# Chapter 4
# Analysis of Data-Flow Complexity and Architectural Implications


**Daniel Lübke, Tobias Unger, and Daniel Wutke**

**Abstract** Service orchestrations are frequently used to assemble software components along business processes. Despite much research and empirical studies into the use of control-flow structures of these specialized languages, like BPEL and BPMN2, no empirical evaluation of data-flow structures and languages, like XPath, XSLT, and XQuery, has been made yet. This paper presents a case study on the use of data transformation languages in industry projects in different companies and across different domains, thereby showing that data flow is an important and complex property of such orchestrations. The results also show that proprietary extensions are used frequently and that the design favors the use of modules, which allows for reusing and testing code. This case study is a starting point for further research into the data-flow dimension of service orchestrations and gives insights into practical problems that future standards and theories can rely on.

## 4.1 Introduction

The usage of analytical business processes is common in practice and has been the subject of many research projects. The logical next step, the execution of business process models, is nowadays catching up on both practical usage and a research subject.

D. Lübke (✉)
FG Software Engineering, Leibniz Universität Hannover, Hannover, Germany
e-mail: daniel.luebke@inf.uni-hannover.de

T. Unger
Opitz Consulting Deutschland GmbH, Gummersbach, Germany

D. Wutke
W&W Informatik GmbH, Ludwigsburg, Germany
e-mail: daniel.wutke@ww-informatik.de



D. Lübke, C. Pautasso (eds.), *Empirical Studies on the Development of Executable Business Processes*, https://doi.org/10.1007/978-3-030-17666-2_4

So far, most research has focused on the control flow of processes, e.g., the graph-based structures in BPMN [20] or the usage of activities in BPEL. For example, Hertis and Juric [7] and Lübke [9] analyzed control-flow dimensions of industrial BPEL processes.

However, for executable processes, especially those that orchestrate multiple services, the data-flow dimension is also important: data needs to be transferred between different activities in the process and needs to be converted into a format consumable by the services being orchestrated.

So far, we know no publications that deal with the implementation and the complexity of data flow in executable business processes and their relationship to the control flow.

Without knowing the data-flow dimension, existing approaches to model, test, and verify business processes cannot judge whether and to what extent they must include the data-flow dimension. Also, there are no reliable sources for practitioners working in implementation projects to estimate implementation and testing effort with regard to the data flow.

In order to fill this gap, we conducted a case study of executable business processes implemented in BPEL that is presented in this chapter. This study aims at providing metrics of data flow and comparing it to the control-flow dimension of processes collected from a number of industry projects. This is a first step to better comprehending the challenges modelers and developers face when developing executable business processes.

Research into data flow has proved difficult because all vendors of BPEL engines provide proprietary implementations and extensions. Without knowing the exact causes, this can be a sign that the technologies provided by the BPEL standard are insufficient and/or that the data-flow implementation is an important development task that vendors chose to optimize in order to better sell their products.

The case study presented in this paper was conducted based on a collection of executable BPEL processes from three companies from different domains, ranging from processes for system-internal service integration to cross-organizational business processes. The analyzed process models target one out of three different BPEL engines and are built using the respective vendor-specific modeling tool and employ the vendor-specific BPEL extension supported by the target platform.

This paper is structured according to the suggestion by Runeson et al. [16]: First, related work is discussed in Sect. 4.2 before BPEL as the modeling language of the process models used in the case study is shortly introduced in Sect. 4.3. The case study design is outlined afterward in Sect. 4.4, and its results are presented in Sect. 4.5. The latter contains subsections for detailing the metrics and their interpretation as well as possible threats to their validity. Finally, conclusions and possible future work are given in Sect. 4.6.

## 4.2 Related Work

### *4.2.1 Earlier Studies*

There are not that many but still some empirical studies on the practical usage of BPEL and BPMN.

Cardoso [4] tried to empirically validate process-flow metrics for BPEL processes with a complexity metric defined by him. However, no data-flow dimensions are discussed.

zur Muehlen and Recker [20] did the first study into the practical usage of BPMN: they studied which visible BPMN elements were used by different stakeholder groups. Because the executable information, especially data input and output, is stored in non-visible attributes, the study does not contain any information about it. In addition, the analyzed process models are descriptive only.

Hertis and Juric [7] did a much larger study into metrics of BPEL processes: however, they collected process-flow-related data only, e.g., different activity counts and activity usage patterns. No data-flow metrics were described nor gathered. Also, Lübke [9] analyzed timelines of static BPEL metrics in an industry project. These metrics were process flow related, and no insights about data flow could be taken from those. Thus, data-flow dimensions of industry BPEL processes are not known.

Song et al. [18] conducted an empirical study on data-flow bugs in BPEL processes. However, the authors did not characterize the data-flow dimension itself but concentrated on three data-flow bug categories.

All in all, no empirical studies into the characterization of data-flow dimensions of executable business processes in BPEL or BPMN2 have been made to the authors' knowledge as of today.

### *4.2.2 Theory*

One of the reasons no empirical studies about the data-flow dimension of processes have been made might be that in the history of the research into business processes. Empirical research has mainly concentrated on analytical models. Even with the rise of standardized executable languages, namely, BPEL and BPMN2, research has mostly concentrated on the already existing properties of analytical models: process-flow complexity.

As a result, not many publications about data flow are available which in turn might explain the missing empirical evidence: if no theories are created that need to be verified, empirical research has no research questions to answer.

Cardoso [5] first raised the question on how to measure data-flow metrics. The metric that would measure the code complexity of data transformations is called "interface integration complexity" by him. However, the paper is only a position

paper that concludes with the question "[h]ow to calculate the interface integration complexity of BPEL4WS containers."

Parizi and Ghani [15] also raised the question of measurements for data-flow complexity. However, they only cite Cardoso's original question and offer no further theory or answers themselves.

Some related work is available from the GRID domain, in which BPEL processes have been used to orchestrate academic workflows. For example, Slomiski [17] compared different approaches and GRID-specific challenges like handling large data sets and streaming data. However, in usual business application domains, data is not that large but structured in a more complex manner: data often needs to be converted between heterogeneous data models, and conversion frequently involves conditional logic to determine the attributes that need to be copied and possibly converted.

The importance of considering data flow in addition to control flow in the context of formal verification of BPEL processes has been recognized by Moser et al. [13] and Zheng et al. [19] where the authors describe algorithms for deriving the data flow of BPEL processes and incorporating it into formal process representations, such as Petri nets or automata.

