Mining von BPMN-Prozessartefakten auf GitHub

Thomas S. Heinze[1], Viktor Stefanko[2] und Wolfram Amme[2]

[1] Institute of Data Science
German Aerospace Center (DLR)
thomas.heinze@dlr.de
[2] Institute of Computer Science
Friedrich Schiller University Jena
[wolfram.amme,viktor.stefanko]@uni-jena.de

Zusammenfassung. Die Validierung von Programmanalysen und Analysewerkzeugen erfolgt häufig anhand einer Sammlung von Beispielprogrammen. Für eine aussagekräftige Bewertung sollte diese Sammlung repräsentativ sein. Gerade im Bereich der Analysen für domänenspezifische Sprachen wird zu diesem Zweck aber oft nur auf Fallstudien oder gar einzelne künstliche Programmbeispiele zurückgegriffen. In diesem Beitrag zeigen wir für die domänenspezifische Sprache BPMN, wie durch systematisches Mining auf `GitHub.com` ein Corpus von BPMN-Prozessen für die Validierung von Analysewerkzeugen generiert werden kann.

1 Einführung

Ansätze zum Repository-Mining, das heißt zur systematischen Gewinnung, Aufbereitung und Analyse von Daten aus Code-Repositorys, erfreuen sich mit der wachsenden Bedeutung von Plattformen wie etwa `GitHub.com`, `GitLab.com`, `Bitbucket.org` und `SourceForge.net` und den damit zur Verfügung stehenden Datenquellen steigender Beliebtheit. So lassen sich vielfältige Fragestellungen zu Softwareentwicklungsprozessen und -produkten auf Grundlage der in Code-Repositorys enthaltenen Informationen untersuchen und Hypothesen zur Softwareentwicklung empirisch testen. Steht in den meisten Arbeiten noch klassischer Quellcode im Vordergrund, wird das zugrundeliegende Prinzip immer mehr auch auf andere Produkte der Softwareentwicklung übertragen. Als exemplarisches Beispiel sei auf das Mining vom UML-Modellen auf `GitHub.com` verwiesen [6,10], welches auch die Inspiration für den in diesem Beitrag vorgestellten Ansatz gab.

Gerade im Bereich der domänenspezifischen Sprachen könnten, wie schon in [10] angedeutet, Ansätze zum Repository-Mining die empirische Forschung unterstützten. So fehlen oft umfassende und insbesondere repräsentative Sammlungen von Programmbeispielen zur Untersuchung von Fragen zur praktischen Sprachverwendung oder zur Sprachunterstützung durch Modellierungs- und Analysewerkzeuge. Zudem stehen oft Anforderungen an den Schutz geistigen Eigentums der Verwendung von realen Programmbeispielen aus einem industriellen oder wirtschaftlichen Kontext gegenüber. Stattdessen wird beispielsweise in

Forschungsarbeiten zu Modellierungs- und Analysewerkzeugen zu Validierungszwecken immer wieder auf künstliche Programmbeispiele oder einzelne Fallstudien zurückgegriffen [10], die natürlich nur eine eingeschränkte Aussagekraft besitzen. In diesem Beitrag greifen wir daher den Ansatz aus [6,10] auf und wenden ihn auf die domänenspezifische Sprache zur Geschäftsprozessmodellierung BPMN an. Dazu generieren wir durch systematisches Mining auf `GitHub.com` einen Corpus von BPMN-Prozessen, den wir anschließend zur Validierung von Aussagen zur praktischen Relevanz eines Analysewerkzeugs für BPMN nutzen.

