zugriff auf die Prozessinstanzvariablen gegeben sein.

Projektvermittler-Ansicht

Um in die Projektvermittler-Ansicht zu gelangen, muss der Vermittler sich mittels Nutzername und Passwort authentifizieren. Die Ansicht verfügt über drei Hauptbestandteile: Ein Formular zur Erstellung neuer Projekte, eine Tasklist, in der die Aufgaben angezeigt werden und eine Übersicht der ausgeschriebenen Projekte.

Neues Projekt anlegen

ABBILDUNG 5.2: Formular für die Erstellung eines neuen Projektes

Auf der in Abbildung 5.2 gezeigten Seite werden neue Projekte vom Vermittler angelegt. Ein Projekt benötigt immer einen aussagekräftigen Titel, eine Beschreibung über die Projektspezifikationen und die vom Entwickler benötigten Eigenschaften wie beispielsweise Vorkenntnisse in bestimmten Programmiersprachen. Mit Klick auf Speichern wird eine neue Instanz des Projektvermittlungsprozesses gestartet.

Ausgeschriebene Projekte anzeigen

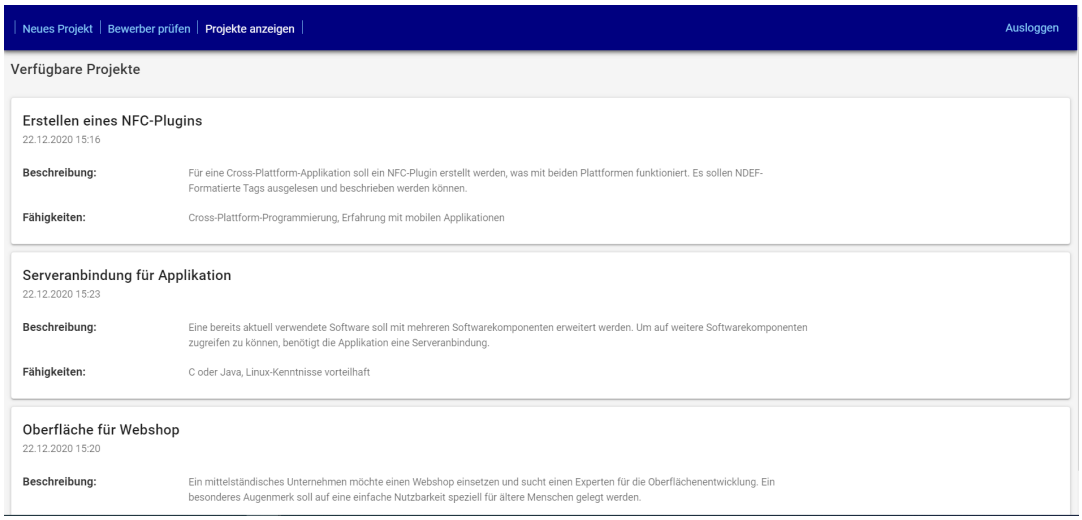

ABBILDUNG 5.3: Liste mit aktuell ausgeschriebenen Projekten

In Abbildung 5.3 kann der Projektvermittler eine Übersicht der aktuell ausgeschriebenen Projekte und deren Details einsehen.

Liste der Aufgaben

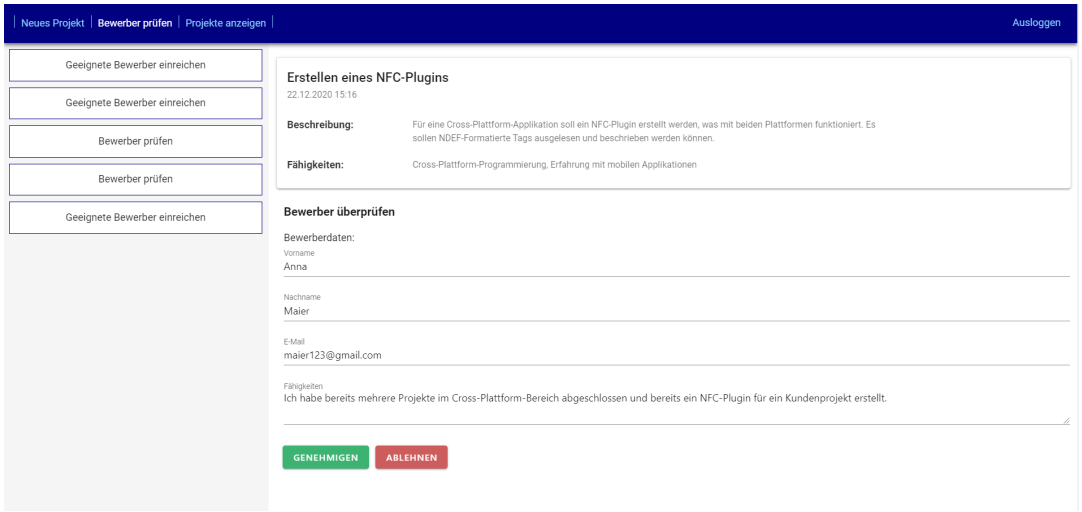

ABBILDUNG 5.4: Liste der Aufgaben mit Formular für die Überprüfung eines Bewerbers

Die Tasklist bildet den Hauptbereich des Projektvermittlers. In der in Abbildung 5.4 dargestellten Oberfläche kann der Projektvermittler in der linken Spalte seine aktuellen Aufgaben einsehen und diese abarbeiten. Der implementierte Geschäftsprozess beinhaltet drei Aufgabentypen, welche vom Projektvermittler zu bearbeiten sind.
Wurde ein neues Projekt angelegt, wird eine Aufgabe vom Typ „Geeignete Bewerber einreichen“ erstellt. Es wird das Projekt und eine Liste der bisher genehmigten Bewerber angezeigt. Wird eine neue Bewerbung eingereicht, bekommt der Vermittler eine Aufgabe vom Typ „Bewerber prüfen“ (siehe Abbildung 5.4). Bei diesem

Aufgabentyp kann der Vermittler die in der Bewerbung angegebenen Daten und das zugehörige Projekt einsehen, um zu überprüfen, ob der Bewerber für das Projekt geeignet ist. Mit Klick auf „Genehmigen“ wird der Experte zur Liste der genehmigten Bewerber hinzugefügt. Bei Nichteignung wird eine Benachrichtigung mit einer Absage an den Bewerber gesendet.

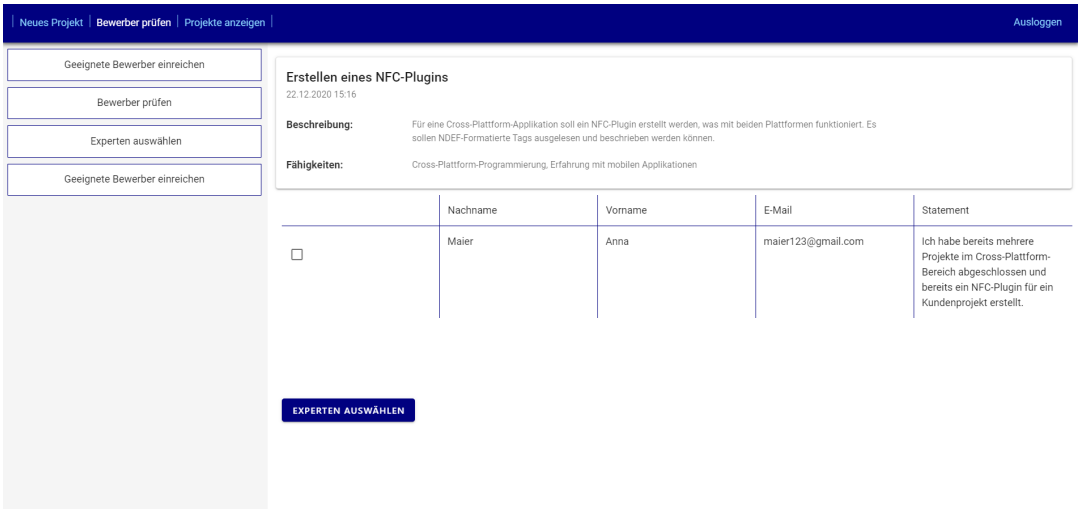

ABBILDUNG 5.5: Liste mit genehmigten Bewerbern zur Auswahl des Experten

Reicht der Vermittler die genehmigten Bewerber ein, werden diese an den Kunden gesandt. Der Kunde kann anschließend den für ihn geeigneten Experten auswählen und teilt seine Auswahl dem Vermittler mit. Als nächstes wählt der Projektvermittler nun bei der Aufgabe „Experten auswählen“ (siehe Abbildung 5.5) den vom Kunden gewünschten Experten aus. Im letzten Schritt werden automatisiert entsprechend Zu-/Absagen versandt.

Bewerber-Ansicht

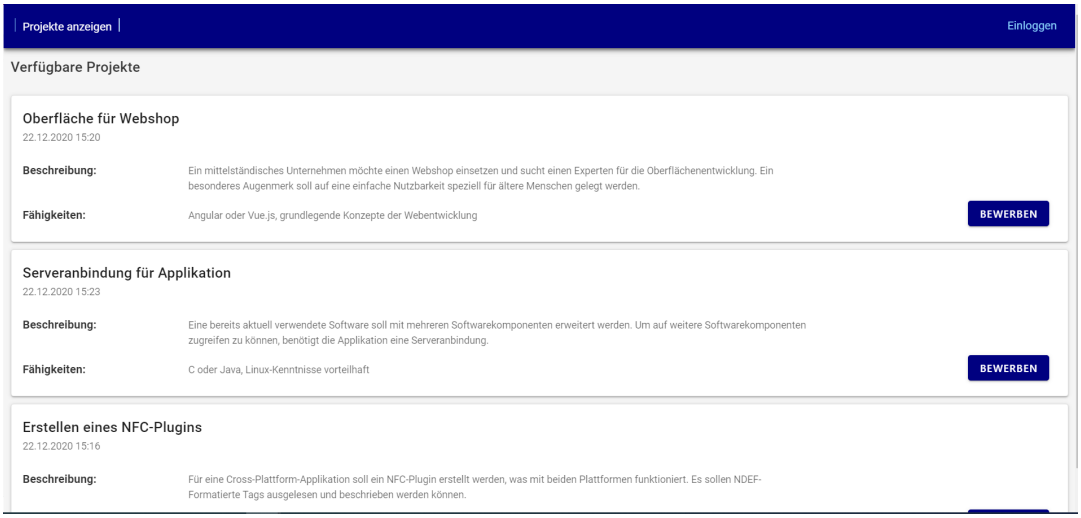

ABBILDUNG 5.6: Liste mit für Bewerbungen geöffneten Projekten

Für das Einreichen einer Bewerbung ist keine Anmeldung erforderlich. Der potenzielle Bewerber sieht eine Übersicht der verfügbaren Projekte (Abbildung 5.6) und

kann sich über den Button „Bewerben“ das in Abbildung 5.7 dargestellte Bewerbungsformular anzeigen lassen. Anschließend wartet der Bewerber auf eine Rückmeldung des Projektvermittlers.

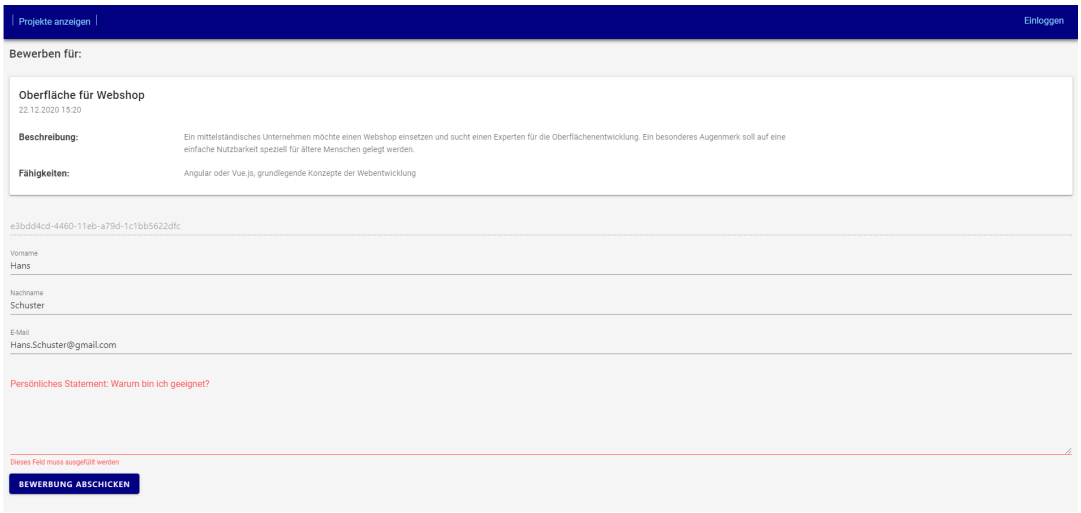

ABBILDUNG 5.7: Bewerbungsformular für Projekte

5.2 Camunda

In der folgenden Implementierung mit Camunda wird der graphisch modellierte Workflow in ein Java-Projekt eingebunden und mit Logik versehen.

Verwendete Software

Für die Implementierung wurde JavaEE und folgende Softwareversionen verwendet:

- Camunda Modeler v4.4.0
- Camunda Wildfly Bundle v7.9.0

5.2.1 Architektur

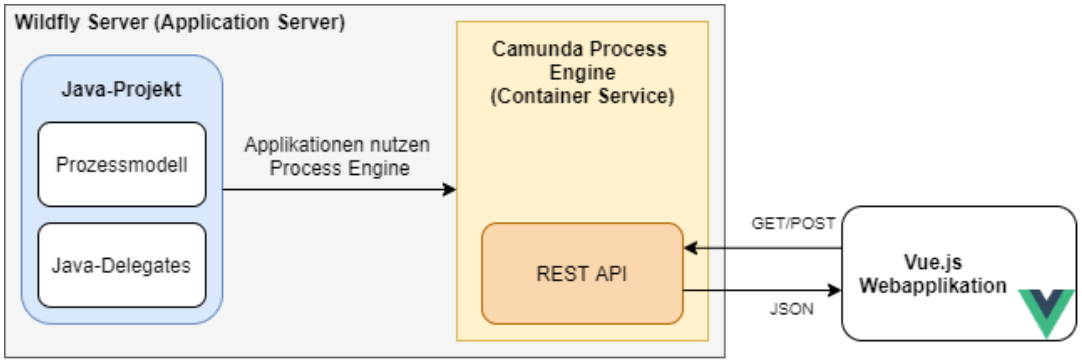

ABBILDUNG 5.8: Architekturübersicht der Camunda Applikation

Die in diesem Projekte verwendete Architektur wird in Abbildung 5.8 dargestellt. Camunda bietet ein Paket bestehend aus einer Application-Server-Installation und der Camunda-Engine an. Diese wird als Container-Service innerhalb des Servers allen Applikationen zur Verfügung gestellt.

Java-Projekt

Das Java-Projekt wird als .war-Datei auf den Application-Server deployed. Es enthält den graphisch modellierten Workflow und die Java-Klassen, welche das Modell mit Logik versehen. Als Grundgerüst dient der Camunda Servlet-Archetype.