A related area to service orchestrations is the design of service choreographies, in which services are not centrally orchestrated but instead call each other. Meyer et al. [12] present an approach that relies on a global data model that is mapped to the local data model of each service. The approach visualizes mappings by the use of UML diagrams and references existing standards like OWL and XPath but does not try to assess which data transformation technique would fit the approach and how much development effort this layer requires. Nikaj et al. [14] also present an approach to derive a REST service design from BPMN choreographies. While the approach helps to identify resources and appropriate verbs, the data model is clearly marked as out of scope.

## 4.3 Business Process Execution Language (BPEL)

The Business Process Execution Language (BPEL) is a language for modeling executable business processes and standardized by OASIS [8]. It is focused on orchestrating web services that are described by WSDL and XML Schema.

BPEL is defined by a set of *activities* that is split into *basic activities* and *structured activities*. Basic activities perform a function, e.g., calling a service (`invoke`), doing data transformations (`assign`), or waiting for an inbound message (`receive`). Structured activities contain other activities and define the control flow between them, e.g., executing one activity after another (`sequence`) or looping (`forEach`, `repeatUntil`, `while`). The structured activity `flow` allows graph-based modeling and parallel activity execution. All other activities are block based and may only be nested hierarchically.

For handling XML data, BPEL mandates XPath and—via an XPath extension function—XSLT. XPath can be used for conditions (e.g., in an `if` or as a loop condition) or for copying data. Data copies are defined in `copy` elements inside an `assign` activity. A small code snippet copying data from a received message to a response message, for example, is implemented as follows (namespace prefixes have been omitted for clarity):

```
1  <process>
2    ...
3    <variable messageType="sayHello" name="sayHelloRequest" />
4    <variable messageType="sayHelloResponse" name="sayHelloResponse" />
5    <variable messageType="name" name="string" />
6    ...
7    <sequence>
8      <receive
9        name="ReceiveSayHello"
10       operation="sayHello"
11       variable="sayHelloRequest"
12     ... />
13     <assign name="PrepareResponse">
14       <copy>
15         <from>bpel:doXslTransform(
16           'prepareSayHelloResponse.xsl',
17             $sayHelloRequest.parameters)
18         </from>
19         <to part="parameters" variable="sayHelloResponse"/>
20       </copy>
21       <copy>
22         <from>$sayHelloRequest.parameters/lastName)</from>
23             <to variable="name"/>
24           </copy>
25     </assign>
26     <reply
27       name="ReplySayHello"
28       operation="sayHello"
29       variable="sayHelloResponse"
30     ... />
31   </sequence>
32 </process>
```

A message is received by the receive activity (line 7) and copied to the variable *sayHelloRequest*. The variable *sayHelloRespnse* is prepared by an assign activity (line 13). The assign has two copy blocks. The first copy (line 14) uses XSLT via BPEL's built-in *doXslTransform* XPath function and copies the result to the response. The second copy (line 21) simply copies the result of an XPath expression to an atomic variable. The reply (line 26) sends the newly created message to the caller.

Like most WS-* standards, BPEL is designed to be extensible: new query and expression languages besides XPath can be referenced by the use of URNs, and a placeholder `extension` activity can contain vendor-specific activities.

Although BPEL has been superseded by the BPMN standard, it is still used, and many companies have large repositories of BPEL processes that contain lessons learned that apply not only to BPEL but to executable business processes and service orchestrations in general.

## 4.4 Case Study Design

### *4.4.1 Research Questions*

We formulate our research goal according to the goal/question/metric (GQM) method [1]:

> The purpose of this study is to *characterize* the *implementation of data flow in BPEL processes* from the point of view of *a solution architect* in the context of *executable business process development projects*.

We refined this overall research goal into the following questions:

**RQ1: Which data-flow modeling choices are preferred on specific tools?** The BPEL standard itself supports XPath and XSLT (via an XPath extension function). However, BPEL is designed with many extensibility points. One of those extensions can be the use of other languages to formulate expressions and queries on XML data. For example, many BPEL engines support XQuery (e.g., Apache ODE and its derivatives, ActiveVOS, and Oracle BPM Suite), while others allow to embed Java code or offer custom XML data mappings (both options, e.g., IBM WebSphere Process Server (WPS)). When several data-flow implementation choices are available, the question which ones are preferred (or being pushed upon) by developers arises. We hypothesize a) that the developers prefer to use the proprietary extensions provided by the tools, which in general should be more prominently offered in the development tools and should be more powerful than the standard ones because otherwise the tool vendors would have had no incentive to implement those, and b) that the most powerful options XQuery and Java are preferred over other implementation choices. We measure this by counting lines of code and expect XQuery and Java to have the largest amount of lines on ActiveVOS/Oracle and WebSphere Process Server, respectively.

**RQ2: What amount of data flow is portable, i.e., standards compliant?** Because we expect the proprietary data-flow implementation choices to be preferred by the developers, our hypothesis is that no BPEL process is fully standards compliant with regard to its data-flow implementation. Because we expect XPath and XSLT to be used nevertheless in some spots (e.g., for formulating conditions and transforming XML data, where XSLT can excel in some circumstances), some portions of the data flow are expected to be standards compliant, i.e., use XPath and/or XSLT. Because we expect most XML messages to be produced by non-standards-compliant code and we expect those to make the bulk of data-flow implementation code, we hypothesize that less of 10% of

the lines of data-flow code are implemented in one of the languages offered by the BPEL standard.

**RQ3: Is the data flow in executable business processes larger than the process flow?** Because executable business processes possibly connect to many different systems exposing different services and business objects (BOs), we expect data transformations to be an integral and large part of a business process solution. As metrics for measuring complexity, we use the number of conditional branches (e.g., if and switch) and the number of iterations (e.g., for, while, and repeat until loops). For measuring the size, we use the number of lines of code (data flow) and the number of basic activities (process flow). For all these metrics, we hypothesize that the data-flow dimension is larger than the process-flow dimension: (1) We estimate that more lines of code are needed than there are basic activities because the XML messages usually contain more than ten elements. (2) We also expect that there are more conditions and loops. The more statements exist (regardless of the abstraction level), the more conditions and loops are required to order them. Following of (1)+(2), we expect more data-flow conditions and iterations, although we doubt any direct relationship. Originally, we planned to compare not only LOCs and counts but also the complexity of BPEL control-flow structures and the complexity of the data-flow implementations. However, while McCabe's cyclomatic complexity [11] can be easily applied to XQuery, no adaption to BPEL nor XSLT is available. Cardoso's complexity metric [3] has many weights in formulas and is not well defined for all graph-based processes (BPEL's flow activity). The weights forbid direct comparisons to McCabe's unweighted complexity metric. Therefore, we decided to use counts of iterations and conditions instead of a more sophisticated complexity metric.