Business Process Model and Notation (BPMN). Die Sprache BPMN definiert ein standardisiertes Format zur IT-Unterstützung von Geschäftsprozessen. Dabei spannt BPMN den Bogen von Domänenexperten, die sich für die Prozessmodellierung verantwortlich zeichnen, zur IT, die Prozesse bis hin zu deren vollautomatischen Ausführung implementiert. Der Sprachstandard in [1] spezifiziert zu diesem Zweck neben einer grafischen Notation zur Prozessmodellierung auch ein XML-Dialekt zur Serialisierung und eine Sprachsemantik. Neben einer Reihe von Entwicklungsumgebungen zur Modellierung stehen verschiedene Ausführungsumgebungen, wie etwa *Activiti*[3] oder *Camunda*[4], zur Verfügung. BPMN ist eine komplexe Sprache und die Prozessmodellierung ist aufgrund des Sprachentwurfs zum Teil recht fehleranfällig. Dementsprechend wichtig sind Analysewerkzeuge zur Unterstützung der Prozessmodellierung. Primär im Forschungsumfeld existieren mehrere Arbeiten zur Prozessanalyse. Die vorgeschlagenen Ansätze decken das Spektrum, beginnend bei *Linter*-Werkzeugen zur Aufdeckung von Verstößen gegen einfache Syntax- und Semantikregeln aus dem Sprachstandard, bis hin zu Techniken und Werkzeugen zur statischen Analyse und formalen Verifikation, wie in [7,8], ab. In diesem Beitrag betrachten wir beispielhaft das Analysewerkzeug *BPMNspector*[5] [4] aus der ersten Kategorie.

Die Schwierigkeiten der Generierung einer repräsentativen Sammlung von Prozessbeispielen für Forschungszwecke sind in der Fachdomäne der Geschäftsprozesse allgemein bekannt und nicht zuletzt durch deren Verwendung begründet: Der mit BPMN modellierte Prozess ist oft das Produkt und unterliegt einer entsprechenden Geheimhaltung [11]. In der Folge überwiegen auch die oben bereits erwähnten Fallstudien und künstlichen Beispiele bei der Validierung von Analysewerkzeugen und -techniken für BPMN [2,4,8,7,9]. Auf das den Autoren einzige bekannte Projekt *BenchFlow*[6] zu einer umfassenden Prozesssammlung kann nicht öffentlich zugegriffen werden (insgesamt 8.363 Modelle, Anteil BPMN: 64% [11]).

Der Beitrag gliedert sich wie folgt: In Abschnitt 2 wird der Mining-Prozess vorgestellt, mit dessen Hilfe Prozessartefakte in Repositorys auf `GitHub.com` gesucht und ein Corpus von BPMN-Prozessen generiert werden konnte. Die Ergebnisse des Mining-Prozesses und der nachfolgenden Analyse auf dem generierten Corpus bildet den Gegenstand von Abschnitt 3. Dabei wird sowohl auf

[3] `https://www.activiti.org` (21.8.2019)
[4] `https://camunda.com` (21.8.2019)
[5] `https://github.com/uniba-dsg/BPMNspector` (21.8.2019)
[6] `http://design.inf.usi.ch/research/projects/benchflow` (21.8.2019)

Charakteristika der identifizierten Prozessartefakte als auch auf die Resultate der Anwendung des Analysewerkzeugs *BPMNspector* eingegangen. Der Beitrag schließt mit einer Zusammenfassung und einer Diskussion zu Einschränkungen für die Interpretation der Ergebnisse des Beitrags in Abschnitt 4.

2 Mining-Prozess und Analyse

Das systematische Repository-Mining ermöglicht die Analyse der in heutigen Code-Repositorys vielfältig vorliegenden Daten zu Softwareentwicklungsprozessen und -produkten. Der Mining-Prozess gliedert sich dabei in mehrere Schritte: Zieldefinition, Datenauswahl, Datengewinnung und -extraktion, Datenvorverarbeitung und -bereinigung, sowie Datenanalyse und schließlich Interpretation der gewonnenen Ergebnisse. Als wichtige Quellen für die Datengewinnung dienen insbesondere öffentliche Repositorys auf Plattformen wie `GitLab.com`, `Bitbucket.org`, `SourceForge.net` oder `GitHub.com`, wobei letztere die wohl führende Plattform für Open-Source-Software ist. Zum Mining auf `GitHub.com` lassen sich grundsätzlich verschiedene Ansätze nutzen:

GitHub API. Einerseits wird durch `GitHub.com` die REST-Schnittstelle *GitHub API*[7] bereitgestellt. Über die Schnittstelle lassen sich verschiedene Anfragen zu Code-Repositorys stellen, so etwa zur Repository-Struktur, aber auch zu Metadaten wie beispielsweise den Autoren oder der dominanten Programmiersprache eines Repositorys. Jedoch ist die Anzahl von möglichen Anfragen limitiert, so sind bis zu 60 Anfragen pro Stunde ohne Authentifizierung und bis zu 5.000 Anfragen pro Stunde mit Authentifizierung erlaubt. Für das systematische und effektive Mining ist nur die letzte Variante sinnvoll.

