ISSN 2356-5462
http://socj.telkomuniversity.ac.id/ijoict/

Intl. Journal on ICT
Vol. 10, No. 2, Dec 2024. pp. 195-213
doi: 10.21108/ijoict.v10i2.1015

FlowForge: A Prototype for Generating User Stories and Gherkin Test Cases from BPMN with DMN Integration and Pattern Matching

Rosa Reska Riskiana[1*], Ryan Oktaviandi Susilo Wibowo[2], Arpriansah Yonathan[3]

[1,2,3] *School of Computing, Telkom University*
Bandung, Indonesia

[*] rosareskaa@telkomuniversity.ac.id

Abstract

Miscommunication between business stakeholders and developers often leads to inconsistencies in software requirements specifications, creating a gap in understanding that can result in software failure. Behaviour-Driven Development (BDD) aims to address this by fostering collaboration and ensuring a shared understanding through structured natural language, primarily using Gherkin syntax for test case documentation. However, while BDD helps bridge this communication gap, integrating it with Business Process Model and Notation (BPMN) to automate test case generation remains challenging, especially when handling complex pathways and evolving process models. This research addresses these challenges by combining Decision Model and Notation (DMN) with pattern matching techniques and introduces FlowForge, a prototype implemented in this research that automates the generation of User Stories and Gherkin test cases directly from BPMN models. FlowForge demonstrates high completeness in BPMN element extraction, with an average completeness of 98.25%. Path accuracy varied, with an average of 87.5%, and execution time averaged 0.36 seconds, showcasing efficiency. The integration of DMN allows for better handling of exceptions and decision logic, providing fully detailed Gherkin test cases. This research improves the automation and reliability of BPMN-based testing and offers a foundation for future work to enhance accuracy, efficiency, and coverage.

Keywords: BPMN, DMN, User Story, Gherkin, BDD, FlowForge

I. Introduction

AFTER decades of advancements in software development, many software-related accidents can still be traced back to issues in requirements specifications. Software Requirements Specifications (SRS) often rely heavily on the personality, experience, and skills of developers to accurately capture requirements from business stakeholders or non-IT personnel [1]. Miscommunication between business analysts and developers, along with inconsistencies between process modelling and the extracted requirements, remains a persistent issue [2]. An agile methodology that addresses this gap is Behaviour-Driven Development (BDD). BDD aims to enhance communication and foster a shared understanding of the software's purpose among all stakeholders, including developers, product owners, analysts, testers, customers, and business representatives [3], [4], [5].

In the BDD framework, there are three core activities: (1) discovering the required behaviour of the software under development, (2) formulating and documenting this behaviour using structured natural language in the form of user stories and scenarios described in Given/When/Then style known as Gherkin , and (3) automating

Received on 20 Nov 2024. Revised on 20 Dec 2024. Accepted and Published on 10 Jan 2025.

tests for the documented behaviour [6]. In the discovering phase, Requirements Engineering offers various techniques and notations for capturing software requirements, one of which is Business Process Model and Notation (BPMN). BPMN is widely used for modelling complex business processes, providing organizations with a visual representation that streamlines operations, improves efficiency, and ensures workflow consistency [7]. BPMN inherits and combine elements from a number of previously notations for business process modelling, including XML Process Definition Language (XPDL) and Activity Diagrams component of UML [2].

One way to verify that business processes function correctly is through testing. However, while BPMN excels in modelling, one of the critical challenges that remains is the automation of testing these process models. Ensuring that modelled processes behave as expected in real-world scenarios is vital, yet manual testing of these models can be time-consuming and prone to errors. Researchers have investigated automating BPMN testing, with the most significant challenge being test case generation [8]. According to [9] there are two primary approaches for this: the first involves using intermediate transformations with complementary models, as leveraged by [10] and the second employs direct transformation techniques [2], [9], [11], and [12]. While approaches that avoid intermediate models are suitable for rapidly changing process models, as they prevent the creation of outdated representations that do not reflect the latest changes [8], the main drawback of this approach is that most techniques supporting it only handle a limited subset of BPMN elements. This limitation arises from the challenges in determining possible paths within complex models.