Camunda-Engine

Die Camunda-Engine ist für das Ausführen des Workflows und das Steuern der Workflow-Instanzen zuständig. Die Engine beinhaltet einen REST-Endpunkt, über den der Zugriff auf die Instanzen und deren Variablen möglich ist.

Webanwendung

Für die Kommunikation zwischen der Camunda-Engine und einer Webanwendung werden HTTP-Anfragen verwendet. Die REST-Schnittstelle der Engine liefert die Antworten im JSON-Format.

5.2.2 Modellierung des Workflows

Der Workflow kann per Drag and Drop mit dem Camunda Modeler (siehe auch Marktanalyse: Camunda) oder per API im Code erstellt werden. Für die im Folgenden beschriebene Implementierung wurde die graphische Modellierung des Workflows verwendet.

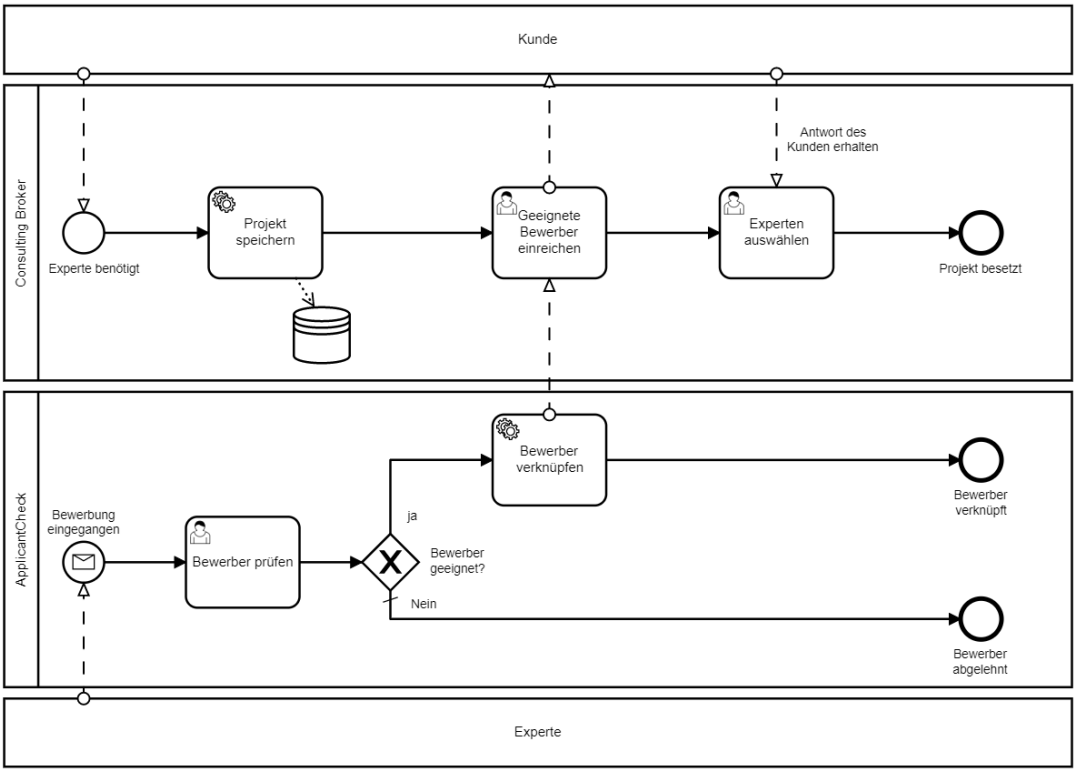

ABBILDUNG 5.9: Geschäftsprozess als Workflow modelliert im Camunda Modeler

Abbildung 5.9 zeigt den graphisch modellierten Workflow. Außerdem besteht die Möglichkeit, zusätzlich Änderungen in der zu Grunde liegenden XML-Datei vorzunehmen. Das Modell wird als Datei mit der Endung .bpmn abgespeichert.

5.2.3 Projektaufbau

Java-Projekt

Das Projekt wurde als Maven-Projekt mit einem Camunda Archetype erstellt. So kommt das Projekt bereits mit der Funktion, einen im Modeler erstellten Workflow auf einen Server zu deployen. Außerdem sind Basis-Testfälle enthalten.

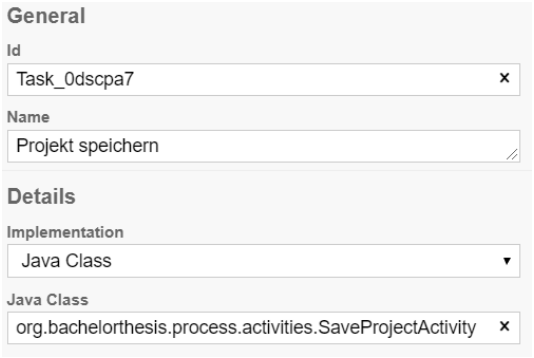

ABBILDUNG 5.10: Binden einer Java-Klasse an eine Service-Task in Camunda

In Camunda wird der Workflow durch das Binden von Klassen an Service-Tasks mit Logik versehen. Diese Service-Tasks sind automatisierte Aufgaben ohne menschliche Interaktion. Das Anbinden der Java-Klassen ist graphisch im Modeler möglich und ist beispielhaft in Abbildung 5.10 dargestellt.

```
1  public class SaveProjectActivity implements JavaDelegate{
2
3    @Override
4    public void execute(DelegateExecution execution) throws Exception {
5
6      String title = (String)execution.getVariable("title");
7      String description = (String)execution.getVariable("description");
8      String skills = (String)execution.getVariable("skills");
9      String projectProcessId = execution.getId();
10     execution.setVariable("id", projectProcessId);
11
12     DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy
       ↪  HH:mm:ss");
13     Date date = new Date();
14     String formattedTimestamp = dateFormat.format(date);
15
16     Project project = new Project(title, description, skills,
       ↪  formattedTimestamp, projectProcessId);
17
```

```
18      ObjectMapper mapper = new ObjectMapper();
19      execution.setVariable("project",
        ↪  mapper.writeValueAsString(project));
20
21    }
22  }
```

LISTING 5.1: An Aufgabe im Camunda Workflow gebundene Java-Klasse

Der Aufbau einer solchen Java-Klasse ist in Listing 5.1 dargestellt. Diese Klasse wird über ihren Klassennamen, wie in Abbildung 5.10 gezeigt, an die Aufgabe im Prozess angehängt. Beim Erreichen der Aufgabe im Workflow wird die execute-Methode der hinterlegten Klasse ausgeführt. Das mitgelieferte DelegateExecution-Objekt enthält eine Referenz auf die aktuelle Workflow-Instanz. In Listing 5.1 werden beispielsweise Variablen aus dieser Instanz gelesen und das daraus generierte Objekt zurück in die Workflow-Instanz geschrieben. Diese Daten befinden sich nun in der Workflow-Instanz und können im weiteren Ablauf verarbeitet werden.

Anbindung der Webapplikation

Camunda bietet eine umfangreiche REST-Schnittstelle, mit der auf die laufende Engine zugegriffen werden kann. Mittels dieser Schnittstelle kann auf jeden Schritt im Workflow zugegriffen werden. Auch das Erstellen neuer Aufgaben und das Fertigstellen von Aufgaben, welche menschliche Interaktionen erfordern, ist daher möglich. Außerdem wird ein Endpunkt für das Auslesen von Workflow-Instanzvariablen bereitgestellt.
Auch das Authentifizieren und Verwalten der Nutzer kann über diese Schnittstelle erfolgen. Die Ausgabe wird dabei im JSON-Format zurückgeliefert und kann von der Webapplikation verarbeitet werden.

5.3 Zeebe

Zeebe bietet bei der Implementierung die Möglichkeit, einen graphisch modellierten Workflow und die in diesem Absatz näher beschriebenen Job-Worker in einem Java-Projekt zu nutzen. So kann das Deployment auf den Zeebe Broker durch Code realisiert werden.

Verwendete Software

Das Java-Projekt wurde auf Basis von JavaEE implementiert. Die von Zeebe genutzten Microservices wurden in folgenden Versionen verwendet:

- Zeebe Broker v0.25.3
- Elasticsearch v6.8.1
- Camunda Operate v0.25.0
- Zeebe Hazelcast Exporter v0.9.1
- Zeebe Simple Tasklist v0.5.0
- ZeeQS v1.0.0

5.3.1 Architektur

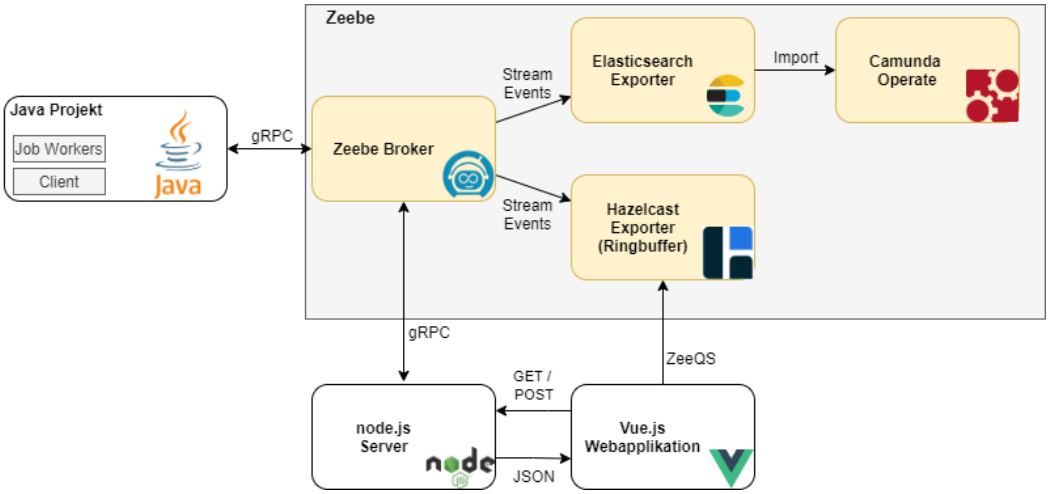

ABBILDUNG 5.11: Architektur der Zeebe Applikation

Für den Aufbau der in Abbildung 5.11 dargestellten Architektur wurde ein Java-Projekt, ein Zeebe Netzwerk aus verschiedenen Microservices und eine Webapplikation verwendet.

Java-Projekt

Das Java-Projekt beinhaltet die sogenannten Job-Worker. Diese Worker sind für das Ausführen der Logik in einem Job verantwortlich. Außerdem wird hier der Zeebe-Client genutzt. Dieser deployed den Workflow im Zeebe Broker. Die einzelnen Worker laufen unabhängig vom Zeebe Broker als Java-Projekt.

Zeebe

Zeebe verwendet einen Microservice-Ansatz. So werden der Zeebe Broker, der Elasticsearch-Exporter, Camunda Operate und der Hazelcast-Exporter jeweils als eigener Microservice zur Verfügung gestellt. In dieser Implementierung wurden die einzelnen Microservices (in Abbildung 5.11 gelb dargestellt) lokal ausgeführt.
Der Zeebe Broker verwendet einen ereignisgesteuerten Ansatz. Um dennoch auf vergangene Events zugreifen zu können, werden Exporter eingesetzt. Diese empfangen die vom Broker gesendeten Events und speichern diese ab. Der Elasticsearch-Exporter wird von Camunda Operate, der internen Nutzeroberfläche, verwendet. Der Hazelcast-Exporter speichert ebenfalls die vom Broker gesendeten Events ab und wird zur Kommunikation mit einer Webanwendung verwendet.

Webanwendung

Die Webanwendung gliedert sich in einen Node.js-Server, welcher via gRPC mit dem Zeebe Broker kommuniziert, und einer Vue.js-Anwendung, welche via *ZeeQS*[75] mit dem Hazelcast-Exporter kommuniziert, auf. Die Kommunikation zwischen diesen beiden Komponenten findet mittels HTTP-Anfragen statt.
Der Node.js-Server ist für die Steuerung der Workflow-Instanzen zuständig. Dieser Server kann neue Instanzen erstellen oder den Broker über abgearbeitete Aufgaben informieren.

Die Vue.js-Anwendung nutzt *ZeeQS*[75] um sich die aktuellen Daten und Variablen der laufenden Workflow-Instanz zu holen.

5.3.2 Modellierung des Workflows

Zeebe setze bei der graphischen Modellierung des Workflows auf einen Drag and Drop Ansatz mit eigenem Modeler. So können Aufgaben einfach an die benötigte Stelle gezogen und mittels Pfeilen verknüpft werden. Zeebe bietet jedoch keine direkte Möglichkeit, Aufgaben mit menschlicher Interaktion zu definieren.
Als Alternative kann der Workflow auch per Code beschrieben werden.
In dieser Implementierung wurde die graphische Modellierung verwendet.

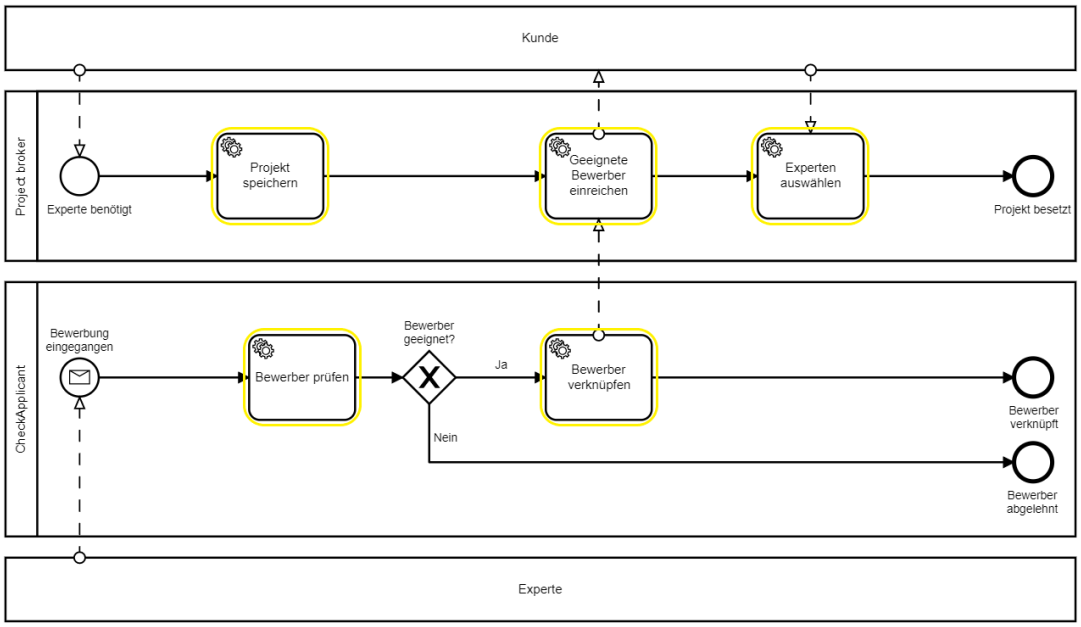

ABBILDUNG 5.12: Workflow modelliert im Zeebe Modeler

Der im Zeebe Modeler entworfene Workflow wird in Abbildung 5.12 dargestellt. Auch im Zeebe Modeler gibt es die Möglichkeit, die zu Grunde liegende XML-Datei direkt zu editieren und als .bpmn-Datei abzuspeichern.