GHTorrent. Alternativ stellt das Projekt *GHTorrent*[8] eine periodisch gespiegelte Datenbank zu allen Code-Repositorys auf `GitHub.com` zur Verfügung [5]. An diese lassen sich Anfragen entweder direkt über eine Webanwendung stellen oder die Datenbanktabellen werden in eine lokale Datenbank importiert und anschließend ausgewertet. Im Fokus des Projekts stehen jedoch Metadaten, so dass etwa die Repository-Struktur nicht in der Datenbank enthalten ist.

Web Scraping. Weiterhin besteht die Möglichkeit, die Webseite von `GitHub.com` zu nutzen, um Informationen zu Code-Repositorys durch Datenextraktion aus den HTML-Seiten der Webseite zu gewinnen.

Git. Schließlich kann bei Kenntnis der URL eines Repositorys dieses natürlich auch mittels *Git* heruntergeladen werden. Dies ist jedoch zeitaufwändig und für das systematische Mining nur eingeschränkt sinnvoll, da neben den Inhalten eines Repositorys beispielsweise auch immer dessen Historie mit geladen wird.

[7] `https://developer.github.com/v3` (21.8.2019)
[8] `http://ghtorrent.org` (21.8.2019)

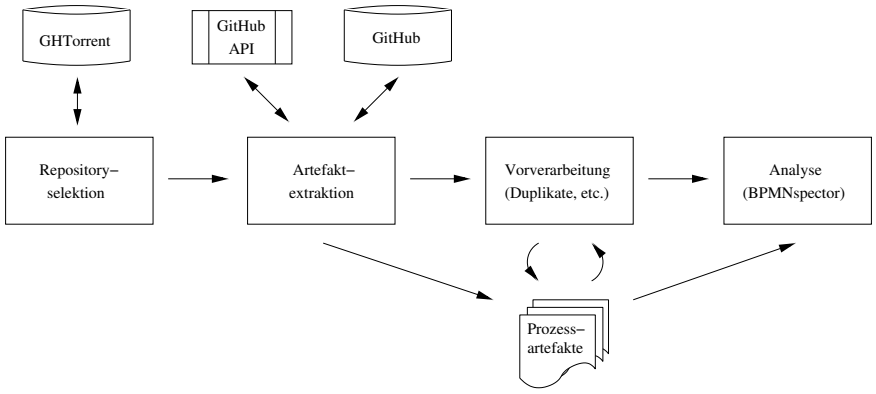

Abb. 1. Schematische Darstellung des realisierten Mining-Prozesses

Für unsere Arbeit wurde analog [6,10] eine Kombination der Ansätze gewählt (siehe auch Abbildung 1). Im ersten Schritt wurde unter Verwendung von *GHTorrent* eine Liste aller Code-Repositorys auf `GitHub.com` erzeugt. Anschließend wurden gelöschte Repositorys oder abgeleitete Repositorys (sogenannte *forked* Repositorys) aus dieser Liste wieder entfernt. Im nächsten Schritt wurden für eine Auswahl der verbleibenden Repositorys via *GitHub API* relevante Metadaten sowie die Repository-Struktur abgefragt und in einer lokalen Datenbank gespeichert. Anhand der Repository-Struktur wurde im Standardzweig (*Default Branch*) eines Repositorys nach möglichen BPMN-Prozessartefakten gesucht. Als Suchkriterium diente dabei das Auftreten der Zeichenkette `"bpmn"` im Dateinamen. Falls entsprechende Artefakte identifiziert werden konnten, wurde das Code-Repository mittels *Git* heruntergeladen und das Repository als auch die Artefakte mit der Datenbank verknüpft. Die dadurch gewonnenen BPMN-Prozessartefakte wurden anschließend weiter verarbeitet. So wurden alle Duplikate entfernt und die Dateien ihren Inhalten nach weiter selektiert. Einbezogen wurden Dateien im XML-Format mit dem Standardnamensraum `"http://www.omg.org/spec/BPMN/20100524/MODEL"`, als Indikator auf BPMN-Prozesse der Sprachversion 2.0 [1]. Im Ergebnis konnte auf diese Weise ein Corpus von in XML serialisierten BPMN-Prozessen generiert werden. Im Anschluss erfolgte der eigentliche Analyseschritt auf den gefundenen BPMN-Prozessen durch das Analysewerkzeug *BPMNspector*, dessen Ergebnisse zur weiteren Auswertung ebenfalls in die Datenbank geschrieben wurden.

