Approach to Generating Functional Test Cases from BPMN Process Diagrams

Pauline von Olberg
Fraunhofer FOKUS
Berlin, Germany
Email: pauline.von.olberg@fokus.fraunhofer.de

Lukas Strey
Fraunhofer FOKUS
Berlin, Germany
Email: lukas.strey@fokus.fraunhofer.de

***Abstract*—Business Process Model and Notation (BPMN) is a popular and widespread modelling language used to describe business processes. These BPMN business process models can serve as a foundation for functional software testing. Functional software testing is an important part of software development, which ensures that software works as expected and that it includes all the desired functionality, as defined in the process models. This position paper presents an approach and considers two different methods on how to automatically create functional test cases from BPMN business process models. The generated test cases shall be understandable for all stakeholders and abstracting from the technical implementation. To achieve this general understandability of the test cases, Gherkin is used as a test case definition language. The two proposed methods will be developed and evaluated in future work. This planned evaluation includes comparing the automatically created test cases with manually created ones.**

***Index Terms*—BPMN, functional software testing, Gherkin, business process modelling**

I. Introduction

In this position paper we explore the generation of functional test cases on the basis of BPMN process diagrams. We aim to give an overview over existing work and propose our own approach alongside two specific methods, which will be developed, tested and evaluated in further work. The following sections give an introduction into the basic concepts used in this paper, followed by the related work and our approach. Lastly, we give an overview over two possible solutions and the planned next steps.

A. BPMN

The Business Process Model and Notation (BPMN) modelling language is a cross-sector de facto standard provided by the Object Management Group (OMG), used for describing business processes [1]. BPMN defines different types of diagrams, among which is the commonly used process diagram, which defines different kinds of business activities that are connected with each other through a control flow. This control flow can additionally be enriched with various types of gateways, forking and joining the flow. Not only will a process diagram modelled with BPMN depict the different steps of a business process, but it will also give some contextual information about the participants involved in the process and, more precisely, which role or unit of an organisation is responsible for fulfilling certain sequences of tasks. Furthermore, a BPMN process diagram offers capabilities to capture message flows between different participants, and there are also options for modellers to make use of multiple types of events in the control flow as well as associations to data objects, annotations and more (see Figure 1). Overall, BPMN allows for semantically rich and precise documentation of business processes, alongside enabling organisations to communicate their processes in a standardised manner.

B. Functional Software Testing

Besides the purpose of communicating and documenting business processes, BPMN diagrams can also function as a starting point to creating functional test cases for the respective software products. Functional test cases specify behavioural scenarios. These can be derived from the sequence flows modelled in the process diagrams.

Functional test cases are a core piece of functional software testing, which is a type of software testing that validates the developed software system against the functional requirements. It ensures that the system works as expected and that it offers all the desired functionalities, which can be extracted from the BPMN process diagrams. Therefore, functional software testing, alongside non-functional software testing, contributes to assuring that the quality criteria of a software product are met [2, 3].

C. Use Case

In the process of modernising the federal budget procedures of Germany, the need to automatically create functional test cases based on existing BPMN process diagrams emerged, which is why we have chosen this project context as a specific use case. The project is an overarching activity that aims to support the federal procedures of the German Ministry of Finance with modernised software solutions. This includes processes like the creation of the federal budget or the planning of personal resources in the federal administration. As such, quality of the developed software products is of upmost importance. The processes that shall be modernised can be clearly defined, and to establish an understanding by

all stakeholders, many of them have already been modelled as BPMN process diagrams. To ensure the necessary quality of the developed software and to facilitate trust in the solution, it shall be extensively tested. This testing shall include the desired functionality modelled with BPMN. Documenting the generated test cases in a universal and abstract language is a target, so that all teams within the project context can use the test cases as a foundation for test-driven development, independently of the teams' technology stack. This allows for automation in the development process. An automated generation that would only need to be verified, would additionally alleviate the workload of the context experts, solving an existing bottleneck in the project.