5.3.3 Projektaufbau

Java-Projekt

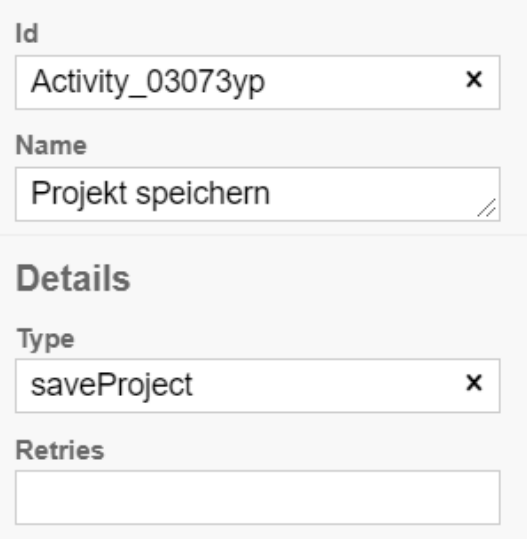

ABBILDUNG 5.13: Typ-Zuweisung in Zeebe Activity

Zeebe nutzt einen ereignisgesteuerten Ansatz. Dabei bekommen die einzelnen Activities (in Abbildung 5.12 gelb umrandet) im Modeler einen Typ zugewiesen. Dies wird in Abbildung 5.13 veranschaulicht.

```
1 client.newWorker().jobType("saveProject").handler(new
  ↪ SaveProjectWorker(client)).open();
```

LISTING 5.2: Hinzufügen eines Workers zu einem Aufgabentyp in Zeebe

Im Hauptprogramm können Klassen (sogenannte Job-Worker) erstellt werden, die wie Event-Handler fungieren. Anschließend muss dem Zeebe-Client im Code mitgeteilt werden, welcher Worker auf welchen Aufgaben-Typ hört. Dies wird mit dem in Listing 5.2 gezeigten Code erreicht. Die newWorker-Methode des Zeebe-Clients erstellt einen neuen Worker, welcher auf Aufgaben reagiert. Mittels jobType wird angegeben, durch welchen Aufgabentyp dieser Worker aktiviert wird. Außerdem muss noch eine Java-Klasse, die den auszuführenden Code abbildet, angegeben werden. Nach dem öffnen mittels open() ist der Worker bereit, auf Ereignisse zu reagieren.
Dadurch können auch Aufgaben mit menschlicher Interaktion realisiert werden, indem ein Worker auf den Typ „user“ hört und sich nicht beendet, bis ein externes „Job erledigt“ Kommando gesendet wurde. In dieser Implementierung wurde für Aufgaben mit menschlicher Interaktion das Projekt *zeebe-io/zeebe-simple-tasklist*[76] verwendet, welches diese Worker implementiert.

```java
public class SaveProjectWorker implements JobHandler{

ZeebeClient workflowClient;

public SaveProjectWorker(ZeebeClient cl) {
  this.workflowClient = cl;
}

@Override
public void handle(JobClient client, ActivatedJob job) throws
 ↪  Exception {

  String vars = job.getVariables();
  ObjectMapper mapper = new ObjectMapper();
  String skills = mapper.readValue(vars,
   ↪  ObjectNode.class).get("skills").asText();
  String description = mapper.readValue(vars,
   ↪  ObjectNode.class).get("description").asText();
  String title = mapper.readValue(vars,
   ↪  ObjectNode.class).get("title").asText();
  DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
  Date date = new Date();
  String formattedTimestamp = dateFormat.format(date);

  Project project = new Project(title, description, skills,
   ↪  formattedTimestamp, job.getWorkflowInstanceKey()+"");
  project.setId(job.getWorkflowInstanceKey());
  Map<String, Object> processVariables = job.getVariablesAsMap();
  processVariables.clear();
  processVariables.put("project", project);
  workflowClient.newSetVariablesCommand(job.getWorkflowInstanceKey()).
   ↪  variables(processVariables).send();
  client.newCompleteCommand(job.getKey()).variables(processVariables).
   ↪  send();
  }
}
```

LISTING 5.3: Implementierung eines Zeebe Workers

Listing 5.3 zeigt beispielhaft die Implementierung eines Workers. Dieser muss das JobHandler-Interface implementieren. Die handle-Methode enthält die Logik, welche beim Erreichen der Activity im Workflow ausgeführt werden soll. Als Übergabeparameter werden ein JobClient, welcher zum Ausführen jobspezifischer Kommandos dient, und eine ActivatedJob-Instanz mitgeliefert. Diese Instanz beinhaltet eine Referenz auf den aktuellen Job. So können Variablen aus diesem ausgelesen oder gesetzt werden. Zeebe speichert die Job-Variablen als String, weshalb diese erst in das gewünschte Format konvertiert werden müssen. Nach der Ausführung der eigentlichen Logik können die Job-Variablen mithilfe des Workflow-Clients in die

Workflow-Instanz gespeichert werden oder mittels eines Complete-Kommandos als Rückgabewert der Job-Ausführung gesetzt werden.

Anbindung der Webapplikation

Zeebe bietet keine direkte Schnittstelle zum Auslesen und Steuern der Workflow-Instanzen. Zur Steuerung des Workflows aus der Webapplikation wurde das ebenfalls von Zeebe veröffentlichte Projekt *Node.js client library for Zeebe Microservices Orchestration Engine*[77] verwendet. Zur Verwendung des Clients wurde ein Node.js-Server aufgesetzt, welcher dann als Schnittstelle für die Webanwendung dient.
Da Zeebe einen ereignisgesteuerten Ansatz verfolgt und keine direkte Möglichkeit zum Auslesen der Workflow-Instanzvariablen bietet, wird ein sogenannter Exporter benötigt. Dieser hört auf alle Events des Zeebe Brokers und speichert diese. So kann jederzeit auf Daten in der Workflow-Instanz zugegriffen werden. Zeebe bietet den *Zeebe-Hazelcast-Exporter*[78] an. Dieser speichert alle Events in einem Ringbuffer. Mittels eines Endpunktes, den der Zeebe Query Service *ZeeQS*[75] zur Verfügung stellt, können anschließend Daten aus den im Hazelcast-Exporter gespeicherten Events ausgelesen werden.

5.4 Bonita

Ein Bonita Projekt lässt sich auf zwei Arten realisieren. Einmal über die folgende Implementierung in Bonita Studio, aber auch über ein Java-Projekt. Wird der Weg über das Java-Projekt gewählt, muss der Workflow im Code beschrieben werden und es existiert kein graphisches Modell.

Verwendete Software

Für die Implementierung wurde folgende Software verwendet:

- Bonita Studio v2021.1 Build 7.12.1

5.4.1 Architektur

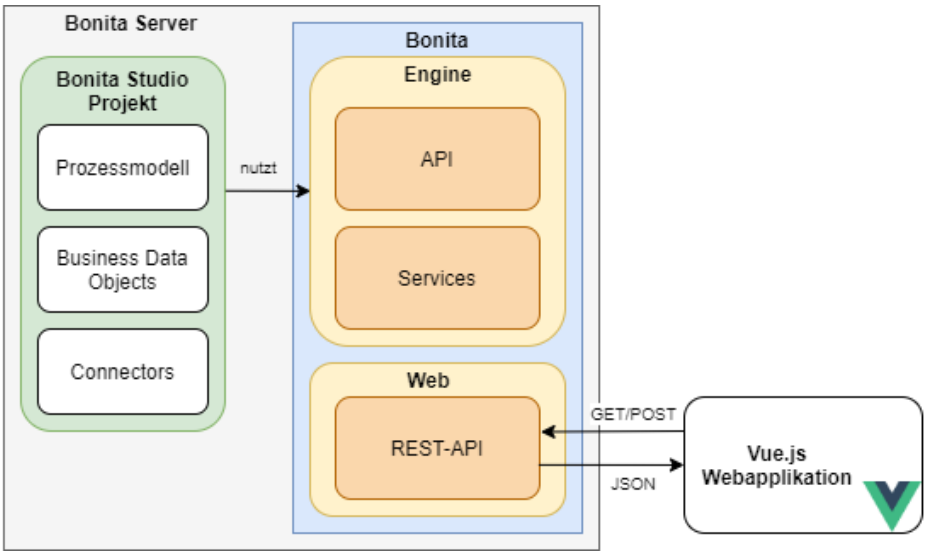

ABBILDUNG 5.14: Architekturübersicht der Bonita Applikation

Bonita setzt bei der Architektur (siehe Abbildung 5.14) auf einen Server, welcher das Projekt und die Engine enthält.

Bonita Studio Projekt

Bonita verwendet Bonita Studio als eigene Entwicklungsumgebung. Diese basiert auf der Entwicklungsumgebung Eclipse und enthält Funktionen, um Workflows zu modellieren, mit Logik zu versehen, zu bauen und auf einen Server zu deployen.
Das Bonita Studio Projekt enthält den graphisch modellierten Workflow. Außerdem können eigene Objekte erstellt werden. Durch Java-Code können die einzelnen Workflow-Aufgaben mit automatisierten Funktionen versehen werden.

Bonita-Engine & Bonita Web

Der Bonita Server enthält neben dem Studio Projekt ebenfalls die Engine. Sie wird vom Projekt mittels API genutzt und enthält BPM-relevante Services. Diese Services kümmern sich um den korrekten Ablauf des Workflows und bilden die eigentliche BPM-Logik in Bonita.
Zusätzlich zur Engine wird ein eigenes Webprojekt mit Möglichkeiten zur Verwaltung der Workflows geboten. Außerdem ermöglicht eine REST-API den Zugriff und die Steuerung von Workflow-Instanzen durch HTTP-Anfragen.

Webapplikation

Bonita besitzt eine umfangreiche REST-API. Diese bietet die Möglichkeit, Workflows und Workflow-Instanzen wie in dem beschriebenen Anwendungsfall, über eine Webapplikation zu erstellen und zu steuern. Mittels HTTP-Anfragen können so Daten an den Workflow oder einzelne Workflow-Instanzen übergeben oder ausgelesen werden.

5.4.2 Modellierung des Workflows

Bonita bietet, wie in Abschnitt 5.4.1 erwähnt, eine eigene Entwicklungsumgebung namens Bonita Studio. In diese Umgebung ist ein eigener Modeler zur graphischen Modellierung von Workflows integriert. Aufgaben können per Drag and Drop in den Workflow aufgenommen werden.

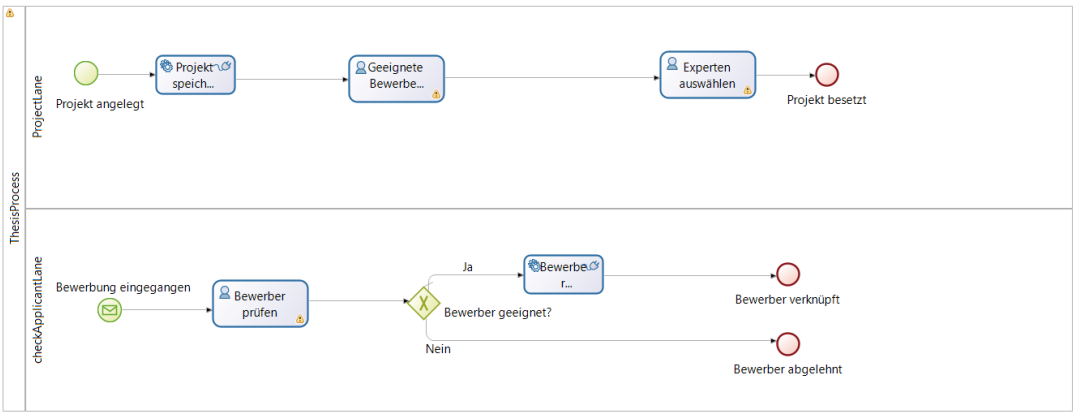

ABBILDUNG 5.15: Workflow modelliert in Bonita Studio

Der in Abbildung 5.15 gezeigte Workflow wurde in Bonita Studio modelliert. Außerdem wird die Möglichkeit geboten, BPMN 2.0-Diagramme zu importieren oder das als .proc-Datei erstellte Model im BPMN 2.0-Format zu exportieren.

5.4.3 Projektaufbau

Bonita Studio Projekt

Das zu Grunde liegende Projekt wurde in Bonita Studio erstellt. Die Projektstruktur (Abbildung 5.16) enthält die Organisationsdaten, Business-Objekte, Diagramme, Formulare sowie die zu den Connectoren gehörenden Definitionen und Implementierungen.
Unter Organisation können Rollen und Gruppen definiert werden. Diese werden für das Zuteilen von Aufgaben verwendet. So kann eine Aufgabe einem Nutzer oder einer Nutzergruppe zugeordnet werden.
Business-Objekte sind benutzerdefinierte Objekte, welche in eine eingebaute Datenbank geschrieben werden.
Die modellierten Workflows sind unter Diagrams gespeichert.
Wird die eingebaute Weboberfläche für das Abarbeiten von Aufgaben mit Nutzerinteraktion verwendet, können Formulare definiert werden, welche dann mit der entsprechenden Aufgabe verknüpft sind und dem Nutzer beim Abarbeiten dieser Aufgaben angezeigt werden.
Die Connectoren dienen dazu, automatisierte Aufgaben im Workflow mit Logik zu versehen oder die Anbindung an ein Drittsystem zu realisieren.