3 Ergebnisse

Der vorangehend beschriebene Mining-Prozess wurde in Python implementiert und im Frühjahr 2019 ausgeführt (Implementierung und Ergebnisse online[9] [12]). Auf Grundlage des *GHTorrent*-Datensatzes vom 1.11.2018 wurden 61.632.173

[9] `https://github.com/ViktorStefanko/BPMN_Crawler` (21.8.2019)

Dateialter (Jahre)	< 1	1	2	3	4	5	6	> 6	k.A.
Anzahl Artefakte	7.656	5.154	3.291	2.344	712	690	985	404	70

Anzahl Änderungen (inkl. Erstellung)	1	2	3	4	> 4	k.A.
Anzahl Artefakte	16.285	3.378	620	572	427	24

Anzahl Duplikate	keine	1	2	3	4	5	6	7	> 7
Anzahl Artefakte	8.030	652	571	193	110	219	272	342	118

Dateiformat	Anzahl Artefakte
XML (BPMN 2.0)	16.907
XML (Andere)	384
Graphikformat (.jpg/.pdf/...)	1.635
Andere	2.380

Anzahl Entwickler	Anzahl Artefakte
1	19.595
2	982
3	605
> 3	100
k.A.	24

Tabelle 1. Charakteristika gefundener BPMN-Artefakte (insgesamt 21.306 Artefakte)

nicht gelöschte und nicht abgeleitete Code-Repositorys auf `GitHub.com` ermittelt. Da aufgrund von Zeitbeschränkungen nicht alle Repositorys berücksichtigt werden konnten, wurden analog [6] zufällig 10% ausgewählt. Das Mining von Prozessartefakten nahm in paralleler Ausführung mit vier Programminstanzen knapp 31 Tage in Anspruch. Diese Laufzeit ist vor allem der Limitierung von Anfragen durch die *GitHub API* geschuldet und kann durch Verwendung einer größeren Zahl von Programminstanzen und *GitHub API*-Nutzerkonten verringert werden. Nach Abschluss des Mining-Prozesses konnten unter den analysierten 6.163.217 Code-Repositorys, 1.251 Repositorys mit Prozessartefakten identifiziert werden, wobei in diesen insgesamt 21.306 mögliche BPMN-Artefakte, das heißt Dateien mit der Zeichenkette `"bpmn"` im Namen, gefunden wurden.

3.1 Charakterisierung der Prozessartefakte

Im nächsten Schritt wurden zunächst Metadaten für die identifizierten Repositorys und BPMN-Prozessartefakte untersucht. So lassen sich mittels des Werkzeugs *Code Maat*[10] das Alter gefundener Artefakte, die Anzahl an Änderungen und Entwicklern bestimmen (siehe auch Tabelle 1). Zu diesem Zweck wertet das Werkzeug die Versionshistorie der Artefakte in den Code-Repositorys aus.