**RQ4: Are data-flow implementations mainly large but linear or mainly complex?** From an architecture perspective, an interesting question is where complexity is located. Thus, one important question is whether the code concerned with the data flow is not only large compared to the process flow but whether it is mainly linear, thus "easy" code, or whether it contains many control-flow structures. We expect the data-flow code to be simple because we expect most of the code to simply insert values into XML templates. Only at some points we expect decisions for optional elements or loops for lists. However, we expect more conditions than loops. Therefore, we hypothesize that we have maximum one condition per four lines of data transformation code and maximum one iteration per five lines of code.

**RQ5: What are possible factors for increased complexity of data flow?** From an architecture point of view, it is important to know and identify drivers of data-flow complexity to better plan and estimate implementation and testing. Because we think that data flow is mainly needed to prepare messages, we hypothesize that the number of message exchange activities (receive, reply, invoke, onMessage, onEvent) correlates linearly to the lines of data-flow code. If this correlation holds, it can be used on analytical models, which contain the message exchanges, to better judge the technical implementation later on. We

also hypothesize that the data-flow complexity measured by counting conditions and iterations also correlates linearly to the number of message exchange activities: the usage of conditions and iterations is probably dependent on the differences in the schemas being integrated but should behave the same within one project.

### *4.4.2 Case and Subject Selection*

For answering the outlined research questions, we conduct a case study on processes from three different companies. All processes are BPEL processes so that the choices and metrics are comparable and influences of product choices can be isolated from the modeling language.

The processes target different BPEL engines (Informatica ActiveVOS 9.2, IBM WebSphere Process Server 7.1 and WebSphere Business Process Manager (BPM) 8.5, Oracle Business Process Management Suite 12c) and are modeled using the respective vendor-supplied modeling tool.

Informatica ActiveVOS is a BPEL engine which supports the full WS-BPEL 2.0 standard but also has proprietary extensions for modeling BPEL processes and visualizing them as BPMN. One of these extensions is the support of XQuery for expressions and queries, i.e., in all places where XPath is allowed. XQuery as a superset of XPath is more powerful and can be used to fully replace XSLT transformations.

IBM WebSphere Process Server (WPS) and its successor Business Process Manager (BPM) are workflow engines on top of a JEE application server. In addition to BPEL, IBM BPM also supports modeling and execution of processes modeled in BPMN. As this study focuses on BPEL processes, only the BPEL-specific aspects of BPM are discussed. Besides WS-BPEL 1.1 processes, WPS/BPM supports the execution of state machines and business rules. Service integration is performed via an integration solution (WebSphere ESB) that comes with the workflow engine. Regarding data flow, WPS/BPM supports standard XPath expressions as well as the vendor-specific business object maps, XML maps, and Java code embedded in the BPEL process model.

Oracle Business Process Management Suite 12c is a toolset and integration platform for development and execution of SOA-based applications. Among other components, the BPEL Process Manager supports the execution of BPEL 2.0 processes. Oracle also provides a set of vendor-specific extensions like XQuery integration in XPath, a replay activity for restarting scopes, and human tasks.

In the following, we present an overview of the processes used in the case study, following the categorization proposed in Chap. 2.

The first project that is contained in our analysis is Terravis (see Table 4.1). Terravis is a process integration platform that allows to conduct cross-organizational processes between land registries, notaries, and banks [2]. The project uses the ActiveVOS BPEL engine, which is developed by Informatica. We analyzed a

**Table 4.1** Aggregated metadata for the ActiveVOS process collection (Terravis, classification according to [10])

| | |
|---|---|
| Collection name | Terravis |
| Process count | 86 |
| Domain | Land register transactions |
| Geography | Switzerland |
| Time | 12-2017 |
| Boundaries | Cross-organizational 23%, intraorganizational 17%, intra-system 60% |
| Relationship | Calls another 17%, calls another/is being called 24%, event triggered 16%, is being called 1% |
| Scope | Core 34%, technical 38%, auxiliary 28% |
| Process model purpose | Executable |
| People involvement | Mostly 17%, partly 3%, none 79% |
| Process language | WS-BPEL 2.0, BPEL4People, plus vendor extensions |
| Execution engine | ActiveVOS 9.2.x |
| Model maturity | Productive 100% |

**Table 4.2** Aggregated metadata for the WPS/BPM process collection

| | |
|---|---|
| Collection name | Banking and Insurance |
| Process count | 75 |
| Domain | Banking, insurance |
| Geography | Germany |
| Time | 05-2017 |
| Boundaries | Cross-organizational 4%, intraorganizational 67%, within system 29% |
| Relationship | Calls another 12%, is being called 64%, is being called/calls another 5%, no call 19% |
| Scope | Core 13%, technical 87% |
| Process model purpose | Executable |
| People involvement | None 92%, partly 8% |
| Process language | WS-BPEL 1.1 plus vendor extensions |
| Execution engine | IBM WebSphere Process Server 7.1, IBM Business Process Manager 8.5 |
| Model maturity | Illustrative 3%, productive 73%, retried 24% |

snapshot taken from the repository, which was taken in December 2017. We needed to exclude processes from this project that use XQuery 2.0 features that are not supported by our analysis tool.

The second process collection contains processes from the banking and insurance domain (see Table 4.2) with a strong focus on technical integration processes. The processes use the IBM WebSphere Process Server and IBM Business Process Manager BPEL engines and integration solutions. The analyzed snapshot was taken in May 2017.

**Table 4.3** Aggregated metadata for the Oracle SOA Suite process collection

| | |
|---|---|
| Collection name | Wholesale and Retail Trade |
| Process count | 23 |
| Domain | Commerce |
| Geography | Europe |
| Time | 2015 |
| Boundaries | Cross-organizational 17%, intraorganizational 83% |
| Relationship | Calls another 96%, is being called 4% |
| Scope | Technical 100% |
| Process model purpose | Executable |
| People involvement | None 96%, partly 4% |
| Process language | WS-BPEL 2.0 plus vendor extensions |
| Execution engine | Oracle SOA Suite 12.1 |
| Model maturity | Illustrative 4%, productive 96% |

**Table 4.4** Proprietary extensions for data-flow definition

| | |
|---|---|
| Informatica ActiveVOS | XQuery in assign activities, import of XQuery modules for usage in XQuery statements embedded into the assign activities |
| Oracle BPM Suite | XQuery in XPath |
| IBM WebSphere Process Server/Business Process Manager | Java, business object (BO) maps, XML maps |

The Oracle process collection shown in Table 4.3 is used to integrate retailers with their suppliers.