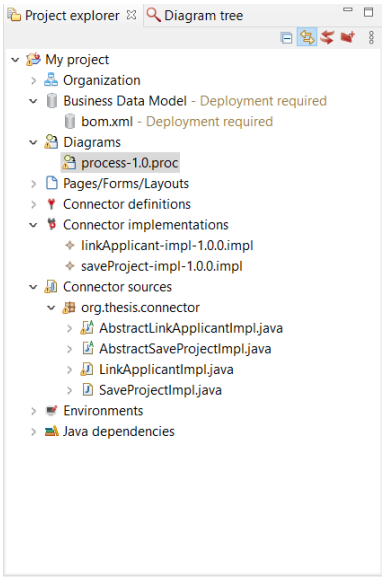

ABBILDUNG 5.16: Struktur des Bonita Studio Projektes

Ein Connector benötigt immer eine Definition, welche vorgibt, was für Eingabe- und Ausgabewerte der Connector nutzt. Diese Definition kann in Bonita Studio mit einem graphischen Tool durchgeklickt werden und erstellt anschließend eine XML-Datei, welche angibt, wie der Connector heißt und in welchem Paket die zugehörigen Klassen abgelegt sind.

```
1  public abstract class AbstractSaveProjectImpl extends
   ↪  AbstractConnector {
2
3    protected final static String TITLE_INPUT_PARAMETER = "title";
4    protected final static String DESCRIPTION_INPUT_PARAMETER =
     ↪  "description";
5    protected final static String SKILLS_INPUT_PARAMETER = "skills";
6    protected final String PROJECT_OUTPUT_PARAMETER = "project";
7
8    protected final java.lang.String getTitle() {
9      return (java.lang.String) getInputParameter(TITLE_INPUT_PARAMETER);
```

```
10    }
11
12    protected final java.lang.String getDescription() {
13      return (java.lang.String)
        ↪  getInputParameter(DESCRIPTION_INPUT_PARAMETER);
14    }
15
16    protected final java.lang.String getSkills() {
17      return (java.lang.String)
        ↪  getInputParameter(SKILLS_INPUT_PARAMETER);
18    }
19
20    protected final void setProject(com.company.model.Project project) {
21      setOutputParameter(PROJECT_OUTPUT_PARAMETER, project);
22    }
23
24    @Override
25    public void validateInputParameters() throws
      ↪  ConnectorValidationException {
26      try {
27        getTitle();
28      } catch (ClassCastException cce) {
29        throw new ConnectorValidationException("title type is invalid");
30      }
31      try {
32        getDescription();
33      } catch (ClassCastException cce) {
34        throw new ConnectorValidationException("description type is
          ↪  invalid");
35      }
36      try {
37        getSkills();
38      } catch (ClassCastException cce) {
39        throw new ConnectorValidationException("skills type is invalid");
40      }
41    }
42  }
```

LISTING 5.4: Definition eines Bonita Connectors

Anschließend können per Klick auf eine Option in der Entwicklungsumgebung die Implementierungsdateien erstellt werden. Dabei handelt es sich um eine abstrakte Klasse (Listing 5.4), welche die vorherige Definition an Eingabe- und Ausgabevariablen als Java-Code enthält. In Listing 5.4 ist beispielhaft die Definition des Connectors, welcher aus den Nutzereingaben Titel, Beschreibung und Skills ein Projekt-Objekt als Ausgabe erstellt, dargestellt.

```java
public class SaveProjectImpl extends AbstractSaveProjectImpl {

  @Override
  protected void executeBusinessLogic() throws ConnectorException{
    Project pro = new Project();
    pro.setTitle(getTitle());
    pro.setDescription(getDescription());
    pro.setSkills(getSkills());
    pro.setExecutionId(getExecutionContext().getProcessInstanceId()+"");

    DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
    Date date = new Date();
    String formattedTimestamp = dateFormat.format(date);

    pro.setDateCreated(formattedTimestamp);
    setProject(pro);

  }

  @Override
  public void connect() throws ConnectorException{
    //[Optional] Open a connection to remote server

  }

  @Override
  public void disconnect() throws ConnectorException{
    //[Optional] Close connection to remote server

  }

}
```

LISTING 5.5: Implementierung eines Bonita Connectors

Außerdem wird eine Implementierungsklasse erstellt, welche die Connector-Methoden beinhaltet. Diese Implementierung wird in Listing 5.5 dargestellt. Die connect-Methode kann verwendet werden, um eine Verbindung zu einem Drittsystem wie beispielsweise einem Remote-Server, herzustellen. Die disconnect-Methode dient dem Trennen dieser Verbindung.
Die Logik des Connectors wird in der executeBusinessLogic-Methode ausgeführt. In dem in Listing 5.5 dargestellten Fall wird aus den Eingabeparametern ein Objekt erstellt und als Ausgabewert gesetzt.
Dieser Connector kann nun an einen Schritt im Workflow angebunden werden. Bei der Anbindung besteht die Möglichkeit, Eingabeparameter zu spezifizieren und festzulegen, welche Workflow-Instanzvariable den Ausgabewert annehmen soll.

Anbindung der Webapplikation

Um die REST-Schnittstelle von Bonita nutzen zu können, muss ein Authentifizierungstoken im Header enthalten sein. Dieser wird über eine POST-Anfrage mit Nutzername und Passwort an einen Login-Service gewonnen. Anschließend kann die REST-Schnittstelle zur Steuerung und Verwaltung der Workflow-Instanzen und Workflow-Instanzvariablen verwendet werden. Bonita bietet mehrere Endpunkte, je nachdem auf welchen Bereich des Systems zugegriffen werden soll. Erfolgt beispielsweise eine Anfrage bezüglich der Steuerung eines Workflows, so wird diese an den BPM-Endpunkt gerichtet. Für die Nutzerauthentifizierung ist der Identity-Endpunkt zuständig. Dieser Ansatz gewährleistet eine gute Übersichtlichkeit. Die zurückgelieferte JSON-Antwort kann von der Webapplikation ausgelesen und verarbeitet werden.

5.5 Cadence

Im Folgenden wird die Implementierung des vorgestellten Beispielworkflows in einem Cadence-Projekt beschrieben. Cadence setzt dabei auf die Definition des Workflows in einem Java-Projekt. Der Cadence-Service führt den Workflow-Code nicht direkt aus, sondern liefert Events an den extern laufenden Worker-Prozess. Dieser Worker-Prozess behandelt die vom Cadence-Service gesendeten Events und liefert das Ergebnis zurück an diesen Service.

Verwendete Software

Für diese Implementierung wurde JavaEE und der bereitgestellte Docker-Container mit folgenden Microservices verwendet:

- Zookeper v3.4.6
- Kafka v2.12-2.1.1
- Elasticsearch v6.5.1
- Cadence v0.16.0
- Cadence Web v3.20.0

5.5.1 Architektur

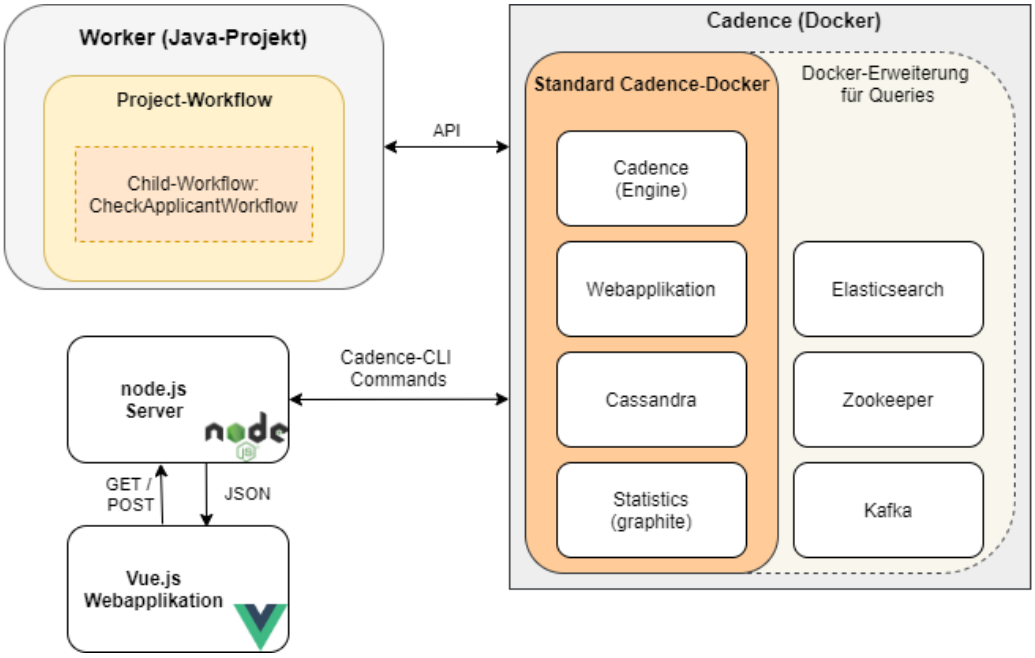

ABBILDUNG 5.17: Architektur der Cadence Applikation

Die in Abbildung 5.17 gezeigte Architektur der Cadence-Applikation gliedert sich in drei wesentliche Bestandteile auf: den Worker, den Cadence Docker und die Webanwendung.

Worker

Das Java-Projekt bildet den sogenannten Worker. Der Worker beinhaltet die Implementierung des Workflows. Dieser Workflow wird in Cadence als Java-Klasse implementiert. Die Klassenmethoden bilden die einzelnen Aufgaben im Workflow ab. Klassenvariablen fungieren in diesem Fall als Workflow-Instanzvariablen. Im implementierten Workflow beinhaltet der „Project-Workflow“, welcher für das Erstellen der Projekte und Verwalten der Bewerber zuständig ist, den Kind-Workflow „CheckApplicantWorkflow“. Dieser Kind-Workflow dient zur Überprüfung neu eingegangener Bewerbungen und verknüpft diese gegebenenfalls mit dem Projekt. Ein Worker kann dabei auch mehrere Workflows beinhalten. Diese werden durch Code zu Beginn des Programms im Worker registriert.

Cadence Docker

Das Herzstück der Anwendung bildet der Cadence Docker. Dieser Container beinhaltet die Standard-Microservices. Dazu gehört die Engine, welche für die Verwaltung der Workflows zuständig ist, die Webapplikation für eine graphische Übersicht der laufenden Workflow-Instanzen, Cassandra als Datenbank und Graphite um Statistiken zu den Workflow-Instanzen und deren Ablauf einsehen zu können.
In dieser Implementierung wurde der erweiterte Docker-Container verwendet, welcher zusätzliche Microservices beinhaltet, um den Kommandozeilen Kommandos weitere Filterfunktionen hinzuzufügen.

Webapplikation

Um die Workflows durch eine Webanwendung zu steuern bietet Cadence ein eigenes Command-Line-Interface. Um diese Kommandos abzusetzen, wird ein Node.js-Server verwendet. Dieser ruft die Kommandos auf und bekommt die Ausgabe, welche eigentlich auf der Kommandozeile ausgegeben wird, zurückgeliefert. Die Vue.js-Anwendung kommuniziert mittels HTTP-Anfragen mit diesem Node.js-Server und bekommt die Daten als JSON zurückgeliefert.

5.5.2 Modellierung des Workflows

Cadence bietet keine Möglichkeit, Workflows graphisch darzustellen oder zu modellieren. Die Workflows werden durch Java-Klassen im Code beschrieben.

5.5.3 Projektaufbau

Java-Project

In Cadence wird ein Workflow, wie in Abschnitt 5.5.1 erwähnt, durch eine Java-Klasse repräsentiert. Diese Klasse besitzt eine Klassenmethode, welche beim Start des Workflows ausgeführt wird (eine sogenannte Workflow-Methode). Zur Erhöhung der Ausfallsicherheit werden „Activities“ zur Anbindung an Drittsysteme verwendet. Diese können aus der Workflow-Methode ausgeführt werden. So wird sichergestellt, dass bei Problemen mit einem Drittsystem nicht die Ausführung des Workflows beeinträchtigt wird.
Ein Workflow besteht dabei aus einem Interface, in dem die Workflow-spezifischen Methoden (vergleichbar mit Aufgaben in einem BPMN-Diagramm, siehe Unterabschnitt 5.2.2) definiert werden und einer Implementierung dieses Interfaces, in der die Methoden mit Funktion versehen werden.

```java
public interface ProjectWorkflow {

  @WorkflowMethod(executionStartToCloseTimeoutSeconds = 20160,
  ↪  taskList = "ThesisTasklist")
  public void execute(Object obj);

  @SignalMethod(name = "userTaskFinished")
  public void userTaskFinished();

  @SignalMethod(name = "newApplicant")
  public void newApplicant(Object obj);

  @SignalMethod(name = "selectExpert")
  public void selectExpert(Object obj);

  @QueryMethod(name = "getProcessVariables")
  public ProcessVariables getVariables();

}
```

LISTING 5.6: Cadence Implementierung des „Experte benötigt“ Workflows

Die Methoden werden mit Annotationen versehen (siehe Listing 5.6), welche der zu Grunde liegenden Engine die Funktion der Methoden mitteilen. Die Workflow-Methode bildet den eigentlichen Workflow. Diese wird beim Start des Workflows ausgeführt und steuert den Ablauf der Aufgaben in Form von Methodenaufrufen.
Die Signal-Methoden werden verwendet, um ein externes Signal zu verarbeiten. So können beispielsweise Aufgaben mit menschlicher Interaktion realisiert werden. Dabei wird in der Workflow-Methode auf ein Ergebnis aus der Signal-Methode gewartet.
Über Query-Methoden können Workflow-Instanzvariablen ausgelesen werden. Diese sind in Cadence einfache Klassenvariablen der Workflow-Implementierung.

Das Modell kann man sich dazu wie folgt vorstellen: Service-Tasks (Aufgaben ohne menschliche Interaktion) werden direkt in der Workflow-Methode durch Methodenaufrufe ausgeführt. Aufgaben mit menschlicher Interaktion werden über Signal-Methoden realisiert, auf deren Ergebnis die Workflow-Methode wartet.

Anbindung der Webapplikation

Cadence wird in einem Docker-Container bereitgestellt. Die Workflow-Instanzen werden über ein im Docker bereitgestelltes Command-Line-Interface gesteuert. Über dieses können Signale zum Auslösen der Signal-Methoden gesendet werden, aber auch Workflow-Instanzen gestartet oder Query-Methoden zum Auslesen der Workflow-Instanzvariablen aufgerufen werden. Beim Auslösen eines Signals oder Erstellen einer Workflow-Instanz können Variablen übergeben werden. In Listing 5.6 sind diese als Übergabeparameter der entsprechenden Methode realisiert.
Cadence arbeitet mit Domains um mehrere Workflows logisch von einander zu

trennen. Die Command-Line-Interface Kommandos werden dabei immer auf einer angegebenen Domain ausgeführt.

```
1  docker run --network=host --rm ubercadence/cli:master --do thesis
   ↪  workflow list --query "WorkflowType='ProjectWorkflow::execute'
   ↪  AND CloseTime = missing"
```

LISTING 5.7: Beispiel eines Command-Line-Interface Kommandos mit Query in Cadence

Das in Listing 5.7 aufgeführte Kommando listet alle laufenden Instanzen des Workflows „ProjectWorkflow::execute“, wobei execute die Workflow-Methode ist, auf. Das Kommando „workflow list“ wird im Docker auf der Domain „thesis“ ausgeführt. Dadurch werden alle Workflow-Instanzen in dieser Domain angezeigt, auch bereits beendete. Durch eine Query kann das Ergebnis weiter eingeschränkt werden, beispielsweise wie hier nur auf die aktuell laufenden Instanzen eines bestimmten Workflows.

Kapitel 6

Analyse und Auswertung der Workflow-Management-Systeme unter dem Aspekt der Anwendbarkeit von Software-Engineering-Konzepten

6.1 Camunda

In diesem Kapitel werden die Kriterien anhand der vorangegangenen beispielhaften Implementierung eines Workflows in Camunda geprüft.

6.1.1 Entwicklungswerkzeuge

Versionsverwaltung

Camunda unterstützt bei der Modellierung von Workflows den BPMN 2.0-Standard[79]. So ist das Modell grundsätzlich eine XML-Datei. Die Delegates, welche die Schritte im Workflow mit Logik versehen, werden in Java geschrieben. Somit verwendet Camunda kein proprietäres Format. Da Versionskontrollsysteme wie GIT mit allen gängigen Dateiformaten nutzbar sind[80], ist mit Camunda die Nutzung einer Versionsverwaltung möglich.