Im Ergebnis wurde für die überwiegende Zahl von Prozessartefakten (16.101 oder 76%) ein Alter, das heißt ein Zeitraum seit Erstellung oder letzter Änderung, von nicht mehr als zwei Jahren bestimmt. Dies kann zum einen sicher auf den sprunghaften Anstieg der Anzahl von Code-Repositorys auf `GitHub.com` in den letzten Jahren zurückgeführt werden, lässt sich zum anderen aber als

[10] `https://github.com/adamtornhill/code-maat` (21.8.2019)

Indiz für die wachsende Verbreitung von BPMN auffassen. In Übereinstimmung zum Mining von UML-Modellen in [6,10] konnte für die große Mehrheit von BPMN-Prozessartefakten (16.285 oder 76%) festgestellt werden, dass es sich um statische, das heißt nicht modifizierte Inhalte von Code-Repositorys handelt. Mehrere Interpretationen sind für dieses Ergebnis möglich. So kann BPMN analog UML in der Prozessmodellierung, etwa zu Dokumentationszwecken, genutzt werden und daher eine eher statische Rolle in Repositorys spielen [6]. Auch ist der Einsatz von BPMN-Artefakten in Testsammlungen für den Test von Ausführungsumgebungen wie *Activiti* oder *Camunda* denkbar. Tatsächlich deuten die Dateinamen eines Teils der gefundenen BPMN-Prozesse genau darauf hin (beispielsweise `"InclusiveGatewayTest.testLoop.bpmn20.xml"`). Eine tiefergehende Analyse bleibt weiterführenden Arbeiten überlassen. Noch deutlicher als für die Anzahl an Änderungen fällt das Resultat für die Anzahl an Entwicklern von BPMN-Artefakten aus. So ergibt sich für einen geringen Anteil von Artefakten (1.687 oder 8%) eine Beteiligung von mehr als einem Entwickler.

Weiterhin wurde untersucht, inwiefern es sich bei den BPMN-Artefakten um originale Repository-Inhalte handelt. Knapp die Hälfte der Prozessartefakte (10.599) konnten dabei mittels des Werkzeugs *Duplicate Files Finder*[11] als Duplikate anderer Artefakte aus demselben oder aus anderen Code-Repositorys identifiziert werden. Die Duplikate repräsentieren 2.677 oder 13% der gefundenen Artefakte. In der überwiegenden Zahl treten dabei nicht mehr als fünf Duplikate für einen originalen Prozessartefakt auf. Jedoch gab es interessanterweise Fälle, in denen bis zu 89 Duplikate gefunden wurden. Berücksichtigt werden muss, dass die ermittelte Zahl an Duplikaten lediglich eine untere Grenze für deren tatsächliche Anzahl auf `GitHub.com` ist, da nur 10% der Code-Repositorys in den Mining-Prozess einbezogen wurden. Die genauere Untersuchung der Ursachen für die hohe Anzahl von Duplikaten ist Gegenstand weiterführender Arbeiten. Eine mögliche Erklärung könnte in manuell abgeleiteten Repositorys oder anderen Arten der Wiederverwendung von Repository-Inhalten liegen [6].

Werden die Dateiformate der Prozessartefakte betrachtet, bestätigt sich die Erwartung, dass es sich bei den meisten von ihnen (16.907 oder 79%) um in XML serialisierte BPMN-Prozesse handelt. Daneben spielen Grafik- und Dokumentenformate eine gewisse Rolle. Wiederum liegt der Anteil an Duplikaten für die identifizierten BPMN-Prozesse bei knapp der Hälfte (8.003 oder 47%). Insgesamt konnte auf diese Weise ein Corpus von 8.904 originalen in XML serialisierten Prozessen der BPMN-Sprachversion 2.0 generiert werden.

3.2 Resultate der Analyse mit *BPMNspector*

Im nachfolgenden Analyseschritt wurden die identifizierten BPMN-Prozesse mit *BPMNspector* analysiert. Ziel der Analyse war dabei, zu untersuchen, inwiefern die durch das Analysewerkzeug *BPMNspector* adressierten Syntax- und Semantikfehler sowie die Prüfung auf diese von praktischer Bedeutung ist.