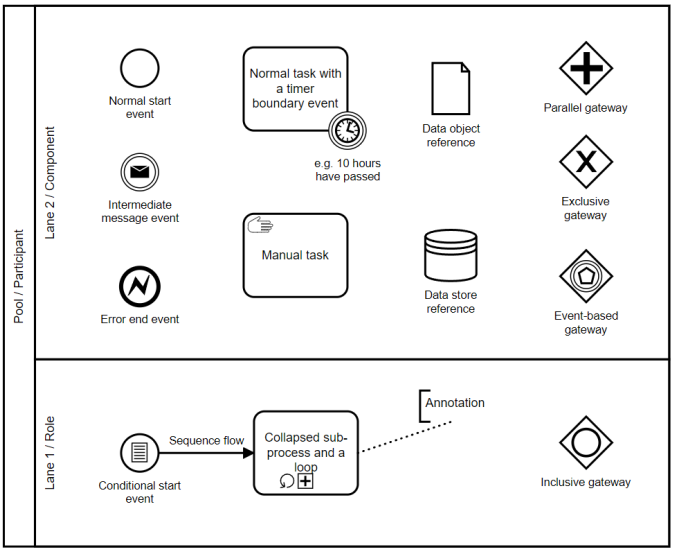

Fig. 1. Subset of BPMN elements

II. Related Work

There has been a multitude of research conducted on the topic of functional test case generation from BPMN process diagrams. Firstly, [4], [5] and [6] present approaches that involve supplementary inputs for the test case generation besides BPMN process diagrams. In [4], authors present a tool for test case generation from a BPMN diagram together with a Business Process Execution Language (BPEL) diagram and an XML Schema Definition. BPEL diagrams are used to explain service behaviours. [5] is a recently published work that presents an approach to generate test cases from BPMN with DMN (Decision Model and Notation). Paths traversing from the entry to the exit of the BPMN model alongside existing test cases and DMN rules are inputs of this method. Whereas [6] focuses on input variables in the BPMN diagram, which have to be manually filled in by the user if the information of the variables that are necessary for creating test cases is incomplete. In [7], the authors focus on an approach for generating test cases from a process diagram, where they first extract business rules from that diagram and then generate test cases based on those rules. Further, the authors state that the generated test sequences are more oriented towards manual testing. Considering that most of the already existing process diagrams in the project context of the modernisation of the federal budget procedures of Germany do not include any business rules, and considering that our goal is to focus on the automated generation of test cases, written in Gherkin, we see the need to extend upon this work. Researchers in [8] use the category partition method (CPM) to generate test cases from business process models. CPM automatically generates test frames based on all possible paths generated from the model, which can then be used to generate the final test cases. The researchers also compared the automatically built test cases with traditionally constructed test cases based on requirements. The outcome was in favour of the automatically generated test cases in terms of time needed to generate test cases, completeness, code coverage and more [8]. Lastly, [9] and [10] describe approaches that are similar to the one we present in this paper, as the output of those methods are test cases written in Gherkin, a language close to the business level, not containing any technical details. [10] suggests a method to generate test cases from BPMN models, for automated testing of web applications implemented with the support of BPM suites. This method consists of (i) identifying execution paths from the flow analysis in the BPMN model and (ii) generating the initial code of test scripts to be run on a given web application testing tool based on Selenium and Cucumber, but only manual tasks that can be performed by users have been evaluated for the test case generation. ETAP-Pro [9] is a test automation platform based on a keyword-driven approach, which is the most comparable to our approach. However, the keyword-driven approach, along with all other presented approaches, still does not cover all relevant information from the BPMN diagrams, leaving several open gaps, which is why we intend to use a more comprehensive method. Questions that are being omitted by the authors of [9] are, for example, how to deal with the various types of loops and events in a process diagram. Further, they did not consider information about the different participants, components or roles of a business process. We therefore aim to promote a more extensive and holistic approach that includes all relevant information regarding the functionality of the business processes and the respective software system.