Build-Management-Tools

Der Einsatz von Build-Management-Tools wird in Camunda unterstützt. Die Archetypes werden als Maven Projekte angeboten.[79] Maven bietet eine Konfigurationsdatei im Projekt an, in welcher die benötigten Bibliotheken und Abhängigkeiten definiert werden können. Außerdem besteht die Möglichkeit, das Format der Ausgabe-Datei zu spezifizieren. Beim Build-Vorgang werden die Daten aus dieser Datei ausgelesen und angewandt.[4] Dies hat den Vorteil, dass keine einzelnen JAR-Dateien als Abhängigkeiten heruntergeladen und manuell in den Buildpfad eingepflegt werden müssen.

Testabdeckung

Camunda bietet zusätzliche Werkzeuge zum Testen von Workflows. So existiert eine Bibliothek, die es ermöglicht, die Testabdeckung im graphischen Workflow-Modell zu veranschaulichen. Die bei den Service-Tasks zu Grunde liegenden Java-Delegates

können durch JUnit-Tests abgedeckt werden. Dies wurde in der Implementierung überprüft (siehe Listing A.1 im Anhang). Lediglich der in Script-Tasks direkt im Workflow-Modell verfasste Code kann nicht durch JUnit- Tests abgedeckt werden. Dieser Code wird in das XML des Prozessmodells integriert und müsste daher aus dieser XML-Datei extrahiert werden, um es testen zu können.

Continuous Integration/Continuous Delivery

Für das automatisierte Testen, Bauen und Ausliefern einer Camunda-Anwendung können Tools wie Jenkins verwendet werden[81]. Jenkins ist das am häufigsten verwendete Open Source Automatisierungstool[8] .

6.1.2 Systemeigenschaften

Kompatible Standards

Die Camunda Services GmbH war aktiv daran beteiligt den BPMN 2.0-Standard (siehe Abschnitt 2.1.5) für die Prozessmodellierung mit zu definieren[45]. Der Modeler unterstützt daher diesen Standard.
Für die Programmierung und Verwaltung eines Workflows lässt sich Camunda, wie in der Implementierung gezeigt, in eine Java-Anwendung integrieren. Im modellierten Workflow können dann die entsprechenden Java-Klassen eingebettet werden, um den Workflow mit Logik zu versehen. Für die Anbindung des Java-Projektes an die Engine bietet Camunda Java-Bibliotheken, die in das Projekt eingebunden werden.
Bezüglich der Entwicklungsumgebung besteht keine Vorgabe seitens Camunda. Die zur Verfügung gestellten vorgefertigten Projektkonstrukte können über jede Umgebung, welche Maven unterstützt, oder über die Kommandozeile angelegt werden. Die Programmierung kann anschließend in einer beliebigen Entwicklungsumgebung oder auch in einem einfachen Texteditor stattfinden.

Entwicklungsfreiheit

Camunda bietet für Softwareentwickler einige Hilfestellungen bei der Entwicklung an, die aber nicht zwingend genutzt werden müssen. Zum Modellieren des Workflows kann der Camunda Modeler(Abschnitt 3.5) als graphisches Tool verwendet werden. Da der BPMN 2.0-Standard (siehe Abschnitt 2.1.5) unterstützt wird, kann das Modell auch mittels einer einfachen XML-Datei erstellt werden.
Außerdem bietet Camunda Archetypes als Grundgerüst für Java-Projekte an. Diese beinhalten bereits grundsätzliche Funktionen, wie beispielsweise das Deployment von Workflows auf einem Server. Es besteht kein Zwang, diese vorgefertigten Projektkonstrukte zu verwenden.

Anbindung an Webapplikation

Für die Anbindung an eine Webapplikation stellt Camunda eine in Abschnitt 5.1 beschriebene REST-Schnittstelle zur Verfügung. Durch HTTP-Anfragen an diese Endpunkte kann aus einer Webapplikation in vollem Umfang auf die Funktionen der Engine zugegriffen werden. Dies hat den Vorteil, dass man zum Verwalten der Workflow-Instanzen keine weitere Softwarekomponente benötigt und so eine einfache Möglichkeit erhält, eine eigene Tasklist zu implementieren.

Typsicherheit der Prozessinstanzvariablen

Der Zugriff auf die Workflow-Instanzvariablen erfolgt in Camunda über ein execution-Objekt.

```
1  public void execute(DelegateExecution execution) throws Exception {
2
3    String title = (String)execution.getVariable("title");
4    String description = (String)execution.getVariable("description");
5    String skills = (String)execution.getVariable("skills");
6    String projectProcessId = execution.getId();
7    execution.setVariable("id", projectProcessId);
8
9    DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
10   Date date = new Date();
11   String formattedTimestamp = dateFormat.format(date);
12
13   Project project = new Project(title, description, skills,
      ↪  formattedTimestamp, projectProcessId);
14
15   ObjectMapper mapper = new ObjectMapper();
16   execution.setVariable("project", mapper.writeValueAsString(project));
17
18 }
```

LISTING 6.1: Zugriff auf Camunda Workflow-Instanzvariablen aus Java-Projekt

In Listing 6.1 wird in einem Ausschnitt aus der Implementierung gezeigt, wie das Setzen und Auslesen von Workflow-Instanzvariablen in Camunda gehandhabt wird. Dieser Ansatz gewährleistet keine Typsicherheit, da die Variablen über deren Namen aufgerufen und gesetzt werden. So ist nicht definiert, welchen Typ eine Variable besitzt und es kann beim Auslesen nicht sichergestellt werden, dass diese den erwarteten Typ besitzt, da der Typ zur Laufzeit geändert werden könnte.

6.2 Zeebe

Dieses Kapitel wertet die Umsetzung der Kriterien im Workflow-Management-System Zeebe aus.

6.2.1 Entwicklungswerkzeuge

Versionsverwaltung

Da Zeebe, wie in der Implementierung beschrieben, den BPMN 2.0-Standard verwendet und für die Programmierung auf Hochsprachen setzt, ist eine Nutzung der Versionsverwaltung für Zeebe Projekte möglich.

Build-Management-Tools

Der für den dargestellten Anwendungsfall verwendete Java-Client lässt sich mit allen gängigen Build-Management-Tools integrieren[82]. Die benötigten Bibliotheken können dabei als Abhängigkeiten eingebunden und durch das Build-Management-Tool verwaltet werden.

Testabdeckung

Für den Code der Zeebe Worker ist eine volle Abdeckung durch JUnit-Tests möglich, da es sich dabei um Java-Klassen handelt. Dies wurde in der Implementierung überprüft (siehe Listing A.2 im Anhang). Zeebe bietet einen Testcontainer als Docker an, empfiehlt jedoch, die Tests auf der Umgebung, in welcher der Code zukünftig laufen wird, durchzuführen. So kann sichergestellt werden, dass der Code auch später auf der Produktivumgebung lauffähig und durch die Tests abgedeckt ist.

Continuous Integration/Continuous Delivery

Da die Worker in Zeebe, wie in Abschnitt 5.3.1 erwähnt, als Java-Projekt entwickelt werden, kann die Anwendung mittels Continuous Integration und Continuous Delivery automatisiert gebaut und ausgeliefert werden. Tools wie Jenkins können für das automatisierte Bauen und Ausliefern von Java-Anwendungen verwendet werden[83].

6.2.2 Systemeigenschaften

Kompatible Standards

Zeebe unterstützt für die Modellierung von Workflows den BPMN 2.0-Standard, daher kann das Modell im Zeebe Modeler graphisch entworfen oder als XML-Datei beschrieben werden.
Das Projekt wird in Zeebe nicht auf einen Server deployed, sondern als eigene Anwendung gestartet und greift lediglich auf den Zeebe Broker zu. Daher kann das Projekt in jeder Programmiersprache, in der Zeebe einen Client zur Verfügung stellt, implementiert werden.
Clients in Java und Go sind direkt von Zeebe entwickelt. Zusätzlich gibt es weitere Projekte mit Clients in diversen Programmiersprachen, deren Entwicklung von der Community unterstützt wird.[84] Um die Spanne an unterstützten Programmiersprachen stetig zu erweitern, bietet Zeebe der Community Anleitungen für die Implementierung eines Clients.

Entwicklungsfreiheit

Zeebe legt nicht fest, mit welcher Sprache oder Entwicklungsumgebung gearbeitet werden muss. Da dieses System auf einem Microservice-Ansatz basiert, können die verschiedenen Komponenten selbst zusammengestellt werden und sind nach den Bedürfnissen des Projektes anpassbar.

Anbindung an Webapplikation

Durch den Microservice-Ansatz von Zeebe und die Verfügbarkeit eines Clients in Node.js, lassen sich auch Webapplikationen, wie in Abschnitt 5.3 erläutert, an Zeebe anschließen. Es wird keine direkte REST-Schnittstelle von Zeebe zur Verfügung

gestellt. Da Zeebe auf Events basiert, ist es nicht möglich, Daten direkt vom Broker abzufragen. Durch die hohe Vielfalt an Microservices, die mit Zeebe integrierbar sind, ist es jedoch über sogenannte Exporter möglich, Daten der Workflow-Instanzen abzufragen. Diese Exporter hören die Events des Brokers ab und speichern diese, sodass auch nachdem ein Event bearbeitet wurde, die Daten verfügbar bleiben.

Typsicherheit der Prozessinstanzvariablen

In Zeebe können Workflow-Instanzvariablen ausgelesen oder gesetzt werden.

```
1 Map<String, Object> processVariables = job.getVariablesAsMap();
2 processVariables.put("project", project);
3 workflowClient.newSetVariablesCommand(job.getWorkflowInstanceKey()).
  ↪ variables(processVariables).send();
```

LISTING 6.2: Zugriff und Setzen der Zeebe Workflow-Instanzvariablen aus Java-Projekt

Die Variablen sind als Schlüssel-Werte-Paare in der aktuellen Aufgabe abrufbar. (siehe Listing 6.2)
Durch das Setzen und Auslesen der Variablen über deren Namen kann keine Typsicherheit gewährleistet werden, da durch Tippfehler oder Verwechslung der Variablennamen die ausgegebene oder gesetzte Variable nicht mehr stimmen würde.

6.3 Bonita

In diesem Unterkapitel wird geprüft, inwieweit Bonita die gewählten Kriterien erfüllt.

6.3.1 Entwicklungswerkzeuge

Versionsverwaltung

Bonita beschreibt in *Bonita 7.10 - How to use Git for versioning with Community Edition*[85] die Integration von GIT mit Bonita Studio. In der Open Source Version kann in Bonita Studio nur ein Workspace mit einem Projekt verwendet werden. Der Speicherort dieses Workspaces lässt sich dadurch nicht anpassen. Dies bedeutet, dass ein erstelltes Repository im von Bonita erstellen Workspace Ordner liegt. Ein Verschieben dieses Repositorys, beispielsweise zur Übersichtlichkeit, ist nicht möglich. Außerdem können Schwierigkeiten beim Zusammenführen verschiedener Versionen des Worfklow-Modells auftreten, da die .proc-Datei keinen Standard erfüllt. Der Entwickler muss sich hier erst in die nicht standardisierten Tag-Namen einarbeiten um die Dateien zusammenführen zu können.

Build-Management-Tools

Der Einsatz von Build-Management-Tools wird in Bonita Studio nicht unterstützt. In dem erstellten Projekt ist keine Build-Management-Datei, die für das Verwalten von Abhängigkeiten zuständig ist, enthalten. Außerdem müssen Abhängigkeiten vom

Entwickler als .jar-Dateien importiert werden. Dies zeigt, dass Bonita Studio keine Build-Management-Tools für das Bauen der Anwendung und das Verwalten von Abhängigkeiten unterstützt.

Testabdeckung

Bonita Studio bietet integrierte Ansätze für das Testen der Anwendung. So wird der modellierte Workflow über eine graphische Validierung auf Fehler überprüft. Diese Validierung prüft beispielsweise, ob alle User-Tasks Formulare angebunden haben. Die Connectoren können über eine graphische Oberfläche testweise durchlaufen werden. Diese Funktion wurde in der Implementierung angewendet. Auch für Ausdrücke (sogenannte Expressions) in den einzelnen Aufgaben ist eine Evaluationsfunktion integriert. Diese graphischen Testfunktionen sind jedoch nicht vergleichbar mit JUnit-Tests. Ein in Bonita Studio entworfener Workflow kann als .bar-Datei exportiert und in einem Java-Projekt mittels JUnit-Tests getestet werden[86].

Continuous Integration/Continuous Delivery

Bonita unterstützt Continuous Integration. Zur Nutzung wird ein Bonita eigenes Build-Skript benötigt[87]. Dieses entspricht keinem Standard und bedeutet deshalb einen höheren Einarbeitungsaufwand für die Entwickler.
Für die Nutzung von Continuous Integration und Continuous Delivery kann in Bonita Jenkins verwendet werden[88].

6.3.2 Systemeigenschaften

Kompatible Standards

Bonita unterstützt den BPMN 2.0-Standard für die Modellierung von Workflows[67]. Die Datei wird in Bonita Studio als .proc-Datei abgespeichert, kann jedoch auch im BPMN-Format exportiert oder importiert werden.

```
1  <elements xmi:type="process:StartEvent"
   ↪  xmi:id="_79ERE23gEeuR1onY0bCukQ" name="Projekt angelegt"
   ↪  outgoing="_79ERKG3gEeuR1onY0bCukQ">
2  <dynamicLabel xmi:type="expression:Expression"
   ↪  xmi:id="_79ERFG3gEeuR1onY0bCukQ" name="" content=""
   ↪  returnTypeFixed="true"/>
3  <dynamicDescription xmi:type="expression:Expression"
   ↪  xmi:id="_79ERFW3gEeuR1onY0bCukQ" name="" content=""
   ↪  returnTypeFixed="true"/>
4  <stepSummary xmi:type="expression:Expression"
   ↪  xmi:id="_79ERFm3gEeuR1onY0bCukQ" name="" content=""
   ↪  returnTypeFixed="true"/>
5  </elements>
```

LISTING 6.3: Definition des Start-Elements „Projekt anlegen“ in .proc-Datei

In Listing 6.3 wird ein Auszug aus dieser .proc-Datei dargestellt. Zu sehen ist das Start-Element „Projekt anlegen“ in XML. Dadurch wird ersichtlich, dass dem graphischen Workflow-Modell eine XML-Datei zugrunde liegt.

Entwicklungsfreiheit

Die Entwicklungsfreiheit in Bonita ist eingeschränkt. Bonita Studio bietet eine vollumfängliche Möglichkeit, Workflows zu modellieren und mit Logik zu versehen(siehe Abschnitt 5.4). Möchte der Entwickler diese eigene Entwicklungsumgebung nicht nutzen, kann ein Workflow auch per Java-Code beschrieben und mit Logik versehen werden. Das hat allerdings den Nachteil, dass kein graphisches Modell des Workflows existiert und die Connectoren in eigene Projekte ausgelagert werden müssen[89].

Anbindung an Webapplikation

Die Anbindung an eine Webapplikation funktioniert in Bonita über eine in Abschnitt 5.5 beschriebene REST-Schnittstelle. Über sehr übersichtlich gestaltete Endpunkte kann die gesamte Applikation gesteuert werden. So existiert für jeden Teilbereich wie beispielsweise das Identifizieren von Nutzern oder das Steuern des Workflows ein eigener Endpunkt. Als Antwort auf HTTP-Anfragen wird ein JSON zurückgeliefert.