[11] `http://doubles.sourceforge.net` (21.8.2019)

Prozesse ohne Regelverletzung	1.471
Prozesse mit Regelverletzung	7.376
keine Analyse möglich	57
Prozesse insgesamt	8.904

Regelverletzung	Ext.101	Ext.023	Ext.107	Ext.150	Ext.151
Betroffene Prozesse	5.299 (60%)	5.206 (58%)	5.112 (57%)	1.846 (21%)	1.796 (20%)

Regelverletzung	Ext.023	XsdCheck	Ext.107	Ext.092	Ext.101
Absolute Anzahl	58.015 (39%)	43.516 (29%)	7.398 (5%)	6.788 (5%)	6.699 (5%)

Regelverletzung	Kurzbeschreibung
Ext.023	Fehlender ein-/ausgehender Sequenzfluss für Knoten
Ext.092	Quellendefinition für Datenassoziation nicht eindeutig
Ext.101	Fehlender ausgehender Sequenzfluss für Startereignis
Ext.107	Endereignis definiert keinen eingehenden Sequenzfluss
Ext.150	Fehlender Sequenzfluss für Knoten (ohne Startereignis)
Ext.151	Fehlender Sequenzfluss für Knoten (ohne Endereignis)
XsdCheck	Verletzung des normativen XML-Schemas für BPMN

Tabelle 2. Top-5 der mittels *BPMNspector* identifizierten Regelverletzungen

Als Resultat der Analyse ergab sich nur für eine Minderheit der Prozesse keine Verletzung von Syntax- und Semantikregeln aus dem BPMN-Standard [1,3]. Hingegen lag für 7.376 Prozesse (83%) mindestens eine Regelverletzung vor. Dieses Ergebnis wurde so nicht erwartet und übertrifft die in [4] für eine Fallstudie mit 66 Prozessen berichteten Zahlen. Zwar wurden auch dort für die große Mehrheit der Prozesse Syntax- und Semantikverletzungen festgestellt, jedoch zumindest 24 Prozesse als fehlerfrei identifiziert. Offenbar treten die durch *BPMNspector* erkannten Regelverletzungen, zumindest bei der Verwendung von BPMN-Prozessen auf `GitHub.com`, sehr häufig auf. Wie bereits in Abschnitt 1 angedeutet, übernimmt *BPMNspector* dabei die Rolle eines *Linters* für BPMN. Dies wird für die am häufigsten identifizierten Regelverletzungen deutlich (siehe auch Tabelle 2). So überwiegen, übereinstimmend mit [4], sowohl bei Zählung betroffener Prozesse als auch bei absoluter Zählung, Verletzungen solcher Regeln, die einfache Konsistenzbedingungen für die Definition des Kontrollflusses vorgeben (Ext.023, Ext.101, Ext.107, Ext.150, Ext.151 [3]).

Bei näherer Betrachtung sticht insbesondere die Regelverletzung Ext.023 heraus.[12] Diese beschreibt die inkonsistente Definition des ein- oder ausgehenden Kontrollflusses für einen Prozessknoten. Zur Illustration diene das folgende Prozessbeispiel (Artefakt `"InitiatorTest.testInitiator.bpmn20.xml"`):

[12] vgl. `https://github.com/matthiasgeiger/BPMNspector-fixSeqFlow` (21.8.2019)

```
<process id="InitiatorProcess" isExecutable="true">
  <startEvent id="start" camunda:initiator="initiator" />
  <sequenceFlow id="flow1"
      sourceRef="start" targetRef="theTask" />
  <userTask id="theTask"
      name="my task" camunda:assignee="${initiator}" />
  <sequenceFlow id="flow2"
      sourceRef="theTask" targetRef="theEnd" />
  <endEvent id="theEnd" />
</process>
```