### *4.4.3 Data Collection and Analysis Procedure*

In the first step, the data transformations of the collected processes have been analyzed: the main goal is to identify the proprietary extensions used for defining the data flow. The extensions found are presented in Table 4.4.

The data collection for the static metrics itself was done with a custom static code analyzer named BPELStats that had been developed under the umbrella of the BPELUnit project and is now developed as a standalone project. BPELStats has been originally developed for gathering the metrics presented in [9] and is available as open source.[1]

Among other metrics, BPELStats can count BPEL activities by type and has been extended as part of this study to compute the number of occurrences, iterations,

[1] https://github.com/dluebke/bpelstats.

conditions, and LOCs for XPath, XSLT, XQuery, BOMaps, XMLMaps,[2] and Java code. The calculation is based on whole files: if a file is imported into the BPEL process, all functions, templates, etc. are counted toward the metrics, and no check is made whether a certain piece is actually called by the process or not. All of our extensions have been contributed to the BPELStats project and are freely available and can be reviewed.

Clean checkouts of the process projects have been done first. The total sample size contains 184 executable BPEL processes. Where necessary, a full build has been triggered before the analysis when the build is necessary to copy all external dependencies (e.g., WSDLs, XML Schemas, reused XSLT and XQuery files) to their correct positions enabling BPELStats to also follow and resolve imports of those files from the BPEL processes.

### *4.4.4 Validity Procedure*

In order to ensure the correctness of the measurements, the BPELStats tool was tested with both unit tests and manual tests. The whole gathering routines were automated by using shell scripts in order to eliminate human error. These shell scripts were also tested by all researchers in this study in order to show that the results are correctly computed.

For allowing other researchers to replicate this study, the used scripts have been made available.[3] This also allows other researchers to check their correctness.

We tried to cover as many different BPEL toolsets as possible to strengthen external validity. With three completely different BPEL engines used in three large industrial projects at different organizations, we are confident to be able to distinguish influences imposed by the tooling and the project from those that are inherent to the problem of service orchestrations and executable business processes.

## 4.5 Results

Within this section, we present the plain results of our case study, which mainly include the metrics being gathered as part of our measurements. The interpretation of these measurements is presented in the following section.

[2] As XMLMaps are compiled to XSL transformations during build time, their metrics are calculated using the XSLT sublanguage parser and hence show up as XSLT metrics in the results with their occurrences being counted separately.

[3] The files are accessible at http://www.daniel-luebke.de/files/bpm-dataflowcomplexity.tgz.

### 4.5.1 Metrics

For answering our research questions, we collected the metrics required to answer them.

The first analysis computed the occurrences of data flow in the process collection and the lines of code of data-flow code. An occurrence is a place at which data-flow code is embedded into the process. If the process has five assign activities, there are at least five occurrences of data-flow mappings, depending on the number and type of copy statements. Each data-flow occurrence contains at least one line of code but can contain multiple ones. This means that the number of occurrences is equal to or less than the number of lines of code.

We aggregated the metrics as shown in Table 4.5 according to the different languages that we found in the process set. The language which has the most occurrences in our process set is XPath. However, XQuery has more lines of code. Java is the third most often used language followed by XSLT.

We aggregated this data further and clustered the languages into portable (standards compliant, i.e., XPath, XSLT) and non-portable (not mandated by the BPEL standard, i.e., XQuery, Java, BOMaps, and XMLMaps). The results are shown in Table 4.6: nearly half of the occurrences (46%) of data-flow code are portable, but only 8.96% of lines of code are written in standard-mandated languages.

In the next step, we analyzed the relationship between the process flow and the data flow. Figure 4.1 depicts the relationship between the number of basic activities and the lines of data-flow code. The plots indicate a linear relationship between basic activities and data flow. Therefore, we computed Spearson's linear correlation coefficient between these two dimensions, which is $c = 0.8162$ (Terravis), $c = 0.9035$ (Wholesale and Retail Trade), $c = 0.6962$ (Banking and Insurance), and

**Table 4.5** Data-flow occurrences and LOCs by engine and implementation choice

| Metric | ActiveVOS | Oracle BPM | IBM WPS/BPM |
|---|---|---|---|
| XPath occurrences | 2419 | 2380 | 738 |
| XPath LOCs | 2865 | 2380 | 738 |
| XSLT occurrences | 4 | 0 | 0 |
| XSLT LOCs | 4437 | 0 | 0 |
| XQuery occurrences | 4173 | 108 | 0 |
| XQuery LOCs | 21, 888 | 10, 010 | 0 |
| Java occurrences | 0 | 0 | 2193 |
| Java LOCs | 0 | 0 | 13, 298 |
| BOMap occurrences | 0 | 0 | 32 |
| BOMap LOCs | 0 | 0 | 3676 |
| XMLMap occurrences | 0 | 0 | 0 |
| XMLMap LOCs | 0 | 0 | 0 |
| Total occurrences | 6596 | 2488 | 2963 |
| Total LOCs | 29, 190 | 12, 390 | 17, 712 |

**Table 4.6** Percentages of used data-flow technologies

| Portability | Percentage (S)LOCs | Percentage occurrences |
|---|---|---|
| Portable | 8.96% | 46% |
| Not portable | 91% | 54% |

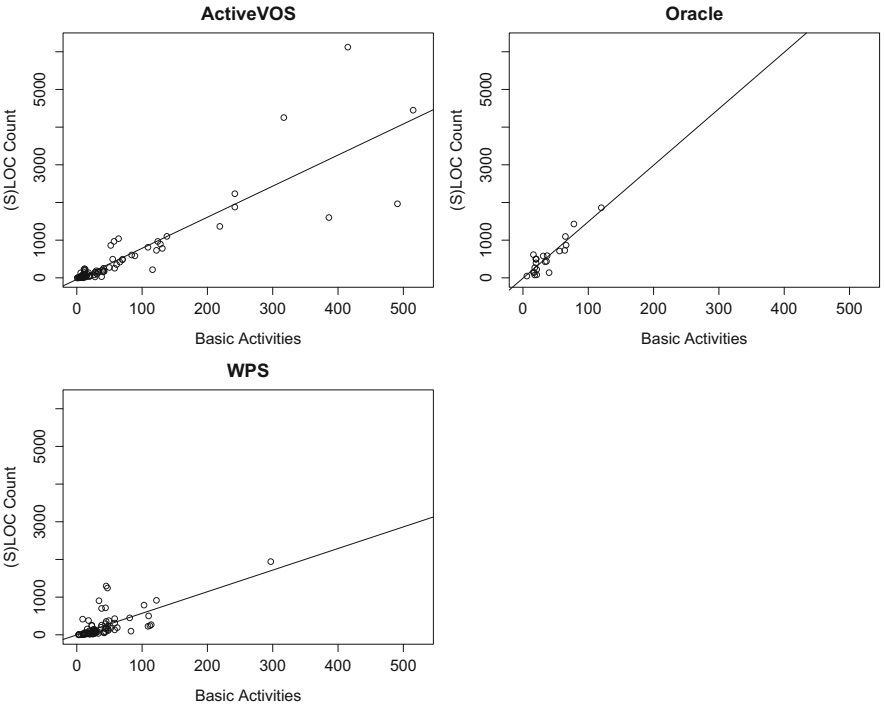


**Fig. 4.1** Data-flow (S)LOC count and number of basic activities

$c = 0.848$ combined for the whole data set. Except for the Banking and Insurance data set, the relationship between the basic activities and the data-flow lines of code is linear. As such, we added regression lines to Fig. 4.1.