Typsicherheit der Workflow-Instanzvariablen

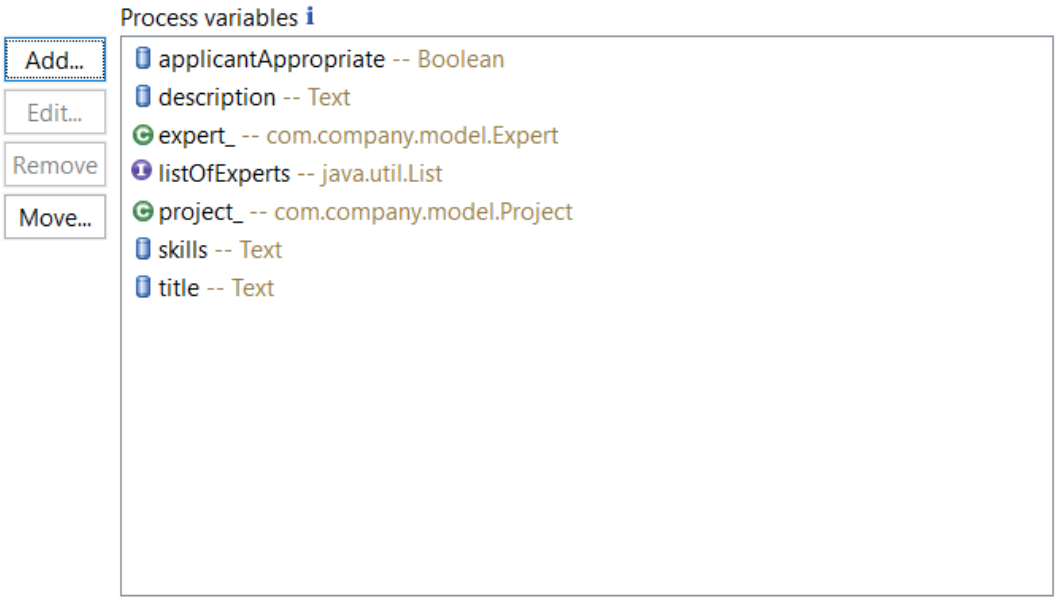

ABBILDUNG 6.1: Übersicht der Variablen eines Bonita Workflows

Bonita kann die Typsicherheit der Workflow-Instanzvariablen gewährleisten, da diese in einer graphischen Oberfläche angelegt werden (Abbildung 6.1). So kann der Entwickler den einzelnen Variablen spezifische Typen zuweisen. Auch die Ein- und Ausgabe-Variablen eines Connectors können per graphischer Oberfläche zugewiesen werden. Dadurch ist zu jeder Zeit ersichtlich, welchen Typ eine Variable besitzt.

6.4 Cadence

In diesem Kapitel wird das Workflow-Management-System Cadence anhand der entworfenen Kriterien überprüft.

6.4.1 Entwicklungswerkzeuge

Versionsverwaltung

Ein Cadence-Projekt kann in Kombination mit einer Versionsverwaltung wie GIT verwendet werden, da die Implementierung auf Java-Klassen beruht[80]. Cadence nutzt statt graphisch modellierten Workflows, wie in Abschnitt 5.5.3 beschrieben, Java-Klassen mit entsprechenden Annotationen. Dadurch können verschiedene Versionen des Codes zusammengeführt werden.

Build-Management-Tools

Cadence bietet für die Entwicklung eine Bibliothek mit Befehlen zur Nutzung der Engine an[90]. Da dieses System nur auf Java-Code basiert und diese Bibliothek zur Integration mit gängigen Build-Management-Tools zur Verfügung steht, ist die Nutzung dieser Tools möglich.

Testabdeckung

JUnit-Tests lassen sich in Cadence unkompliziert erstellen. Durch die Definition und Implementierung des Workflows im Code ist es möglich, eine vollumfängliche Testabdeckung zu erreichen. Dies wurde durch eine Implementierung überprüft (siehe Listing A.3 im Anhang).

Continuous Integration/Continuous Delivery

Die Nutzung einer Continuous Integration und Continuous Delivery Pipeline wird von Cadence unterstützt, da es sich beim Projekt um ein normales Java-Projekt handelt. Tools wie Jenkins unterstützen das automatisierte Bauen und Ausliefern von Java-Anwendungen[91].

6.4.2 Systemeigenschaften

Kompatible Standards

Cadence unterstützt, wie in Unterabschnitt 5.5.2 gezeigt, keinen Standard für die Modellierung von Workflows. Die einzelnen Aufgaben werden über Methoden in einer Java-Klasse (siehe Abschnitt 5.5.3) definiert. Das graphische Workflow-Modell ist bedeutend, um die Kommunikation zwischen Fachbereich und Softwareentwickler zu vereinfachen. Die Definition des Workflows im Code könnte dadurch zu Problemen bei der Verständigung führen. Oft bringt der Fachbereich wenig bis kein Wissen im Bereich Programmierung mit und kann daher den Workflow nicht nachvollziehen.[30]
Für die Implementierung von Workflows in Cadence wird ein Client in Java und Go zur Verfügung gestellt. Weitere Clients in C# und Python sind in der Entwicklung.[92]

Entwicklungsfreiheit

Für die Entwicklung von Softwareprojekten mit Cadence wird keine Entwicklungsumgebung vorausgesetzt. Da Cadence auf reinem Java-Code basiert (siehe Abschnitt 5.5.3), kann ein beliebiger Editor zum Schreiben des Codes verwendet werden.

Anbindung an Webapplikation

Aus der Implementierung geht hervor, dass die Anbindung einer Webapplikation in Cadence nicht direkt möglich ist. Es existiert keine API, um auf Workflow-Instanzen oder deren Variablen zuzugreifen. Für das Auslesen dieser Daten bietet Cadence eigene Kommandozeilen Kommandos. Problematisch bei der Verwendung dieser Kommandos ist, dass diese immer die Kommandozeilenausgabe zurückliefern. Diese Ausgabe beinhaltet auch graphische Darstellungen wie Tabellen. So muss die Ausgabe in das gewünschte Format, beispielsweise JSON, konvertiert werden. Diese Konvertierung kann schnell zu Problemen führen, sollte sich das Format der Kommandozeilenausgabe ändern.

Typsicherheit der Workflow-Instanzvariablen

Cadence speichert Workflow-Instanzvariablen als Klassenvariablen der Workflow-Klasse. Diese können dann über Query-Methoden (siehe Listing 5.6) ausgelesen werden. Durch diesen Ansatz ist die Typsicherheit der Workflow-Instanzvariablen zu jeder Zeit gewährleistet, da den Variablen ein fester Typ zugewiesen wird.

6.5 Graphische Auswertung der Kriterien

Für die graphische Auswertung der Kriterien wurden die Abstufungen Kriterium erfüllt[✓], Kriterium teilweise erfüllt[(✓)] und Kriterium nicht erfüllt[✗] verwendet. Durch diese Veranschaulichung soll auf einen Blick ersichtlich sein, welche Kriterien von den Systemen wie gut umgesetzt werden. Dabei wurde die Einteilung in Entwicklungswerkzeuge und Systemeigenschaften beibehalten.

6.5.1 Entwicklungswerkzeuge

System / **Kriterium**	Camunda	Cadence	Zeebe	Bonita
Versionsverwaltung	✓	✓	✓	(✓)
Build-Management-Tools	✓	✓	✓	✗
Hohe Testabdeckung	(✓)	✓	✓	(✓)
Continuous Integration/ Continuous Delivery	✓	✓	✓	(✓)

TABELLE 6.1: Tabellarische Auswertung der Entwicklungswerkzeuge

Wie in Tabelle 6.1 ersichtlich, werden die Entwicklungswerkzeuge großteils unterstützt. Bonita bietet zwar die Möglichkeit, eine Versionskontrolle zu nutzen, da das Modell jedoch eine .proc-Datei ist und keine .bpmn-Datei können aufgrund von nicht standardisierten Tag-Namen Schwierigkeiten beim Zusammenführen auftreten. Bonita unterstützt keine Build-Management-Tools für das Bauen der Anwendung und das Verwalten von Abhängigkeiten im Projekt. In Camunda lässt sich der Code in Script-Tasks nicht ohne hohen Mehraufwand durch JUnit-Tests testen. Bonita bietet eine graphische Testabdeckung, welche jedoch nicht mit JUnit-Tests vergleichbar ist. Ein Ausführen von JUnit-Tests ist kompliziert, da das Bonita Studio Projekt exportiert und in ein Java-Projekt importiert werden muss. Continuous Integration und Continuous Delivery lassen sich bei Bonita nur durch ein eigenes Build-Skript verwenden. Da kein Standard angewandt wird, müssen sich Entwickler in die Struktur und den Aufbau dieses Skriptes einarbeiten.

6.5.2 Systemeigenschaften

System / **Kriterium**	Camunda	Cadence	Zeebe	Bonita
Kompatible Standards	✓	(✓)	✓	✓
Entwicklungsfreiheit	✓	✓	✓	✗
Anbindung an Webapplikation	✓	(✓)	(✓)	✓
Typsicherheit der Workflow-Instanzvariablen	✗	✓	✗	✓

TABELLE 6.2: Tabellarische Auswertung der Systemeigenschaften

Tabelle 6.2 zeigt, welche Systemeigenschaften die einzelnen Systeme besitzen. Cadence setzt bei der Implementierung auf durch Java-Klassen beschriebene Workflows und setzt daher keinen Standard für die Modellierung von Workflows ein. Bonita lässt sich mit der eigenen Entwicklungsumgebung Bonita Studio oder einem Java-Projekt mit komplexer Projektstruktur nutzen. Daher besteht bei Bonita keine Entscheidungsfreiheit seitens der Entwickler. Cadence und Zeebe bieten keine REST-Schnittstelle, um eine unkomplizierte Anbindung an eine Webapplikation zu ermöglichen. Die Typsicherheit von Workflow-Instanzvariablen können Camunda und Zeebe nicht gewährleisten, da beide den Variablen keine festen Typen zuweisen.

Kapitel 7

Fazit

Um sich dem Thema Workflow-Management anzunähern, wurden zu Beginn dieser Arbeit die grundlegenden Begriffe erklärt. Ebenfalls verdeutlicht wurden die Unterschiede von Geschäftsprozessen und Workflows. Durch eine umfassende Literaturrecherche wurden das Vorgehen bei der Digitalisierung von Geschäftsprozessen und die Notwendigkeit des Einsatzes gängiger Software-Engineering-Konzepte dargelegt. Um zu untersuchen, wie gut sich diese Konzepte mit Workflow-Management-Systemen integrieren lassen, wurde ein Geschäftsprozess in ausgewählten Systemen mit verschiedenen Ansätzen implementiert. Nach einer Auswahl für die Entwicklung wichtiger Software-Engineering-Konzepte wurden diese als Kriterien formuliert und anhand der Implementierung und einer umfassenden Recherche untersucht.
Die wichtigsten Erkenntnisse dieser Untersuchung waren, dass die Systeme eine grundsätzlich gute Integration der ausgewählten Konzepte wie Versionskontrolle, Testabdeckung, Einsatz von Build Management Tools und Nutzung von Continuous Integration und Continous Delivery, bieten. Einzig Camunda und Bonita setzen diese nicht vollends um. Bonita bietet zwar eine Versionskontrolle an, durch das eigene .proc-Dateiformat des modellierten Workflows wird hier jedoch kein Standard unterstützt und das Zusammenführen verschiedener Version wird durch nicht standardisierte Tag-Namen erschwert. Camunda bietet keine Testabdeckung für in den Workflow integrierte Script-Tasks. Bonita unterstützt den Einsatz von Build-Management-Tools wie Maven nicht. Außerdem funktioniert die Nutzung von Continuous Integration und Continuous Delivery nur durch den Einsatz eines von Bonita eigens entworfenen Build-Skriptes.
Große Unterschiede waren hingegen bei den Systemeigenschaften zu finden. So gibt es Systeme wie Bonita, welche bei der Digitalisierung von Geschäftsprozessen auf einen hohen graphischen Anteil im Entwurf des Prozessmodells und dem Einbinden von Logik durch eine eigene Entwicklungsumgebung setzen. Cadence hingegen setzt auf einen gegenteiligen Ansatz und ermöglicht die Definition des Geschäftsprozesses per Code. Daher werden in Cadence keine gängigen Standards zur Prozessmodellierung umgesetzt. Camunda und Zeebe setzen bei der Modellierung auf graphische Tools und ermöglichen das Hinzufügen von Logik zu den einzelnen Prozessschritten mittels Code.
Außerdem gab es Unterschiede in der Möglichkeit zur Anbindung eigener Webanwendungen. So bieten Camunda und Bonita umfangreiche REST-Endpunkte zur Steuerung und Verwaltung von Workflow-Instanzen an. Cadence und Zeebe hingegen bieten keinen direkten Zugriff auf die Workflow-Instanzen über eine REST-Schnittstelle. Dies wird nur durch Umwege möglich.
Ein weiteres Kriterium, welches nicht alle Systeme erfüllen konnten, war die Typsicherheit von Workflow-Instanzvariablen. Diese ist wichtig, da sich Entwickler bei

bestehender Typsicherheit sicher sein können, welchen Typ eine Variable zur Laufzeit hat und es nicht zu Problemen beim Auslesen oder Konvertieren der Variable kommt. Da bei Camunda und Zeebe die Variablen über deren Namen gesetzt und ausgelesen werden, ist zur Laufzeit keine Typsicherheit der Variablen gegeben.
Die Forschungsfrage dieser Arbeit „Inwieweit lassen sich klassische Software-Engineering-Konzepte auf die Entwicklung mit unterschiedlichen Workflow-Management-Systemen anwenden?“ lies sich durch die Auswertung der Kriterien beantworten. Diese bietet nun einen Anhaltspunkt über die Auswahl des richtigen Systems für ein anstehendes Projekt.
Zukünftig könnte die Nutzung von Workflow-Management-Systemen in der Cloud untersucht werden. Cloud Computing wird immer öfter genutzt, da bei Bedarf auf einfache Art und Weise hoch skaliert werden kann[93]. Es könnte untersucht werden, welche Systeme bereits jetzt dem Trend in Richtung Cloud-basierte Anwendungen folgen und welche Workflow-Management-Systeme bei klassischen Ansätzen bleiben. Ebenso kann untersucht werden, wie gut die Integration der Workflow-Management-Systeme in eine Cloud-basierte Umgebung funktioniert und wo dabei die Stärken und Schwächen liegen.