A recent study using pattern matching techniques to directly transform BPMN elements into natural-language test cases in User Stories format and Gherkin test cases encountered this issue [2]. One of the challenges of this current transformation from BPMN to User Stories is to find a way to embrace the complex pathways and exceptions present in the models. Additionally, there is a need for a standardized glossary of words and phrases to ensure that the generated Gherkin test cases are readable and reflect the natural flow of the business process. Furthermore, the conceptual model that governs the extraction and transformation of BPMN elements is still under development and needs to ensure scalability and flexibility to handle a variety of process types and complexities. Finally, as BPMN models continue to evolve, it will be necessary to identify and specify new patterns to ensure the transformation process can adapt to emerging process behaviors and structures. Addressing these challenges is essential to fully realizing the potential of automated BPMN-based testing. The study in [9] emphasizes that establishing a comprehensive and unambiguous pattern matching between BPMN elements and Gherkin syntax is a highly complex task. It suggests a structured approach: (1) identifying which BPMN elements and patterns are relevant and should be represented in the test cases, and (2) defining a clear and unambiguous translation of each BPMN element or pattern into corresponding Gherkin syntax to ensure consistent pattern matching.

This paper focuses on addressing the challenges related to complex pathways and exception handling as well as specifying new patterns to ensure that the transformation process can adapt to evolving process structures and behaviors. By enhancing support for various gateway types, this study aims to bridge one of the significant gaps in the current prototype, enabling it to handle more complex BPMN scenarios that are common in real-world applications. Additionally, the specification of new patterns will facilitate a more flexible transformation process that can respond to emerging business process trends. While other challenges, such as developing a standardized glossary for readability and improving the conceptual model for scalability, remain important, these will be reserved for future research to build upon the foundation established in this work.

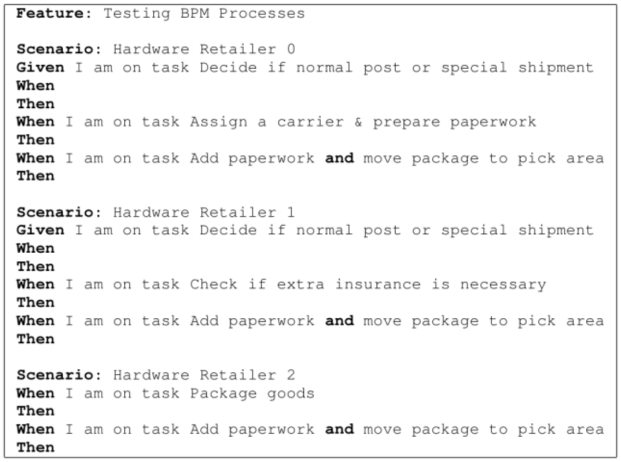
To address the challenges of automating test case generation from BPMN models, integrating Decision Model and Notation (DMN) offers a promising solution. DMN allows for precise modelling of decisions and business rules, particularly for complex decision-heavy workflows where BPMN alone may struggle. A notable solution is presented in [13], which utilizes an integration approach between BPMN and DMN to reveal all potential end-to-end execution paths of a business process. This approach results in test case specifications in a domain-specific language (DSL). A study [14] demonstrated that incorporating DMN enhances the accuracy and completeness of test cases by explicitly capturing decision logic. This integration ensures that User Stories and Gherkin test cases are generated with clearer mapping to decision points, improving the handling of gateways, exceptions, and conditional flows while ensuring comprehensive test coverage. As [9][14] has yet to be implemented with a generating tool, we present FlowForge, a tool designed to assist users in automatically generating User Stories and Gherkin test cases directly from BPMN models. FlowForge integrates DMN and pattern-matching techniques to tackle the challenges associated with complex pathways and exception handling. This research offers several significant benefits. Enhancing support for various gateway types will enable the transformation process to handle a wider range of BPMN scenarios, reflecting the complexity and variability of real-world applications. The specification of new patterns ensures the transformation process is adaptable to evolving process structures and behaviours. These improvements will not only bridge critical gaps in existing prototypes but also refine the generation of test cases. As a result, the proposed approach contributes to advancing automation capabilities, reducing manual effort, and increasing the reliability of business process testing in dynamic environments.

II. LITERATURE REVIEW

A comprehensive literature review was carried out to examine the latest approaches and methodologies for automating the generation of test cases in Gherkin format from BPMN process diagrams. The review aimed to identify research gaps and limitations while also providing readers with an understanding of BPMN.