In order to judge whether the process-flow or data-flow dimension is larger, we computed a two-sided, paired Wilcoxon test: for the Terravis data set $p = 1.101 \times 10^{-10}$, for the Wholesale and Retail Trade data set $p = 2.886 \times 10^{-5}$, and for the Banking and Insurance data set $p = 5.947 \times 10^{-14}$. If all data sets are combined, the Wilcoxon test results in $p = 1.946 \times 10^{-31}$. All computed $p$-values are much smaller than 0.01, which is a commonly accepted threshold for highly significant results.

In the next step, we drilled down into the nature of the control flow and data flow and computed the number of iterations and conditions within each language. We plotted the conditions of each dimension against each other in Fig. 4.2 and the iterations against each other in Fig. 4.3. For each project alone and all combined, we again computed Spearson's linear correlation coefficient for

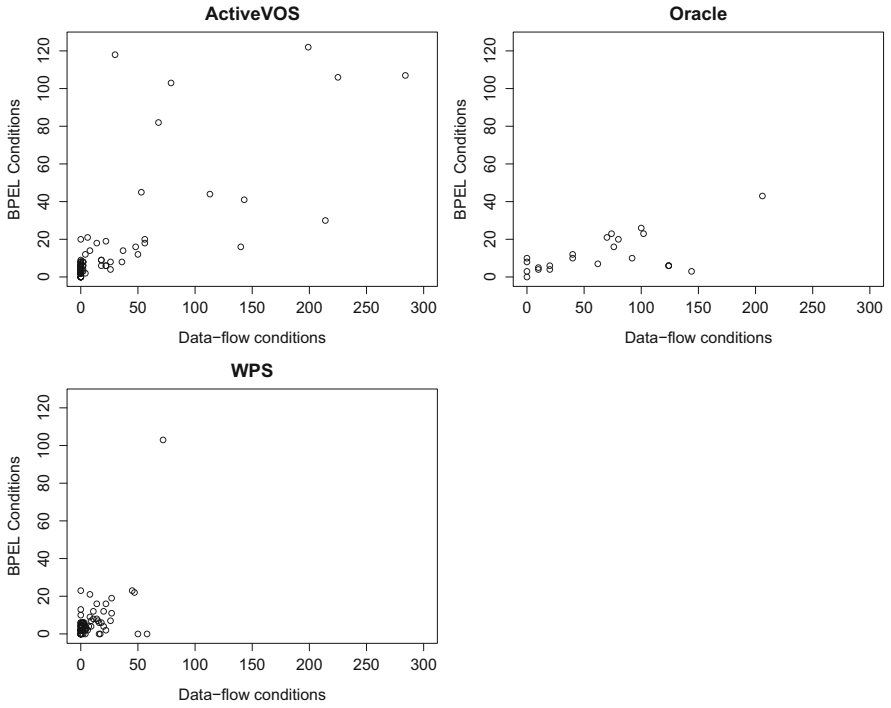


**Fig. 4.2** BPEL conditions and data-flow conditions

both conditions ($c_{cond} = \{0.75, 0.5671, 0.6053, 0.6538\}$) and iterations ($c_{it} = \{0.591, 0.9503, -0.1554, 0.6444\}$). Because $|c| < 0.75|$ holds for correlation coefficients with two exceptions (conditions for Terravis and iterations for Wholesale and Retail Trade), our data cannot support any linear correlation, especially because there are also correlation coefficients with different signs ($c_{it}$ is negative for WPS).

We also conducted a two-tailed, paired Wilcoxon test for differences between the process-flow and data-flow dimension on this data: for conditions of the three projects and all projects combined

$p_{cond} = (0.5215, 0.0001651, 0.6319, 0.07883)$

and for iterations

$p_{it} = (4.997 \times 10^{-7}, 0.0009128, 0.6768, 1.59 \times 10^{-8})$ While for all $p$-values concerning the conditions $p > 0.05$ holds, nearly all $p$-values—except for the WPS project—hold $p < 0.01$.

These differences concerning the different projects and BPEL engines led to another drill-down into the data. As shown in Fig. 4.4, we plotted the distribution of conditions and iterations in relation to the lines of data-flow code. We split this data by projects and additionally combined the processes using XQuery from the Terravis process set and the Wholesale and Retail Trade process set. The plot suggests that the number of conditions and iterations in the data flow is significantly different

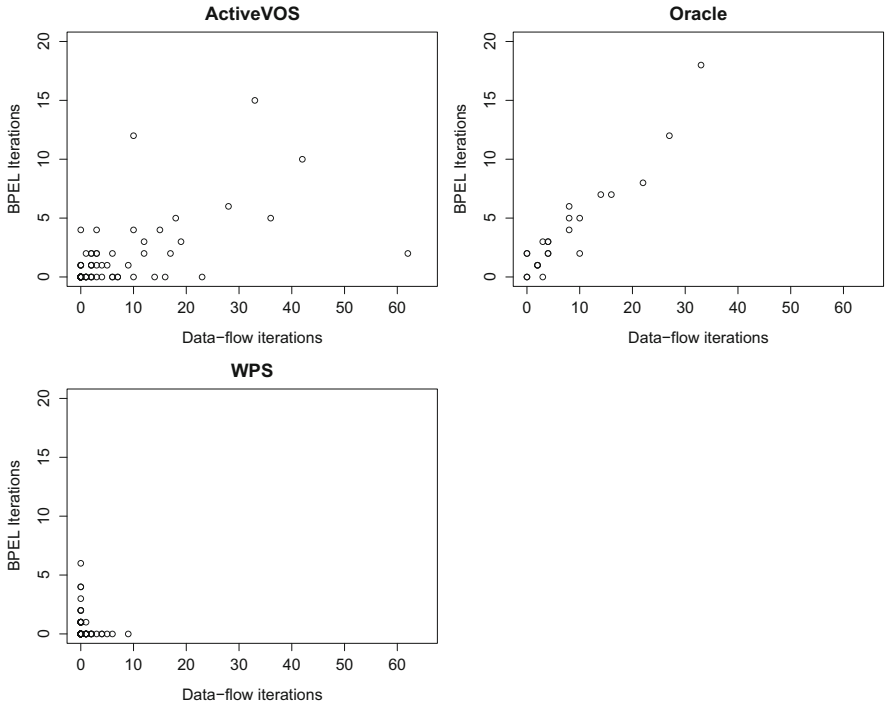


**Fig. 4.3** BPEL iterations and data-flow iterations

between the data sets, also when the data sets use the same language, which is the case in Terravis and Wholesale and Retail Trade for XQuery.