Anhang A

Anhang

A.1 Codelistings

Camunda JUnit-Test

```
1  public class ProcessUnitTest {
2    @Mock
3    Project project;
4    @Mock
5    Expert expert;
6
7    private RuntimeService runtimeService;
8    static {
9      LogFactory.useSlf4jLogging(); // MyBatis
10   }
11
12   @ClassRule
13   @Rule
14   public static ProcessEngineRule rule =
     ↪  TestCoverageProcessEngineRuleBuilder.create().build();
15
16   @Before
17   public void setup() {
18     init(rule.getProcessEngine());
19     runtimeService = rule.getRuntimeService();
20   }
21
22   @Test
23   @Deployment(resources = "process.bpmn")
24   public void testHappyPath() throws JsonProcessingException {
25     HashMap<String, Object> variables = new HashMap<String, Object>();
26     variables.put("title", "TestProjectTitle");
27     variables.put("description", "TestProjectDescription");
28     variables.put("skills", "TestProjectSkills");
29
30     ProcessInstance processInstance =
       ↪  processEngine().getRuntimeService()
31     .startProcessInstanceByKey(ProcessConstants.PROCESS_DEFINITION_KEY
       ↪  ,
       ↪  variables);
```

```
32
33      assertThat(processInstance).isActive();
34
35      assertThat(processInstance).hasPassed("saveProject").isWaitingAt("
        ↪ submitExperts");
36      ObjectMapper objectMapper = new ObjectMapper();
37      expert = new Expert();
38      expert.setProjectId(processInstance.getProcessInstanceId());
39      expert.setEmail("test@test.de");
40      expert.setFirstName("TestUser");
41      expert.setLastName("TestLastName");
42      expert.setStatement("TestStatement");
43      HashMap<String, Object> expertValues = new HashMap<String,
        ↪ Object>();
44      expertValues.put("expert",
        ↪ objectMapper.writeValueAsString(expert));
45      expertValues.put("projectProcessId",
        ↪ processInstance.getProcessInstanceId());
46
47      String messageName = "ApplicationReceived";
48
49      ProcessInstance checkApplicantFalse =
        ↪ runtimeService.createMessageCorrelation(messageName)
50      .setVariables(expertValues).correlateStartMessage();
51      assertThat(checkApplicantFalse).isStarted();
52      complete(task("checkApplicant"),
        ↪ withVariables("applicantAppropriate", false));
53
54      ProcessInstance checkApplicantTrue =
        ↪ runtimeService.createMessageCorrelation(messageName)
55      .setVariables(expertValues).correlateStartMessage();
56      assertThat(checkApplicantTrue).isStarted();
57      complete(task("checkApplicant"),
        ↪ withVariables("applicantAppropriate", true));
58
59      ProcessInstance checkApplicantForList =
        ↪ runtimeService.createMessageCorrelation(messageName)
60      .setVariables(expertValues).correlateStartMessage();
61      assertThat(checkApplicantForList).isStarted();
62      complete(task("checkApplicant"),
        ↪ withVariables("applicantAppropriate", true));
63
64      assertThat(processInstance).isWaitingAt("submitExperts");
65      complete(task("submitExperts"));
66      complete(task("chooseExpert"), withVariables("selectedExpert",
        ↪ expertValues.get("expert")));
67    }
68  }
```

LISTING A.1: Camunda Test für die Implementierung des Beispiel-workflows

Zeebe JUnit-Test

```
public class ProcessUnitTest {
  final ZeebeClient client = ZeebeClient.newClientBuilder()
  .gatewayAddress("127.0.0.1:26500").usePlaintext().build();

  @Mock
  JobClient jc;
  @Mock
  ActivatedJob job;

  @Test
  void shouldConnectToZeebe() throws Exception {
    jc = Mockito.mock(JobClient.class);

    App.deployProcess();
    final JobWorker checkApplicant =
     ↪  client.newWorker().jobType("user").handler(new JobHandler() {

      @Override
      public void handle(JobClient client,
       ↪  io.zeebe.client.api.response.ActivatedJob job) throws
       ↪  Exception {
        if (job.getElementId().equals("checkApplicant")) {
          String expert = "{\"applicantAppropriate\":true,
           ↪  \"expert\":{"
              + "\"id\": \"this.currentTaskId\",\r\n" + "
               ↪  \"email\": \"this.formdata.email\",\r\n"
              + "        \"firstName\":
               ↪  \"this.formdata.firstName\",\r\n"
              + "        \"lastName\": \"this.formdata.lastName\"}}";
          client.newCompleteCommand(job.getKey()).variables(expert).se
           ↪  nd();

        }
      }
    }).open();

    HashMap<String, Object> project = new HashMap<String, Object>();
    project.put("title", "TestProjectTitle");
    project.put("description", "TestProjectDescription");
    project.put("skills", "TestProjectSkills");

    WorkflowInstanceEvent workflowInstance =
     ↪  client.newCreateInstanceCommand().bpmnProcessId("process")
    .latestVersion().variables(project).send().join();
    Thread.sleep(1000);
```

```
38      HashMap<String, Object> variables = new HashMap<String, Object>();
39      variables.put("id", "12345");
40      variables.put("firstName", "UnitTestFirstName");
41      variables.put("lastName", "UnitTestLastName");
42      variables.put("email", "Unit@test.de");
43      variables.put("statement", "UnitTestStatement");
44      variables.put("processId",
        ↪  workflowInstance.getWorkflowInstanceKey());
45      variables.put("checkExpert", true);
46
47      PublishMessageResponse applicantWorkflow = client.newPublishMessag
        ↪  eCommand().messageName("applicationReceived")
48      .correlationKey("=applicant" +
        ↪  workflowInstance.getWorkflowInstanceKey()).variables(variables)
49      .timeToLive(Duration.ofMinutes(10)).send().join();
50      Thread.sleep(5000);
51    }
52  }
```

LISTING A.2: Zeebe Test für die Implementierung des Beispielworkflows

Cadence JUnit-Test

```
1   public class ProcessUnitTest {
2
3     @Rule
4     public TestWatcher watchman =
5     new TestWatcher() {
6       @Override
7       protected void failed(Throwable e, Description description) {
8         if (testEnv != null) {
9           System.err.println(testEnv.getDiagnostics());
10          testEnv.close();
11        }
12      }
13    };
14
15    private TestWorkflowEnvironment testEnv;
16    private Worker worker;
17    private WorkflowClient workflowClient;
18
19    Project pro = new Project();
20    Expert expertOne = new Expert();
21
22    @Before
23    public void setUp() {
24      testEnv = TestWorkflowEnvironment.newInstance();
25      pro.setTitle("TestPro");
26      pro.setDescription("TestProDesc");
```

```
27      pro.setSkills("TestProSkills");
28      expertOne.setFirstName("FirstNameUnitTest");
29      expertOne.setLastName("LastNameUnitTest");
30      expertOne.setEmail("unit@test.de");
31
32      expertOne.setStatement("UnitTestStatement");
33
34
35      worker = testEnv.newWorker("ThesisTasklist");
36
37
38      workflowClient = testEnv.newWorkflowClient();
39    }
40
41    @After
42    public void tearDown() {
43      testEnv.close();
44    }
45
46    @Test
47    public void testSignal() throws JsonMappingException,
      ↪  JsonProcessingException, InterruptedException {
48      worker.registerWorkflowImplementationTypes(ProjectWorkflowImpl.cla
        ↪  ss,
        ↪  CheckApplicantWorkflowImpl.class);
49      testEnv.start();
50      WorkflowOptions workflowOptions =
51      new WorkflowOptions.Builder()
52      .setTaskList("ThesisTasklist")
53      .setExecutionStartToCloseTimeout(Duration.ofDays(30))
54      .build();
55      ProjectWorkflow workflow =
56      workflowClient.newWorkflowStub(ProjectWorkflow.class,
        ↪  workflowOptions);
57      String projectValues = "{\"title\":\"UnitTest\",
        ↪  \"description\":\"UnitTestDescription\",
        ↪  \"skills\":\"UnitTestSkills\"}";
58      WorkflowClient.start(workflow::execute, projectValues);
59      Thread.sleep(1000);
60      workflow.userTaskFinished();
61      ProcessVariables vars = workflow.getVariables();
62      expertOne.setProjectId(vars.project.getId()+"");
63
64      workflow.newApplicant(expertOne);
65      Thread.sleep(1000);
66      workflow.selectExpert(expertOne);
67      Thread.sleep(1000);
68      workflow.userTaskFinished();
69      System.out.println("Test finished");
70      testEnv.close();
71
```

```
72    }
73 }
```

LISTING A.3: Cadence Test für die Implementierung des Beispiel-workflows

Literatur

[1] Torsten Horn. *Annotations (ab Java 5)*. 30.07.2018. URL: https://www.torsten-horn.de/techdocs/java-annotations.htm (besucht am 16. 03. 2021).

[2] Martin H. Weik. „application program interface“. In: *Computer science and communications dictionary*. Hrsg. von Martin H. Weik. Boston, Mass.: Kluwer, 2000, S. 59. ISBN: 978-0-7923-8425-0. DOI: 10.1007/1-4020-0613-6_769.

[3] Roland Schmitz und Walter Kriha. „Application Server Security“. In: *Sichere Systeme*. Hrsg. von Walter Kriha und Roland Schmitz. Xpert.press. Berlin: Springer, 2009, S. 313–362. ISBN: 978-3-540-78958-1. DOI: 10.1007/978-3-540-78959-8_7.

[4] Balaji Varanasi. „Getting Started with Maven“. In: *Introducing Maven*. Hrsg. von Balaji Varanasi. Berkeley, CA: Apress, 2019, S. 1–10. ISBN: 978-1-4842-5409-7. DOI: 10.1007/978-1-4842-5410-3_1.

[5] Jürgen Halstenberg, Bernd Pfitzinger und Thomas Jestädt. „DevOps-Aktivitäten“. In: *DEVOPS*. Hrsg. von JRGEN HALSTENBERG. essentials. [S.l.]: MORGAN KAUFMANN, 2020, S. 15–26. ISBN: 978-3-658-31404-0. DOI: 10.1007/978-3-658-31405-7_3.

[6] Moshe Zadka. „Docker“. In: *DevOps in Python: Infrastructure As Python*. Hrsg. von Moshe Zadka. Berkeley, CA: Apress, 2019, S. 147–150. ISBN: 978-1-4842-4432-6. DOI: 10.1007/978-1-4842-4433-3_12.

[7] Linus Nyman und Tommi Mikkonen. „To Fork or Not to Fork: Fork Motivations in SourceForge Projects“. In: *Open source systems: grounding research*. Hrsg. von Scott Hissam. Bd. 365. IFIP Advances in Information and Communication Technology. Heidelberg: Springer, 2011, S. 259–268. ISBN: 978-3-642-24417-9. DOI: 10.1007/978-3-642-24418-6_18.

[8] Renan Vieira Antunes, Gabriela Matias Navarro und Simone Hanazumi. „Test Framework for Jenkins Shared Libraries“. In: *Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing*. [Place of publication not identified]: ACM, 2018, S. 13–19. ISBN: 9781450365550. DOI: 10.1145/3266003.3266008.

[9] Jeff Friesen. *Java XML and JSON: Document Processing for Java SE*. 2nd ed. Berkeley, CA: Apress, 2019. ISBN: 978-1-4842-4329-9. DOI: 10.1007/978-1-4842-4330-5.

[10] Adalberto R. Sampaio u. a. „Improving microservice-based applications with runtime placement adaptation“. In: *Journal of Internet Services and Applications* 10.1 (2019). ISSN: 1867-4828. DOI: 10.1186/s13174-019-0104-0.

[11] Russ Ferguson. „JavaScript and Development Tools“. In: *Beginning JavaScript*. Hrsg. von Ferguson. Berkeley, CA: Apress, 2019, S. 11–24. ISBN: 978-1-4842-4394-7. DOI: 10.1007/978-1-4842-4395-4_2.

[12] Mark Massé. *REST API design rulebook: Designing Consistent RESTful Web Service Interfaces*. Sebastopol, CA: O'Reilly, 2012. ISBN: 9781449319908.

[13] Sufyan bin Uzayr, Nicholas Cloud und Tim Ambler. „Vue.js“. In: *JavaScript Frameworks for Modern Web Development*. Hrsg. von Sufyan bin Uzayr, Nicholas Cloud und Tim Ambler. Berkeley CA: Apress und Imprint: Apress, 2019, S. 523–539. ISBN: 978-1-4842-4995-6. DOI: 10.1007/978-1-4842-4995-6_14.

[14] „Glossar zu Prozessmanagement“. In: *HMD Praxis der Wirtschaftsinformatik* 46.2 (2009), S. 117–118. ISSN: 2198-2775. DOI: 10.1007/BF03340350.

[15] Thomas Schäl. *Workflow management systems for process organisations: Zugl.: Aachen, Techn. Hochsch., Diss., 1995*. Bd. 1096. Lecture notes in computer science. Berlin: Springer, 1996. ISBN: 9783662215746. DOI: 10.1007/BFb0013558. URL: http://www.springerlink.com/content/k2kmt33j21kw.

[16] Andreas Gadatsch. „IT-Unterstützung für das Prozessmanagement“. In: *Grundkurs Geschäftsprozess-Management*. Hrsg. von Andreas Gadatsch. Lehrbuch. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 153–199. ISBN: 978-3-658-27811-3. DOI: 10.1007/978-3-658-27812-0_6.

[17] C. Nedeß, A. Friedewald und N. Davids. „Prozessunterstützung in Produktentwicklungsprojekten“. In: *Design for X*. Hrsg. von Harald Meerkamm. Erlangen: Univ. Erlangen-Nürnberg Lehrstuhl Konstruktionstechnik, 2006, S. 33–42. ISBN: 3980853942.

[18] Hartmut Ernst, Jochen Schmidt und Gerd Beneken. „Software-Engineering“. In: *Grundkurs Informatik*. Hrsg. von Hartmut Ernst, Jochen Schmidt und Gerd Hinrich Beneken. Lehrbuch. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 691–727. ISBN: 978-3-658-30330-3. DOI: 10.1007/978-3-658-30331-0_15.

[19] Jeff Dalton. „Continuous Integration“. In: *Great Big Agile*. Hrsg. von Jeff Dalton. Berkeley, CA: Apress, 2019, S. 151–152. ISBN: 978-1-4842-4205-6. DOI: 10.1007/978-1-4842-4206-3_22.

[20] Jorge Acetozi. „Continuous Delivery“. In: *Pro Java clustering and scalability*. Hrsg. von Jorge Acetozi. Berkeley, CA: Apress, 2017, S. 117. ISBN: 978-1-4842-2984-2. DOI: 10.1007/978-1-4842-2985-9_22.

[21] Holger Arndt. „Grundlagen der Prozessoptimierung“. In: *Logistikmanagement*. Hrsg. von Holger Arndt. Springer-Gabler Lehrbuch. Wiesbaden: Springer-Gabler, 2015, S. 35–42. ISBN: 978-3-658-07212-4. DOI: 10.1007/978-3-658-07212-4_3.

[22] Wil van der Aalst und Kees Max van Hee. *Workflow management: Models, methods and systems*. 1. MIT Press paperback ed. Cooperative information systems. Cambridge, Mass.: MIT Press, 2004. ISBN: 0262720469.

[23] „Grundlegende Begriffe“. In: *Grundkurs Geschäftsprozess-Management*. Hrsg. von Andreas Gadatsch. OnlinePlus. Wiesbaden: Vieweg, 2008, S. 1–71. ISBN: 978-3-8348-0363-4. DOI: 10.1007/978-3-8348-9422-9_1.