A. Related Work

Current research on BPMN-based test case generation encounters several fundamental challenges that limit its effectiveness in practical applications. De Moura et al. introduced a two-step methodology for end-to-end test case generation. This process begins with traversing the XML Process Definition Language (XPDL) representation of a BPMN model to produce an Excel file that documents all process flows. In this file, columns represent specific tasks, while rows correspond to possible flows, with tasks included in each flow marked by an "X." Test scripts are then automatically generated based on the information in this table [11]. While this method successfully produces Gherkin-based test scenarios as shown in Fig. 1, it has certain limitations. The step definitions generated are incomplete, particularly in the “Then” component of the Given-When-Then format of Gherkin scenarios which specifies the expected outcome corresponding to a particular Given and When condition. While the authors do not provide detailed explanations of the Gherkin generation process, it can be inferred that the extracted process flows, presented as tables, lack clarity in capturing outcomes. This limitation results in Gherkin scenarios that primarily address preconditions (Given) and current states (When), without adequately detailing expected results (Then). Another drawback is the treatment of parallel paths as separate flows, which fails to accurately represent real-world scenarios and provides no differentiation from conditional paths. Limitations of this approach are that DMN are not taken into account [13].

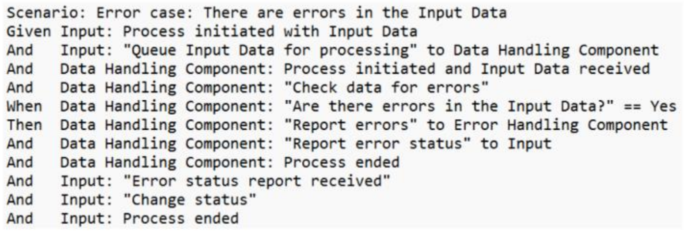


```
Feature: Testing BPM Processes

Scenario: Hardware Retailer 0
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Assign a carrier & prepare paperwork
Then
When I am on task Add paperwork and move package to pick area
Then

Scenario: Hardware Retailer 1
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Check if extra insurance is necessary
Then
When I am on task Add paperwork and move package to pick area
Then

Scenario: Hardware Retailer 2
When I am on task Package goods
Then
When I am on task Add paperwork and move package to pick area
Then
```

Fig. 1. Test Scenarios Generated [11]

Paiva et al. addressed some of these limitations in their work by introducing a tool called ETAP-Pro (End-to-end Test Automation Platform for Processes), which employs a keyword-driven approach. ETAP-Pro employs Depth-First Search (DFS), similar to the approach used by [13], to generate all possible paths within a BPMN model represented in XPDL format. Unlike earlier methods, ETAP-Pro considers all possible activity orders in parallel branches, rather than treating them as separate process flows. Furthermore, it generates complete Gherkin scenarios with the Given-When-Then structure. However, this approach also has shortcomings, particularly its inability to capture role-responsibilities, which are integral to BPMN models. The keyword-driven approach outlines test cases or processes (associated with different paths in the model), activities (which represent the actions to be performed), and technical methods (reusable methods that support these activities). However, it does not account for pools to extract role-specific responsibilities, which are essential for accurately modelling BPMN-based workflows [12].