The median values of conditions per lines of data-flow code are below 0.15. Except for the Wholesale and Retail Trade data set, they are even below 0.10. All values for the iterations are below 0.075, and the median values are all below 0.025. The data indicates that there are more conditions per lines of code than iterations.

For answering the last research question, we computed the number of message exchange activities in the processes and computed the correlation coefficient to the lines of data-flow code. The results are summarized in Table 4.7: For two of the three data sets, we get a correlation coefficient c with $c > 0.75$, which hints at a linear relationship. The exception is the WPS data set, which mainly uses Java. The mean of the message activities per lines of data-flow code is between 0.026 and 0.054 in the data sets.

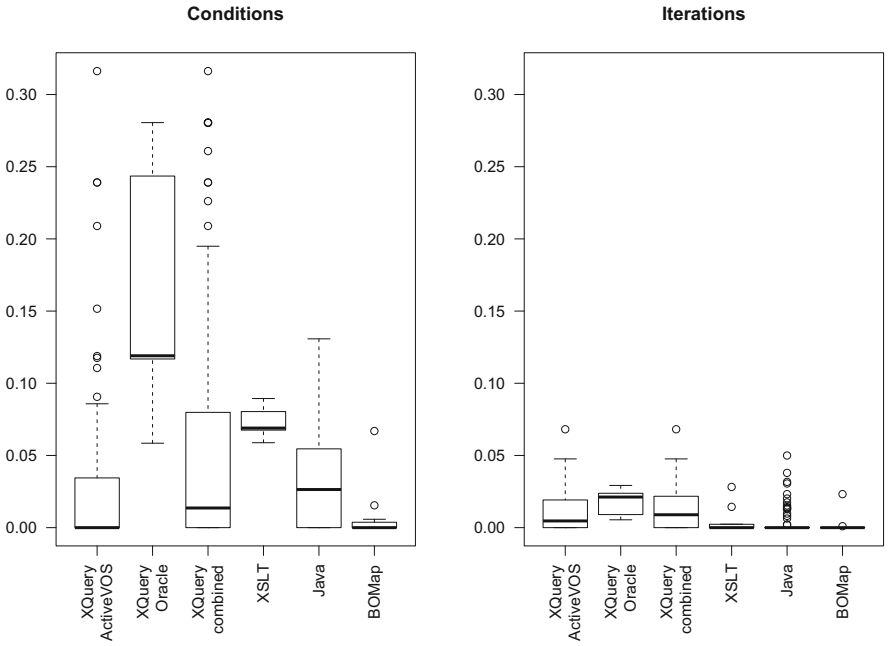


**Fig. 4.4** Number of data-flow conditions and iterations per LOC

**Table 4.7** Message exchanges per process collection and (S)LOC

| Metric | ActiveVOS | Oracle BPM | IBM WPS |
|---|---|---|---|
| Message activities | 2238 | 316 | 750 |
| (S)LOC | 41,100 | 12,390 | 17,712 |
| Message activities/(S)LOC | 0.054 | 0.026 | 0.042 |
| Correlation coefficient | 0.792 | 0.907 | 0.693 |

## *4.5.2 Interpretation*

### 4.5.2.1 RQ1: Which Data-Flow Modeling Choices Are Preferred on Specific Tools?

The results presented in Table 4.5 confirm hypothesis (a) as well as hypothesis (b). However, the result is most succinct for the IBM WPS collection. One reason is that Java can be used almost anywhere in IBM Process Server. Furthermore, IBM's tooling supports the Java extensions very well. ActiveVOS and Oracle BPM do not support Java in conditions.

In the case of Oracle, the interpretation is not that easy. Although XQuery is a popular option for data mapping, the numbers in Table 4.5 show that also XPath is very popular. One reason is that Oracle integrates XQuery as an XPath extension function. Whenever XQuery is used, XPath is also used. This means from a

BPEL perspective you always use a BPEL standard option (XPath), because Oracle decided to extend XPath. The fact that there are much more XQuery LOCs than XPath LOCs confirms that XQuery is the preferred option for data transformation. Furthermore, Oracle provides a modeling tool for XPath.

Compared to IBM and Oracle, ActiveVOS is the most standard compliant engine. However, the disproportionately larger number of XQuery LOCs proves that the vendor extension XQuery is preferred over XPath.

Form a skill perspective, we quite often observe that BPEL processes are modeled by developers. For them, XQuery might be easier to learn than XSLT; and due to Java being a common language used for developing enterprise software, a large number of those developers are probably already familiar with Java, so they do not need to learn new languages like XPath or XSLT.

#### 4.5.2.2 RQ2: What Amount of Data Flow Is Portable, i.e., Standards Compliant?

Table 4.6 shows that the amount of non-portable data-flow code is over 90%, i.e., porting a process would lead to a nearly complete reimplementation of the data flow. However, the occurrences of BPEL-compliant data-flow implementations and implementations using vendor extensions are nearly equal. One reason is that ActiveVOS and Oracle BPEL only support XML-based implementations in conditions. In addition, developers try to make implementation of conditions easy, i.e., XPath is sufficient by moving complicated transformation to assign activities. Ironically, the possibility to use Java in WPS also makes things easier for developers. Java provides a larger set of operators that allow to formulate conditions more easy and compact. For example, the Exclusive OR (XOR) is not supported in XPath but in Java.

Another aspect of portability is that each vendor chooses a different implementation of an extension, even if the extension itself is provided by multiple vendors. For example, the ActiveVOS XQuery extension is not compatible with Oracle's extension, and IBM Java activity is not compatible with Oracle's Java activity. Thus, portability would also not increase if comparing the process collections pairwise.

Our results also support the conclusion that there is something wrong with the standard. Maybe the standard fell victim to its extensibility because every vendor used the extensibility to differentiate its product. Maybe the vendors also only approved the standard because of its extensibility because portability was not really an issue.