Der Prozess enthält drei Knoten: Start- und Endereignis und Aktivität `theTask`. Die Sequenzflüsse `flow1` und `flow2` sollen den Startknoten mit der Aktivität und die Aktivität mit dem Endknoten verbinden. Der BPMN-Sprachstandard [1] gibt nun vor, dass ein Sequenzfluss, etwas redundant, als `SequenceFlow`-Element zu definieren ist und dessen Quelle und Ziel das Element mittels der Attribute `incoming` und `outgoing` referenzieren sollen. Offensichtlich fehlen die Attribute hier sowohl für die Elemente zur Definition des Start- und Endereignis, als auch für das `userTask`-Element zur Definition der Aktivität, standardtreu müsste für letzteres beispielsweise stehen:

```
...
<userTask id="theTask"
    name="my task" camunda:assignee="${initiator}">
  <incoming>flow1</incoming>
  <outgoing>flow2</outgoing>
</task>
...
```

Das Analysewerkzeug *BPMNspector* identifiziert dementsprechend für diese Prozessdefinition Ext.023 als Regelverletzung. Versuchsweise wurden Regelverletzungen dieser Art für die Prozesse des Corpus behoben und die reparierten Prozesse danach noch einmal mit *BPMNspector* analysiert. Im Ergebnis hat sich die Zahl der Prozesse mit mindestens einer Regelverletzung um knapp die Hälfte auf 4.106 von 8.904 Prozessen (46%) verringert. Wie bereits in [4] angemerkt, scheint dies vor allem auf die mangelnde Analyseunterstützung in den zur Prozessmodellierung eingesetzten Entwicklungsumgebungen hinzudeuten und die Notwendigkeit von Werkzeugen wie *BPMNspector* zu unterstreichen.

4 Zusammenfassung

In diesem Beitrag haben wir exemplarisch für die domänenspezifische Sprache BPMN gezeigt, wie sich durch systematisches Mining auf `GitHub.com` eine umfassende Sammlung von Programmen, beziehungsweise Prozesse im Fall von BPMN, generieren lässt. Ein solcher Corpus kann dann für verschiedene Zwecke eingesetzt werden [10]. Hier haben wir die Validierung der praktischen Relevanz

eines Analysewerkzeugs für BPMN untersucht. Im Einzelnen wurden 10% der Repositorys auf `GitHub.com` nach BPMN-Prozessartefakten durchsucht. Im Ergebnis konnten 1.251 Repositorys mit insgesamt 21.306 Artefakten identifiziert werden. Unter diesen befanden sich 16.907 BPMN-Prozesse und nach Entfernung von Duplikaten ergab sich ein Corpus von 8.904 in XML serialisierten BPMN-Prozessen der Sprachversion 2.0. Die nachfolgende Analyse der Prozesse mittels *BPMNspector* ergab für 7.376 oder 83% der Prozesse mindestens eine Verletzung von Regeln des BPMN-Sprachstandards und unterstützt damit die Aussagen in [4] zur Notwendigkeit von Analysewerkzeugen wie *BPMNspector*.

Einschränkungen der Aussagekraft. Die Anwendungsdomäne von BPMN ist die IT-Unterstützung von Geschäftsprozessen. Diese finden vorwiegend im Wirtschaftssektor Anwendung, so dass Prozesse sehr oft das Produkt sind und daher nur selten veröffentlicht werden [11]. Demnach ist anzunehmen, dass die Verwendung von BPMN in Verbindung mit öffentlichen Code-Repositorys teilweise anderen Zwecken dient. Die Verallgemeinerung der Ergebnisse dieses Beitrags ist damit nur eingeschränkt möglich. Es bleibt weiterführenden Arbeiten überlassen zu untersuchen, inwiefern sich BPMN-Prozesse aus der natürlichen Anwendungsdomäne und aus öffentlichen Code-Repositorys ähneln.

Die Datenquelle für das Mining war auf `GitHub.com` beschränkt und bezog nur 10% der Repositorys mit ein. Zudem war der Ansatz für die Identifizierung von BPMN-Prozessen recht simpel und hat nicht alle Prozessartefakte erfasst. Die Zulässigkeit einer Verallgemeinerung oder Übertragung der Ergebnisse für alle BPMN-Artefakte auf `GitHub.com` oder auch für andere Plattformen muss daher noch näher untersucht werden. In Anbetracht der Anzahl identifizierter Prozesse und der Tatsache, dass `GitHub.com` eine der führenden Plattformen ist, sind wir jedoch davon überzeugt, dass die Ergebnisse ein Indiz auf die Verwendung von BPMN in öffentlichen Code-Repositorys liefern.