III. Our Approach

To solve the problems we have highlighted in the introduction, namely generating understandable test cases from existing BPMN diagrams, we aim to extend upon the presented related work. Our goal is to automate the test case generation whilst enabling the verification of these test cases by project stakeholders. The automation shall decrease the workload of the context experts needed to verify the process implementation. Our approach aims to consider and utilise all relevant semantic elements that may appear in the diagrams.

Besides the objective to create test cases that can be understood and verified by stakeholders with the corresponding busi-

BPMN elements and patterns already considered in [9]	BPMN elements and patterns to be additionally considered in our approach
• Activities with no regard to the type of activity • Parallel and exclusive gateways • Start and end events • Non-boundary intermediate events • Separation between pools but no mention of the pools' names	• Loops • Event-based and inclusive gateways • All types of events including Intermediate Boundary Events • Lanes and pools also with regard to their name • Sub-processes • Annotations

Fig. 2. Comparison of elements and patterns considered between Paiva et al. [9] and our suggested approach

ness knowledge, we additionally want to ensure that the test cases are independent of the specific technical implementation of the software system. In the given project context, components are implemented with various programming languages, and we therefore need a notation that maintains independent of these technicalities, so that our approach can be consistently used throughout the entire project.

It is for these reasons that we decided to generate test cases in the form of Gherkin scenarios, just as the authors did in [9, 10]. Gherkin is a logical language that is used to formulate test cases on an abstract level without going into implementation details. It is a domain specific language that uses plain English or other spoken languages if preferred. Gherkin has been developed in connection with Cucumber, which is an open-source tool that supports the automation of test scenarios formulated with the Gherkin language.[1] However, in our approach, we purely focus on the creation of the Gherkin files themselves. A single Gherkin Feature File corresponds to a single process diagram, and it may contain multiple scenarios. A scenario in Gherkin is a list of steps. Each step begins with one of the keywords 'Given', 'When', 'Then', 'But' or 'And' (see Figure 3). Gherkin also defines a concept to group 'Given'-steps together that appear in every scenario of a feature file under a Background section, which helps to manage redundancy. Additionally, there are options to write comments in the test cases, and options to run the same scenario multiple times with different combinations of values.[2]

```
Scenario: Error case: There are errors in the Input Data
Given Input: Process initiated with Input Data
And   Input: "Queue Input Data for processing" to Data Handling Component
And   Data Handling Component: Process initiated and Input Data received
And   Data Handling Component: "Check data for errors"
When  Data Handling Component: "Are there errors in the Input Data?" == Yes
Then  Data Handling Component: "Report errors" to Error Handling Component
And   Data Handling Component: "Report error status" to Input
And   Data Handling Component: Process ended
And   Input: "Error status report received"
And   Input: "Change status"
And   Input: Process ended
```

Fig. 3. First Gherkin scenario example

[1]https://cucumber.io/docs/guides/overview/ (Accessed: 05.07.2022)
[2]https://cucumber.io/docs/gherkin/reference/ (Accessed: 02.06.2022)

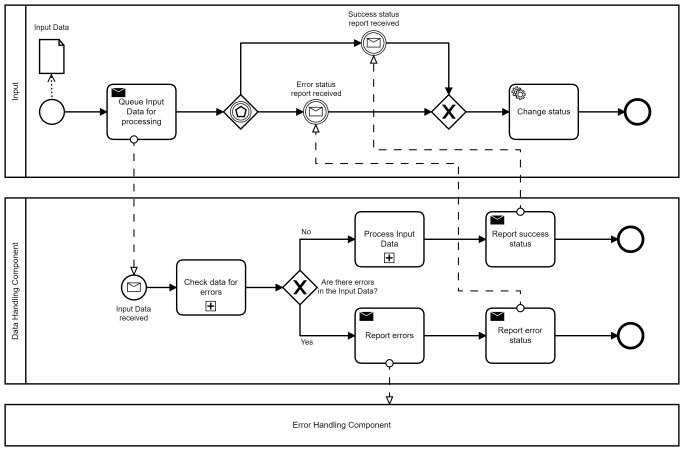

Fig. 4. Example BPMN process diagram

Gherkin offers the advantage that non-technical professionals as well as product owners can understand and approve the resulting test cases swiftly, which reduces the effort for this personnel bottleneck.