For architects, this raises the question which language is the best option to model data flow. This also raises a lot of research questions: What is the best, most productive, easy-to-maintain set of data transformation languages? How to decide on a language in a concrete project setting?

#### 4.5.2.3 RQ3: Is the Data Flow in Executable Business Processes Larger than the Process-Flow?

The highly significant $p$-values (all $p <= 0.01$) confirm our assumption. However, this contradicts our experience from modeling and implementing process-based systems. We spend a large part of the modeling effort on modeling the control flow. The results suggest that we should spend more effort on the data flow. Also, a lot of project guidelines we saw lack in defining rules for data flow. Therefore, guidelines should also provide rules and norms for handling the implementation of the data flow. In addition, the research domain should give data flow more attention.

#### 4.5.2.4 RQ4: Are Data-Flow Implementations Mainly Large but Linear or Mainly Complex?

Table 4.4 shows that the measurement proves our hypothesis to some extent. However, the assumption that we expect less conditions than loops is disproved because we observed significantly different numbers (especially conditions) for each observed language. Especially, the XQuery transformations of the Wholesale and Retail Trade process collection contain more conditions than loops. This aspect needs further research. This research should also investigate whether the design of the XML Schemas, the chosen query language, or even unknown properties influence the number of loops and conditions.

#### 4.5.2.5 RQ5: What Are Possible Factors for Increased Complexity of Data Flow?

Table 4.7 shows a strong linear correlation between message exchange activities and LOCs. One reason might be that transformations are mostly used to prepare a service invocation. In a lot of processes, we observe that an invoke activity is preceded and/or succeeded by assign activities. For example, this correlation allows to estimate effort for implementing data transformations based on the number of message exchange activities. Even if we confirmed the strong linear correlation, the slack between the projects is different. For example, the Wholesale and Retail Trade process collection shows a higher number of LOCs per message exchanges than the IBM collection. The reason for this might be similar to Sect. 4.5.2.1, i.e., the expressiveness of the data transformation language used.

### *4.5.3 Threats to Validity*

We tried to eliminate internal threats to validity as much as possible: because we only rely on automatically and objectively measured metrics for which the metric

tool and all scripts are available for public scrutiny, there should be no measurement errors.

Regarding processes built for the IBM WPS/BPM engines, we identified the following threats to internal validity due to our measurement method: WPS/BPM script elements allow the execution of arbitrary Java code, not only data transformations. The collected metrics for Java code do not differentiate between data transformation and, e.g., business logic and are calculated based on all Java code embedded in the BPEL process model. This classification would be needed to be done manually and is both time-consuming and error prone. In addition, WPS/BPM supports importing Java classes from library projects and external libraries. Both forms of imports have not been considered when calculating the Java metrics. Finally, the WPS/BPM BPEL engine comes with an integration solution (WebSphere ESB) which supports the definition and execution of so-called mediation flows. As part of the integration logic, data can be transformed in various ways, e.g., via XSL transformations or custom Java code. Data transformations defined in mediation flows have not been considered when calculating the respective data-flow metrics in the BPEL processes.

Because this case study is based on a small set of projects, there are however threats to external validity. Especially, the question of generalizability to other BPEL projects has to be raised.

This case study uses three different BPEL tool chains, and we could identify differences between the different projects. However, due to limitations in our data set, we cannot tell whether these differences arise from the tools only—which is certainly true for the different data-flow languages being offered—or whether differences are caused by other project constraints. There are no theories nor empirical evidence for possible impact factors: from our point of view, probable candidates are the difference between the service contracts of composed services and architecture choices that can distribute data transformations between different infrastructure artifacts (e.g., enterprise service bus (ESB) or process implementation) and own services (e.g., custom services implemented in a "classical" programming language like Java).

The results clearly show that more than half of the data flow is implemented with non-portable vendor extensions. To our knowledge, our data set covers all proprietary extensions except JavaScript that is, for example, offered by ActiveVOS. However, under these circumstances, we also expect that the same data-flow complexity is contained therein, even though it might be implemented differently.

Further cases will be needed to judge and strengthen the external validity and also to build and validate first theories about causalities that influence data-flow complexity. This case study will therefore serve as a first piece of the puzzle to unravel the understanding and drivers of data flow in executable business processes, but clearly further steps are required in order to exactly pinpoint influencing factors.

## 4.6 Conclusions and Future Work

### *4.6.1 Conclusions for Researchers*

To our knowledge, this case study was the first empirical study that did not only consider the process-flow dimension but also the data-flow dimension of executable business processes—in this case modeled in BPEL.

Our study clearly shows that the data-flow dimension is larger than the process-flow dimension regarding both statements (LOCs vs. activities) and complexity (conditions and iterations). This means that theoretical concepts, e.g., for verifying process flow, fail to deliver a complete solution until they are not extended by data-flow analysis. This also means that with current techniques, verification of the process flow alone cannot replace testing of executable business processes because the tests are covering the data transformations that are otherwise left out. While "processes are not data," it is certainly true that "processes cannot be implemented without data flow."

The study also shows that BPEL processes, although they are modeled in a standardized language, are amended with many proprietary extensions. This in turn means that researchers need to have knowledge about the tools that have been used to develop the processes being researched.

Our case study resulted in a new possible estimation metric: the number of message exchanges that can be computed on the analytical process model prior to implementation of the executable model correlates very well to the lines of data-flow code. However, the average number of message exchanges per lines of data-flow code varies between 0.03 and 0.05 between the data sets.

Better formalization of data transformations would improve the possibilities to analyze the data flow from a research perspective. With this in mind, BPMN 2.0 as a standard lacks even behind BPEL with all proprietary extensions: the BPMN standardization committee opted to not standardize any technical bindings. As such, there are no data transformation languages mandated by the standard. Instead, the choice is left to the tool vendor. This means that BPMN research with the goal of gaining insights into executable BPMN in a general sense is probably impossible. A realistic goal would be to study the BPMN "dialect," which is created by the vendor implementation. With this in mind, the usage of BPEL processes as research subjects is probably easier but warrants the question of how much of the empirical results gained in BPEL processes can be generalized to BPMN executable processes.