Eine stichprobenartige Inspektion der identifizierten BPMN-Artefakte hat eine Vielzahl von Prozessen erkennen lassen, die in Testsammlungen für den automatisierten Test von Ausführungsumgebungen verwendet werden (vergleiche mit Abschnitt 3.1). In Verbindung mit der hohen Anzahl an Duplikaten kann daher eine Verzerrung vorliegen, die eine Verallgemeinerung von Schlussfolgerungen auf Grundlage des generierten Corpus von BPMN-Prozessen einschränkt. Darüber hinaus ist davon auszugehen, dass ein Teil der Artefakte aus Code-Repositorys stammt, die zur Aus- und Weiterbildung eingesetzt werden und nicht der eigentlichen Anwendungsdomäne von BPMN entsprechen [6,10].

Literatur

1. *Business Process Model and Notation (BPMN), Version 2.0.* Object Management Group (OMG) Standard, 2011. – `https://www.omg.org/spec/BPMN/2.0/PDF`
2. Fahland, Dirk ; Favre, Cédric ; Jobstmann, Barbara ; Koehler, Jana ; Lohmann, Niels ; Völzer, Hagen ; Wolf, Karsten: Instantaneous Soundness Checking of Industrial Business Process Models. In: *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, Springer, 2009 (Lecture Notes in Computer Science 5701), S. 278–293

3. Geiger, Matthias: *BPMN 2.0 Process Model Serialization Constraints*. Bamberger Beiträge zur Wirtschaftsinformatik und angewandten Informatik, Nr. 92, 2013
4. Geiger, Matthias ; Neugebauer, Philipp ; Vorndran, Andreas: Automatic Standard Compliance Assessment of BPMN 2.0 Process Models. In: *Proceedings of the 9th Central European Workshop on Services and their Composition (ZEUS 2017), Lugano, Switzerland, February 13-14, 2017*, CEUR-WS.org, 2017 (CEUR Workshop Proceedings 1826), S. 4–10
5. Gousios, Georgios: The GHTorent dataset and tool suite. In: *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, IEEE Computer Society, 2013, S. 233–236
6. Hebig, Regina ; Quang, Truong H. ; Chaudron, Michel ; Robles, Gregorio ; Fernandez, Miguel A.: The Quest for Open Source Projects that use UML: Mining GitHub. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016*, ACM, 2016, S. 173–183
7. Heinze, Thomas S. ; Amme, Wolfram ; Moser, Simon: Static analysis and process model transformation for an advanced business process to Petri net mapping. In: *Software: Practice and Experience* 48 (2018), Nr. 1, S. 161–195
8. Heinze, Thomas S. ; Türker, Jasmin: Certified Information Flow Analysis of Service Implementations. In: *Proceedings of the 11th IEEE Conference on Service-Oriented Computing and Applications, SOCA 2018, Paris, France, November 20-22, 2018*, IEEE Computer Society, 2018, S. 177–184
9. Prinz, Thomas M. ; Amme, Wolfram: Practical Compiler-Based User Support during the Development of Business Processes. In: *Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers*, Springer, 2013 (Lecture Notes in Computer Science 8377), S. 40–53
10. Robles, Gregor ; Ho-Quang, Truong ; Hebig, Regina ; Chaudron, Michel ; Fernandez, Miguel A.: An extensive dataset of UML models in GitHub. In: *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, IEEE Computer Society, 2017, S. 519–522
11. Skouradaki, Marigianna ; Roller, Dieter ; Leymann, Frank ; Ferme, Vincenzo ; Pautasso, Cesare: On the Road to Benchmarking BPMN 2.0 Workflow Engines. In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*, ACM, 2015, S. 301–304
12. Stefanko, Viktor: *Repository-Mining von BPMN-Prozessen.* unveröffentlichte Bachelorarbeit, Friedrich-Schiller-Universität Jena, 2019