Von Olberg and Strey [9] propose generating clear and functional test cases from BPMN models using Gherkin as the test case definition language. They introduce two approaches for achieving this clarity: an intermediate transformation and a direct transformation. Additionally, they provide examples of test scenarios generated using these two approaches, as illustrated in Fig. 2. Their examples demonstrate the ability to produce complete Gherkin scenarios, including information about BPMN pools that correspond to the roles performing specific activities. However, based on our research, no prototypes of this approach have been implemented so far.



```
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. 2. Test scenarios generated [9]

Seqerloo et al [10] propose a distinct approach by first transforming BPMN into a state graph, where states represent gateways and transitions denote paths. Parallel blocks are represented as a single state, enhancing the Depth-First Search (DFS) algorithm used to traverse the graph and generate paths. These paths are then mapped to test cases based on specific rules. However, a limitation of this approach is that some generated paths result in false positives—instances where the generated paths do not accurately correspond to real-world scenarios, potentially leading to incorrect or redundant test cases. In contrast, a recent study by Mateus et al. [2] introduces a semi-automatic method for generating user stories and Gherkin scenarios directly from process models. This approach utilizes transformation patterns, as illustrated in Fig. 5, which prior research [9] suggests can mitigate information loss typically associated with intermediate transformation steps, such as those in [10]. The study offers a clearer transformation process from BPMN to Gherkin by first generating user stories in natural language, enabling better comprehension of the resulting Gherkin scenarios. This clarity helps bridge the understanding gap between business stakeholders and developers, a feature lacking in previous approaches. However, the resulting user stories primarily focus on the natural flow of the model while neglecting exception scenarios, leading to incomplete Gherkin test cases.

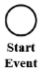
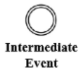
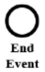
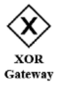
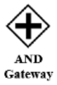
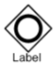
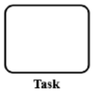
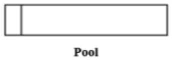
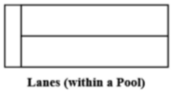
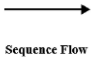
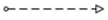
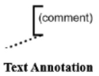
This research closely aligns with our work, which aims to generate User Stories and Gherkin scripts through pattern matching. Our contribution will involve refining the transformation process by introducing new patterns and leveraging DMN to address both the natural flow and exception scenarios, ensuring the completeness of the test generated. A generating tool named FlowForge will be developed to automate this approach to ensure that each BPMN element is accurately mapped to corresponding test scenarios in a language that is both technical and accessible to business stakeholders.

B. FlowForge

FlowForge is a tool that automates the generation of User Stories and Gherkin test cases from BPMN models, simplifying the testing of business processes. By integrating DMN and utilizing pattern-matching techniques, FlowForge aim to ensure comprehensive test case generation for complex pathways, enhancing the completeness of testing. The tool reduces manual effort by transforming BPMN models into test cases, while its DMN integration supports complex decision logic, ensuring thorough coverage of business rules and decision points. Additionally, FlowForge aims to:

1. Improve Automation: Automating the generation of test cases for BPMN models reduces the risk of human error, enhances efficiency, and ensures comprehensive coverage of all potential execution paths.
2. Support Complex Decision Logic: By integrating DMN, FlowForge ensures that decision-heavy workflows are accurately represented.

TABLE I
BPMN BASIC ELEMENTS [15]

Notation	Element Type	Description
	Start Event	Shows the starting point of a particular process, without any incoming Sequence Flows. A circle with an open center represents different types of trigger events, such as message, timer, condition, etc.
	Intermediate Event	Indicates where an event occurs within the process, between the start and end. It highlights where Messages are expected or sent, where delays happen, and where exceptions disrupt the normal flow through exception handling.
	End Event	Indicates the endpoint of a process, with no outgoing Sequence Flows.
	Exclusive (XOR) Gateway	Used to create alternative paths within a process flow, where only one path can be taken. Each path is associated with a condition expression linked to the outgoing Sequence Flows of the Gateway.
	Parallel (AND) Gateway	Used to create parallel paths that are executed simultaneously.
	Inclusive Gateway	Used to create both alternative and parallel paths within a process flow. All condition expressions must be evaluated, but the design should ensure that at least one path is followed.
	Task	An atomic activity within a process flow. A Task is used when the work in the process cannot be further decomposed into smaller tasks.
	Pool	A graphical representation of a participant. A participant can represent a specific entity (e.g., a company) or a more general role (e.g., buyer, seller, etc.).
	Lanes (within a pool)	A Lane is a sub-partition within a process, typically located within a pool.
	Sequence Flow	Used to indicate the order of Flow Elements in a process. Each Sequence Flow has a single source and a single target. Sequence Flow inherits attributes like sourceRef and targetRef to define the incoming and outgoing flow.
	Message Flow	Used to represent the flow of messages between two participants that are ready to send and receive them. These flows must not connect objects within the same pool.
	Annotation	A mechanism that allows the modeler to add supplementary text information for the reader of a BPMN diagram.

C. Business Process Model and Notation (BPMN)

BPMN is a graphical language used to model and execute business process models [15] provided by the Object Management Group (OMG). Its primary goal is to provide a notation that is easily understood by various stakeholders, including business analysts, technical developers, and business professionals. Table I illustrates a visual representation of some key BPMN elements.

Events are used to initiate and conclude process instances, representing changes in behavior, such as `StartEvent`, `IntermediateEvent`, and `EndEvent` while `Activities` denote points in the process flow where work occurs, such as atomic tasks or sub-processes that encapsulate other processes. `ExclusiveGateways` model alternative paths. It illustrates branching points for the subsequent process flow [16], while `ParallelGateways` representing simultaneous paths within the workflow. `InclusiveGateways` on the other hand, model alternative paths within a process flow, allowing one or more paths to be taken based on specific conditions. It illustrates branching points where