### *4.6.2 Conclusions for Practitioners*

While some BPM vendors advertise their tools with "zero code" and other buzzwords, our study clearly shows that the data-flow dimension is important and even larger and more complex than the process-flow dimension. Practitioners and

especially software architects need to be aware of the implications of the "hidden logic" that is usually hidden behind nice-looking models:

- Especially, data-flow implementation seems to be highly specific to a chosen BPMS. None of the studied projects achieved portability. This means that by choosing a tool, the project will probably be dependent on this tool in the future if high migration costs are to be avoided.
- Data flow builds its own new layer of complexity. This means that effort for development and especially testing the higher number of conditions and iterations needs to be planned for. Also, architects should consider making guidelines on how to structure, format, etc. data-flow code in their projects to help developers build better and more maintainable executable processes.
- For better communication between stakeholders, it would be good to see the data flow clearer in the analytical models. Usage of data objects in BPMN is one way, which are only used in about 25% of all models [20], to better communicate which data is sent/received from where.
- We have shown that the number of message exchanges can be used to predict the lines of data-flow code. While this needs to be further improved into an estimation method, architects can leverage this knowledge.
- Although data-flow code uses conditions and iterations, it only uses few of them. This means that most of the code is fairly easy and should be easy to develop and test.
- Additionally, the overall solution architecture should address the data flow more. At which components or areas inside or outside the executable process should what type of data-flow logic be placed? In the projects analyzed as part of this research, most code was embedded directly into the BPEL processes themselves, while only a small amount was placed in surrounding areas like an enterprise service bus or supporting services. For example, Danei et al. [6] propose a multilayered approach with clear responsibilities for each layer. Empirical analysis should be used to gain further insight into benefits and drawbacks of different architectural choices.

### *4.6.3 Outlook and Future Work*

The ultimate goal should be to identify causal relationships between projects' properties and their data-flow complexity. Most likely, this will include architectural decisions outside the modeling language itself, like the complexity and diversity of the underlying data models of the orchestrated services. Also, the use of vendor-specific functionality is very apparent. Causes for this should be subject to future research, especially with the aim to provide improvements to relevant standards. One worthwhile angle of future research would be an empirical comparison of the data-flow languages typically used with regard to development effort and understandability.

Although the case study used BPEL processes, executable processes modeled in BPMN2 also need to deal with data transformations. We expect complexity drivers for data transformations to be independent of the process modeling language. However, a further research angle would be the investigation of data flow in BPMN2 processes as well: unfortunately because BPMN2 is not standardized in the area of technical bindings, we expect research to be more difficult and the metrics harder to compare between different process engines than with BPEL.

We hope that we can pursue these possible future research topics on data flow in executable business processes together with other researchers and industry partners and hope that this case study and the published tooling help others to replicate this study and provide answers to data-flow research questions on other interesting topics!

## References

1. V.R. Basili, Applying the goal/question/metric paradigm in the experience factory, in *Software Quality Assurance and Measurement: A Worldwide Perspective*, vol. 7, no.4 (Chapman and Hall, London, 1995), pp. 21–44
2. W. Berli, D. Lübke, W. Möckli, Terravis – large scale business process integration between public and private partners, in *Lecture Notes in Informatics (LNI)*, ed. by E. Plödereder, L. Grunske, E. Schneider, D. Ull. Proceedings INFORMATIK 2014, vol. P-232 (Gesellschaft für Informatik e.V., Bonn, 2014), pp. 1075–1090
3. J. Cardoso, Complexity analysis of BPEL web processes. Softw. Process Improv. Pract. J. **12**, 35–49 (2006)
4. J. Cardoso, Process control-flow complexity metric: an empirical validation, in *IEEE International Conference on Services Computing, 2006. SCC'06* (IEEE, Piscataway, 2006), pp. 167–173
5. J. Cardoso, About the data-flow complexity of web processes, in *6th International Workshop on Business Process Modeling, Development, and Support: Business Processes and Support Systems: Design for Flexibility. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)* (2015)
6. M. Danei, J. Elliott, M. Heiler, T. Kerwien, V. Stiehl, Effectively and efficiently implementing complex business processes - a case study, in *Empirical Studies on the Development of Executable Business Processes*, Chapter 3 (Springer, Cham, 2019)
7. M. Hertis, M.B. Juric, An empirical analysis of business process execution language usage. IEEE Trans. Softw. Eng. **40**(08), 738–757 (2014)
8. D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C.K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu, *Web Services Business Process Execution Language Version 2.0* (OASIS, Clovis, 2007)
9. D. Lübke, Using metric time lines for identifying architecture shortcomings in process execution architectures, in *2nd International Workshop on Software Architecture and Metrics (SAM), 2015 IEEE/ACM* (IEEE, Piscataway, 2015), pp. 55–58
10. D. Lübke, A. Ivanchikj, C. Pautasso, A template for categorizing empirical business process metrics, in *Business Process Management Forum - BPM Forum 2017*, ed. by J. Carmona, G. Engels, A. Kumar (Springer, Cham, 2017)
11. T.J. McCabe, A complexity measure, in *Proceedings of the 2Nd International Conference on Software Engineering, ICSE '76*, Los Alamitos (IEEE Computer Society Press, Washington, 1976), 407 pp.

12. A. Meyer, L. Pufahl, K. Batoulis, D. Fahland, M. Weske, Automating data exchange in process choreographies. Inf. Syst. **53**, 296–329 (2015)
13. S. Moser, A. Martens, K. Gorlach, W. Amme, A. Godlinski, Advanced verification of distributed ws-bpel business processes incorporating cssa-based data flow analysis, in *IEEE International Conference on Services Computing (SCC 2007)*, July 2007, pp. 98–105
14. A. Nikaj, M. Weske, J. Mendling, Semi-automatic derivation of RESTful choreographies from business process choreographies. Softw. Syst. Model. **18**(2), 1195–1208 (2019)
15. R.M. Parizi, A.A.A. Ghani, An ensemble of complexity metrics for BPEL web processes, in *Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08* (2008)
16. P. Runeson, M. Höst, A. Rainer, B. Regnell, *Case Study Research in Software Engineeering – Guidelines and Examples* (Wiley, Hoboken, 2012)
17. A. Slomiski, On using BPEL extensibility to implement OGSI and WSRF grid workflows. Concurr. Comput. Pract. Exp. **18**(10), 1229–1241 (2006)
18. W. Song, C.Z. Zhang, H.-A. Jacobsen, An empirical study on data flow bugs in business processes. IEEE Trans. Cloud Comput. **PP**, 1 (2018)
19. Y. Zheng, J. Zhou, P. Krause, Analysis of BPEL data dependencies, in *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)* Aug 2007, pp. 351–358
20. M. zur Muehlen, J. Recker, How much language is enough? Theoretical and practical use of the business process modeling notation, in *Advanced Information Systems Engineering: 20th International Conference, CAiSE 2008 Montpellier, June 16–20, 2008 Proceedings*, ed. by Z. Bellahsène, M. Léonard (Springer, Berlin, 2008), pp. 465–479