Further, our goal is to automate the creation of these test cases based on already existing BPMN diagrams. With regard to that, we propose two different methods for the automatic creation of the desired test cases in the next section of this paper.

As mentioned before, in contrast to [9], our approach aims to consider a wider range of BPMN elements for more comprehensive and unambiguous test cases. Figure 2 displays an overwiev of this.

Figure 4 shows an example process diagram modelled with BPMN. The example is taken out of the presented project context, but it has been considerably simplified and generalised in order to fit into the scope of this position paper and for confidentiality reasons. Figure 3 shows an exemplary Gherkin scenario derived from this diagram. The example shows a basic transformation from BPMN process steps to a Gherkin keyword representation while also including the BPMN pool information. This transformation can serve as a starting point for further development, with the aim to represent more

complex functional scenarios.

IV. Possible Solutions for Future Development

In this position paper, we discuss two general methods that we consider for automatically creating Gherkin test cases from business process diagrams modelled with BPMN.

A. Method 1 - Using an intermediate transformation

The idea behind the first method is to include an intermediate transformation step on the way from BPMN to Gherkin. A transformation of a BPMN diagram to a Petri net, for instance. Petri nets are just one specific option here but another kind of formal model could also be considered. The goal of this intermediate transformation step is to take some complexity out of the semantically rich BPMN model by first transforming it to a more formal and simpler model and then transforming this intermediate model to Gherkin scenarios in a second step.

We have considered Petri nets in particular because there have already been a number of researches conducted on the topic of automatically converting BPMN process diagrams to Petri nets, like [11] and [12], and these approaches already consider a broad range of elements for the transformation. In addition, Petri nets not only help reduce complexity but they also allow for the semantic correctness of BPMN diagrams to be formally validated. This includes checking for deadlocks, livelocks and other inconsistencies [11].

Petri nets further make it easy to work with path and transition coverage, since a Petri net only consists of places and transitions, which allows for straightforward determination of the different paths that tokens may take.

However, there are also issues that need to be considered regarding this intermediate transformation step. Since Petri nets or other formal models are much more simplistic than BPMN, it is a challenging task to ensure that no relevant information is lost in the process. It must be carefully determined which information can be extracted from the BPMN diagrams and how this information can be reduced and translated to the intermediate model. The challenge is to find the right balance between complexity reduction and integrity of the information.

B. Method 2 - Direct transformation

In contrast to the aforementioned method with a formal model as an intermediate step, a different method could be used that enables a direct transformation from BPMN to Gherkin via a pattern matching. The advantage of such a method is that no information is lost through intermediate transformation steps. The entire set of BPMN elements and patterns serves as a base for the pattern matching. Even annotations and information about roles and components from BPMN swim lanes and pools can be included. However, it is a complex task to create a comprehensive and unambiguous pattern matching between all relevant BPMN elements and the rather simple Gherkin syntax.

First, it needs to be exactly determined which information, namely which BPMN elements and patterns, are to be considered as relevant and thus need to be extracted from the diagrams and represented in the test cases. Here, it can be helpful that in practise only a small subset of BPMN elements are commonly used [13]. Secondly, a definite and unambiguous translation from each relevant element or pattern to the elements of the Gherkin syntax has to be decided upon in order to construct a consistent pattern matching. Specifically, the latter will be a challenging task, considering the big discrepancy between the expressive graphical notation of BPMN and the simple plain text notation of Gherkin.