[24] Jörg Becker und Dieter Kahn. „Der Prozess im Fokus“. In: *Prozessmanagement*. Hrsg. von Jörg Becker, Martin Kugeler und Michael Rosemann. Berlin und Heidelberg: Springer, 2012, S. 3–16. ISBN: 978-3-642-33843-4. DOI: 10.1007/978-3-642-33844-1_1.

[25] *Workflow vs. Process*. URL: https://www.microtech.de/wp-content/uploads/2020/08/workflow-management-microtech.png (besucht am 10. 10. 2020).

[26] Michael zur Mühlen und Holger Hansmann. „Workflowmanagement“. In: *Prozessmanagement*. Hrsg. von Jörg Becker, Martin Kugeler und Michael Rosemann. Berlin und Heidelberg: Springer, 2012, S. 367–400. ISBN: 978-3-642-33843-4. DOI: 10.1007/978-3-642-33844-1_11.

[27] Christian Hastedt-Marckwardt. „Workflow-Management-Systeme“. In: *Informatik-Spektrum* 22.2 (1999), S. 99–109. ISSN: 0170-6012. DOI: 10.1007/s002870050129.

[28] Peter Dadam, Manfred Reichert und Stefanie Rinderle-Ma. „Prozessmanagementsysteme“. In: *Informatik-Spektrum* 34.4 (2011), S. 364–376. ISSN: 0170-6012. DOI: 10.1007/s00287-010-0456-0.

[29] Christian Matt. „Workflow-Management-Systeme“. In: *Controlling & Management* 50.4 (2006), S. 199–200. ISSN: 1864-5410. DOI: 10.1365/s12176-006-0538-7.

[30] Michele Chinosi und Alberto Trombetta. „BPMN: An introduction to the standard“. In: *Computer Standards & Interfaces* 34.1 (2012), S. 124–134. ISSN: 09205489. DOI: 10.1016/j.csi.2011.06.002.

[31] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Berlin Heidelberg New York: Springer-Verlag, 2010. ISBN: 978-3-827-42247-7.

[32] Bernward Mütterlein. „Software-Entwicklung“. In: *Handbuch für die Pogrammierung mit LabVIEW*. Hrsg. von Bernward Mütterlein. Heidelberg: Spektrum Akademischer Verlag, 2007, S. 45–101. ISBN: 978-3-8274-2337-5. DOI: 10.1007/978-3-8274-2338-2_4.

[33] Juan F. Perez, Weikun Wang und Giuliano Casale. „Towards a DevOps Approach for Software Quality Engineering“. In: *ICPE '15*. Hrsg. von Murray Woodside. New York, NY: ACM, 2015, S. 5–10. ISBN: 9781450333405. DOI: 10.1145/2693561.2693564.

[34] Pierre Bourque, Hrsg. *Guide to the software engineering body of knowledge: Version 3.0 ; SWEBOK ; a project of the IEEE Computer Society*. Los Alamitos, Calif. [u.a.]: IEEE Computer Soc, 2014. ISBN: 978-0-7695-5166-1.

[35] Meir Wahnon. *meirwah/awesome-workflow-engines*. 30.10.2020. URL: https://github.com/meirwah/awesome-workflow-engines (besucht am 30. 10. 2020).

[36] GitHub. *apache/airflow*. 17.03.2021. URL: https://github.com/apache/airflow (besucht am 17. 03. 2021).

[37] Docs. *Overview | n8n Docs*. 24.08.2020. URL: https://docs.n8n.io/ (besucht am 17. 03. 2021).

[38] GitHub. *argoproj/argo-workflows*. 17.03.2021. URL: https://github.com/argoproj/argo-workflows/blob/master/examples/README.md#hello-world (besucht am 17. 03. 2021).

[39] GitHub. *PrefectHQ/prefect*. 17.03.2021. URL: https://github.com/PrefectHQ/prefect (besucht am 17. 03. 2021).

[40] *Client Libraries and Language Bindings — StackStorm 3.4.0 documentation*. 3.03.2021. URL: https://docs.stackstorm.com/reference/client_libraries.html (besucht am 17. 03. 2021).

[41] *Job Workflows*. 5.03.2021. URL: https://docs.rundeck.com/docs/manual/job-workflows.html#workflow-definition (besucht am 17. 03. 2021).

[42] GitHub. *j-easy/easy-rules*. 17.03.2021. URL: https://github.com/j-easy/easy-rules (besucht am 17. 03. 2021).

[43] *Brigade | Event-driven scripting for Kubernetes*. 17.03.2021. URL: https://brigade.sh/ (besucht am 17. 03. 2021).

[44] GitHub. *elsa-workflows/elsa-core*. 17.03.2021. URL: https://github.com/elsa-workflows/elsa-core (besucht am 17. 03. 2021).

[45] Alexandra Kees und Dominic Raimon Markowski. „Marktübersicht OS BPM- und Workflowmanagement-Software“. In: *Open Source Enterprise Software*. Hrsg. von Alexandra Kees und Dominic Raimon Markowski. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 267–332. ISBN: 978-3-658-25217-5. DOI: 10.1007/978-3-658-25218-2_8.

[46] Adriana Caione u. a. „Knowledge base support for dynamic information system management“. In: *Information Systems and e-Business Management* 14.3 (2016), S. 533–576. ISSN: 1617-9846. DOI: 10.1007/s10257-015-0294-3.

[47] *Designer.png*. URL: https://docs.jboss.org/jbpm/release/latest/jbpm-docs/html_single/Overview/Designer.png (besucht am 16. 03. 2021).

[48] Bilgin Avenoğlu und P. Erhan Eren. „A context-aware and workflow-based framework for pervasive environments“. In: *Journal of Ambient Intelligence and Humanized Computing* 10.1 (2019), S. 215–237. ISSN: 1868-5137. DOI: 10.1007/s12652-017-0633-y.

[49] *ProcessInstanceDiagram.png*. URL: https://docs.jboss.org/jbpm/release/latest/jbpm-docs/html_single/Overview/ProcessInstanceDiagram.png (besucht am 16. 03. 2021).

[50] *Open-source Workflow Evaluation: An evaluation of the Activiti BPM Platform*. 2012. URL: https://www.diva-portal.org/smash/get/diva2:555648/fulltext01.pdf.

[51] Zakir Laliwala und Rehankhan Pathan. *Activiti BPM Beginner's Guide*. Birmingham: Packt Publishing, 2014. ISBN: 978-1-84951-706-5. URL: http://gbv.eblib.com/patron/FullRecord.aspx?p=1477472.

[52] *Activiti Model*. URL: https://www.activiti.org/userguide/images/designer.model.process.png (besucht am 16. 03. 2021).

[53] *Activiti Instances*. URL: https://training-course-material.com/images/2/21/ClipCapIt-150728-095648.PNG (besucht am 16. 03. 2021).

[54] Andrea Delgado und Daniel Calegari. „A generic BPMS user portal for business processes execution interoperability“. In: *2019 XLV Latin American Computing Conference*. [Piscataway, NJ]: IEEE, 2019. ISBN: 9781728155746. DOI: 10.1109/clei47609.2019.235117.

[55] Clara Gaset. „BPM implementation for the worldwide distribution of Azithromycin for Yaws eradication“. Diss. Universitat Politècnica de Catalunya und Universitat Politècnica de Catalunya, 2018. URL: https://upcommons.upc.edu/handle/2117/121976.

[56] *flowable_modeler_design_screen.png*. URL: https://flowable.com/open-source/docs/assets/bpmn/flowable_modeler_design_screen.png (besucht am 16. 03. 2021).

[57] *Flowable applications · Flowable Open Source Documentation*. 2020. URL: https://flowable.com/open-source/docs/bpmn/ch14-Applications (besucht am 15. 10. 2020).

[58] *flowable Instances*. URL: https://flowable.com/open-source/docs/assets/bpmn/flowable_task_processhistory_screen.png (besucht am 16. 03. 2021).

[59] Camunda. *Camunda Engine Evolution since Activiti Fork - Camunda*. 2016. URL: https://camunda.com/blog/2016/10/camunda-engine-since-activiti-fork/ (besucht am 17. 03. 2021).

[60] *Camunda Model*. URL: https://camunda.com/wp-content/uploads/2020/05/quickstart-3.png (besucht am 16. 03. 2021).

[61] Mahdi Saeedi Nikoo, Önder Babur und Mark van den Brand. „A Survey on Service Composition Languages“. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '20. Virtual Event, Canada: Association for Computing Machinery, 2020. ISBN: 9781450381352. DOI: 10.1145/3417990.3421402. URL: https://doi.org/10.1145/3417990.3421402.

[62] *Camunda Instances*. URL: https://camunda.com/wp-content/uploads/2020/05/cockpit-process-definitions-view.png (besucht am 16. 03. 2021).

[63] Pedram Hamidehkhan. *Analysis and evaluation of composition languages and orchestration engines for microservices*. 2019. URL: http://elib.uni-stuttgart.de/handle/11682/10336.

[64] Zeebe. *Zeebe License Overview and FAQ*. 12.01.2021. URL: https://zeebe.io/zeebe-license-overview/ (besucht am 29. 01. 2021).

[65] *Zeebe Model*. URL: https://camunda.com/wp-content/uploads/camunda/zeebe-images/blog/message-correlation/zeebe-modeler-annotated.png (besucht am 16. 03. 2021).

[66] *Zeebe Instances*. URL: https://camunda.com/wp-content/uploads/2020/05/operate-process-screenshot.png (besucht am 16. 03. 2021).

[67] Mickey Farrance Nicolas Chabanoles Philippe Ozil. „Bonita BPM: an open–source BPM–based application development platform to build adaptable business applications“. In: *Proceedings of the BPM Demo Session 2015*. Bd. 1418, S. 21–24. URL: http://ceur-ws.org/Vol-1418/paper5.pdf (besucht am 28. 10. 2020).

[68] *Bonita Model*. URL: https://windows-cdn.softpedia.com/screenshots/Bonita-Open-Solution_24.png (besucht am 16. 03. 2021).

[69] *Bonita Instances*. URL: https://documentation.bonitasoft.com/bonita/2021.1/_images/images/UI2021.1/admin-case-list.png (besucht am 16. 03. 2021).

[70] Jawad Ahmad. „Cadence — The only workflow orchestrator you will ever need“. In: *Noteworthy - The Journal Blog* (23.02.2020). URL: https://blog.usejournal.com/cadence-the-only-workflow-orchestrator-you-will-ever-need-ea8f74ed5563 (besucht am 12. 11. 2020).

[71] *Cadence Instances*. URL: https://camo.githubusercontent.com/ee483f1afe9e30b67885ec888df79c0b0a6c27f6ad3d0f1545fc6c3d2f479113/68747470733a2f2f73332d75732d776573742d322e616d617a6f6e6177732e636f6d2f756265722d636f6
(besucht am 16. 03. 2021).

[72] uber. *cadence-web*. 2020. URL: https://github.com/uber/cadence-web (besucht am 15. 10. 2020).

[73] Dirk W. Hoffmann. „Software-Test“. In: *Software-Qualität*. Hrsg. von Dirk W. Hoffmann. eXamen.press. Berlin: Springer Vieweg, 2013, S. 157–246. ISBN: 978-3-642-35700-8. DOI: 10.1007/978-3-642-35700-8{\textunderscore}4.

[74] Daniel Wasserab u. a. „C++ ist typsicher? Garantiert!“ In: *1617-5468* (2007). ISSN: 1617-5468. URL: https://dl.gi.de/handle/20.500.12116/22771.

[75] GitHub. *ZeeQS*. 22.01.2021. URL: https://github.com/zeebe-io/zeeqs (besucht am 22. 01. 2021).

[76] GitHub. *zeebe-io/zeebe-simple-tasklist*. 25.02.2021. URL: https://github.com/zeebe-io/zeebe-simple-tasklist (besucht am 25. 02. 2021).

[77] GitHub. *Node.js client library for Zeebe Microservices Orchestration Engine*. 22.01.2021. URL: https://github.com/zeebe-io/zeebe-client-node-js (besucht am 22. 01. 2021).

[78] GitHub. *Zeebe-Hazelcast-Exporter*. 22.01.2021. URL: https://github.com/zeebe-io/zeebe-hazelcast-exporter (besucht am 22. 01. 2021).

[79] Allan Fernandez. „Camunda BPM platform loan assessment process lab“. In: *Brisbane, Australia: Queensland University of Technology* (2013).

[80] Heiko Kalista. „Versionsverwaltung mit Git“. In: *Python 3*. Hrsg. von Heiko Kalista. München: Hanser, 2018, S. 403–446. ISBN: 9783446454699. DOI: 10.3139/9783446456891.012.

[81] *Camunda Best Practices*. 1.10.2020. URL: https://camunda.com/best-practices/_book/ (besucht am 23. 02. 2021).

[82] *Maven Repository: io.zeebe » zeebe-client-java*. 23.02.2021. URL: https://mvnrepository.com/artifact/io.zeebe/zeebe-client-java (besucht am 23. 02. 2021).

[83] *Jenkins and Java*. 12.03.2021. URL: https://www.jenkins.io/solutions/java/ (besucht am 16. 03. 2021).

[84] Josh Wulf. *Writing a Zeebe Client in 2020*. 18.06.2020. URL: https://zeebe.io/blog/2020/06/zeebe-client-2020/ (besucht am 16. 02. 2021).

[85] *Bonita 7.10 - How to use Git for versioning with Community Edition*. 28.01.2021. URL: https://documentation.bonitasoft.com/bonita/7.12/git-versioning-community-edition (besucht am 23. 02. 2021).

[86] *Bonita 2021.1 - Continuous integration*. 28.01.2021. URL: https://documentation.bonitasoft.com/bonita/7.12/_continuous-integration (besucht am 23. 02. 2021).

[87] *Set up continuous integration | Bonita Documentation*. 17.03.2021. URL: https://documentation.bonitasoft.com/bonita/2021.1/set-up-continuous-integration (besucht am 17. 03. 2021).

[88] *Bonita for DevOps and Continuous Delivery*. 17.03.2021. URL: https://www.bonitasoft.com/bonita-continuous-delivery (besucht am 17. 03. 2021).

[89] *Bonita 2021.1 - Bonita connector archetype*. 28.01.2021. URL: https://documentation.bonitasoft.com/bonita/7.12/connector-archetype (besucht am 23. 02. 2021).

[90] *Maven Repository: com.uber.cadence » cadence-client*. 23.02.2021. URL: https://mvnrepository.com/artifact/com.uber.cadence/cadence-client (besucht am 23. 02. 2021).

[91] John Ferguson Smart. *Jenkins: Continuous Integration for the Masses*. Sebastopol: O'Reilly Media, 2011. ISBN: 9781449313654.

[92] *Deployment topology | Cadence*. 5.02.2021. URL: https://cadenceworkflow.io/docs/concepts/topology/#workflow-worker (besucht am 23. 02. 2021).

[93] Yong Zhao u. a. „A Service Framework for Scientific Workflow Management in the Cloud". In: *IEEE Transactions on Services Computing* 8.6 (2015), S. 930–944. DOI: 10.1109/TSC.2014.2341235.