V. Conclusions and Outlook

In this position paper, it became clear that there is a need to automatically generate functional test cases from BPMN process diagrams and a need to extend upon already existing work. The project context of the modernisation of the federal budget procedures of Germany highlights this necessity. In this project, business process diagrams already exist and the automatic generation of test cases, based on these diagrams, in an easily understandable language such as Gherkin, would decrease the workload of the context experts.

```
Scenario: Error case: There are errors in the Input Data
Background Input File A
Given Input File A "Kassenzeichen = INVALID"
When  Input: Process initiated with Input File
And   Data Handling Component: "Check data for errors"
Then  Data Handling Component: "Are there errors in the Input Data?" == Yes
And   Data Handling Component: "Report errors" to Error Handling Component
      includes "Kassenzeichen = INVALID"
```

Fig. 5. Second Gherkin scenario example

We discussed two possible methods in this paper that could be used to generate the desired test cases. A method that includes an intermediate transformation from BPMN to a more formal model to help reduce complexity, and a method that supports a direct conversion from BPMN to Gherkin, using a pattern matching. Both methods are not yet fully matured, which is why we intend to elaborate on them in future research. This involves further developing, testing and evaluating them. The evaluation will be performed within the given project context, which provides the necessary data to perform the tests on. One of the goals for future research work is to also consider how this test data could possibly be integrated into the Gherkin scenarios, and how the scenarios can be further tailored to specifically test that data. An explorative example is shown in Figure 5, which offers an alternative to Figure 3 and includes specific test data.

After all, both in this paper proposed methods will be used to automatically generate test cases written in Gherkin, which will be evaluated by context experts, comparing the automatically generated test cases with manually created ones. Criteria of this evaluation will be understandability, accuracy and test coverage among others. This will serve as a basis for the comparison of the developed methods with the intention to determine wether an intermediate transformation step between BPMN and Gherkin is beneficial or if the loss of information during this intermediate step would be too severe.

REFERENCES

[1] OMG. *Business Process Model and Notation Specification Version 2.0*. Object Management Group, 2011.

[2] Gerald D. Everett and Jr. Raymond McLeod. *Software Testing - Testing Across the Entire Software Development Life Cycle*. IEEE Press and Wiley-Interscience, 2007.

[3] IEEE. "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990, pp. 1-84* (1990).

[4] Chaithep Nonchot and Taratip Suwannasart. "A Tool for Generating Test Case from BPMN Diagram with a BPEL Diagram". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists IMECS 2016* Vol I (2016).

[5] Boodsarin Boonmepipit and Taratip Suwannasart. "Test Case Generation from BPMN with DMN". In: *Proceedings of the 2019 3rd International Conference on Software and e-Business, December 2019, pp. 92-96* (2019).

[6] P. Yotyawilai and T. Suwannasart. "Design of a tool for generating test cases from BPMN". In: *International Conference on Data and Software Engineering (ICODSE), 2014, pp. 1-6* (2014).

[7] S. Sriganesh and C. Ramanathan. "Externalizing business rules from business processes for model based testing". In: *IEEE International Conference on Industrial Technology, 2012, pp. 312-318* (2012).

[8] Sarah Khader and Rana Yousef. "Utilizing Business Process Models to Generate Software Test Cases". In: *IJCSI International Journal of Computer Science Issues, Volume 13, Issue 2, March 2016* (2016).

[9] Ana Paiva et al. "End-to-end Automatic Business Process Validation". In: *Procedia Computer Science, January 2018, volume 130, pp. 999-1004* (2018).

[10] J. C. D. Lima J. L. de Moura A. S. Charão and B. de Oliveira Stein. "Test case generation from BPMN models for automated testing of Web-based BPM applications". In: *17th International Conference on Computational Science and Its Applications (ICCSA)* (2017).

[11] Muhammad Umair Mutarraf et al. "Transformation of Business Process Model and Notation models onto Petri nets and their analysis". In: *Advances in Mechanical Engineering* (2018).

[12] Mohamed Z. Ramadan, Hicham G. Elmongui, and Riham Hassan. "BPMN Formalisation using Coloured Petri Nets". In: *Software Engineering Applications, ACTA* (2011).

[13] M. zur Muehlen und J. Recker. "How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation". In: *Proceedings of the 41st Hawaii International Conference on System Sciences* (2008).