different paths can be chosen depending on the evaluation of conditions. Because parallel branches execute concurrently, their order of execution cannot be predetermined statically [7]. As a result, the sequence of execution is inconsequential, allowing the branches to be considered sequentially in any arbitrary order [13]. These elements are interconnected using `SequenceFlows` and are organized within `Pools` and `Lanes`, which can include text `Annotations` to provide additional process details. Additionally, BPMN process diagrams support capturing `MessageFlows` between different participants [15]. The structure of a BPMN diagram is stored in XML format, ensuring compatibility and data exchange [14].

III. RESEARCH METHOD

The objective of this research is to develop a prototype capable of converting a BPMN into a Gherkin test case. This conversion aims to bridge the gap between process modeling and behavior-driven development by providing a seamless transition between process workflows and executable specifications. The approach utilized in this study is the direct transformation integrating DMN and pattern matching, designed to maintain the logical

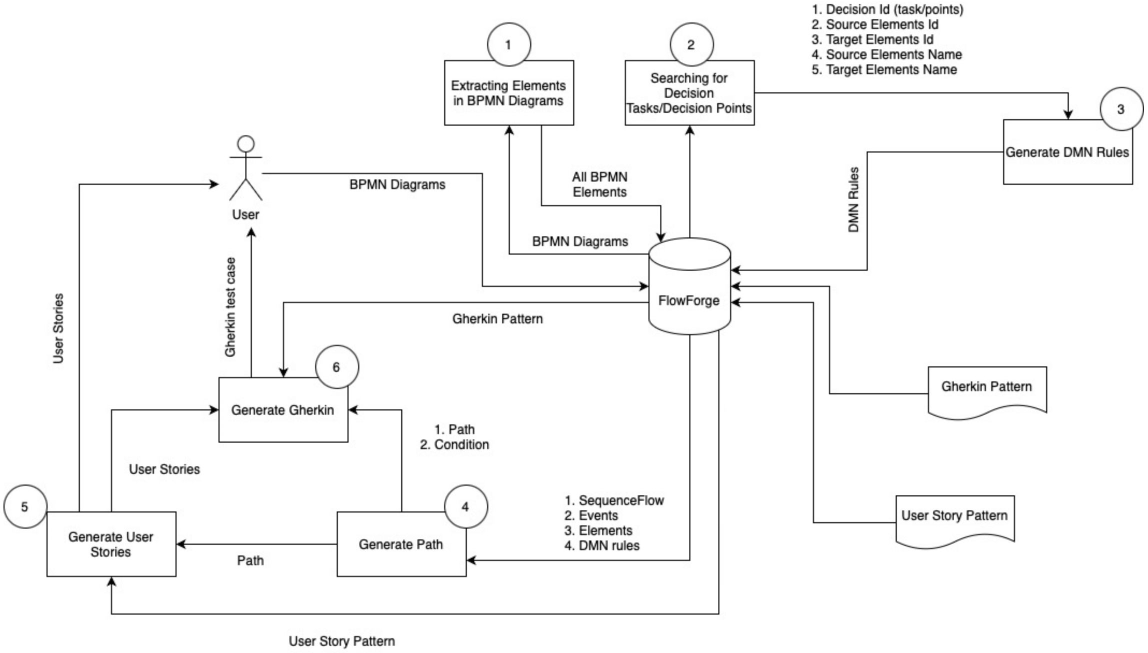


Fig. 3. Approach for Test Case Generation from BPMN to Gherkin

structure of the BPMN elements while translating them into Gherkin syntax. Fig. 3 provides a detailed depiction of how FlowForge achieves its goal through 6 steps process, which has been adapted from [14] and [17].

A. Extracting Elements in BPMN diagrams

The process begins with the identification of key elements within the BPMN diagram, such as events, gateways, tasks, and sequence flows. It is important to detect BPMN decision tasks that will be used to determine which task or subprocess will be executed. By analyzing the interconnections between these elements, the process maps out the pathways through which decisions are made, along with the associated conditions and outcomes. Table II provides an example of the extracted elements by FlowForge derived from BPMN Credit Scoring Asynchronous shown by Fig. 4. All BPMN diagrams used in this research are sourced from a GitHub repository maintained by Camunda, presented for research purposes. Camunda is a software company providing an open-source platform for business process management (BPM). The reason for choosing these BPMN diagrams is that they contain all the key elements mentioned earlier, such as events, all type of gateways, tasks, and sequence flows. Additionally, all complete BPMN diagrams available in the repository have been utilized, ensuring comprehensive coverage of process scenarios.

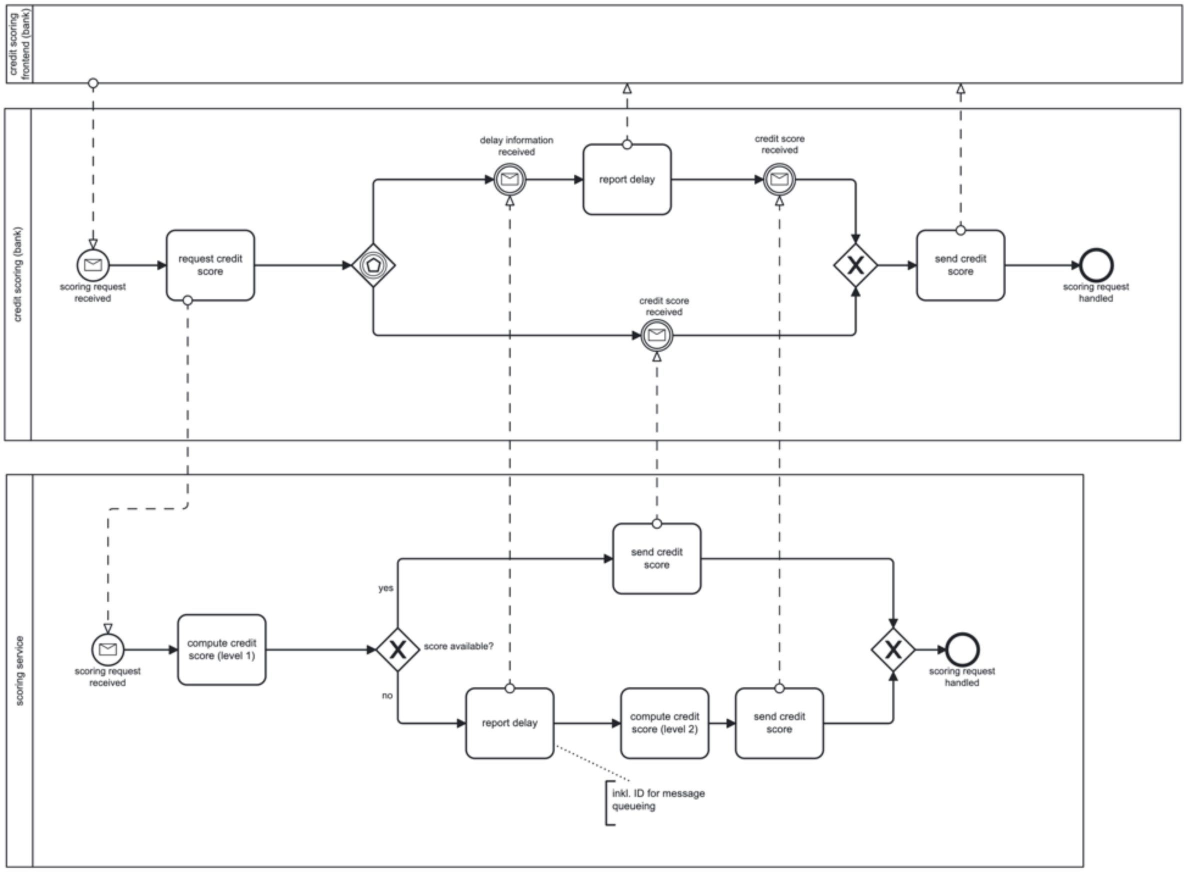


Fig. 4. BPMN Example of “Credit Scoring Asynchronous Process”

In Table II, all elements, including their Type, Name, and ID, are systematically mapped by FlowForge from the BPMN. While multiple elements may share the same name, their unique IDs serve as the primary identifiers. This distinction underscores the importance of utilizing IDs, rather than names, as the key reference for subsequent processes to ensure accuracy and avoid ambiguity.

TABLE II
EXAMPLE EXTRACTED BPMN ELEMENTS OF "CREDIT SCORING ASYNCHRONOUS"

Element Type	Element Name	Element Id
Start Event	scoring request received	StartEvent_0o849un
Task	request credit score	Task_16winvj
Event based gateways	-	EventBasedGateway_02s95tm
Message Event	delay information received	IntermediateCatchEvent_0ujob24
Task	report delay	Task_0l942o9
Message Event	credit score received	IntermediateCatchEvent_0yg7cuh
Exclusive Gateway Merging	-	ExclusiveGateway_125lzox
Task	send credit score	Task_1fzfxey
End Event	scoring request handled	EndEvent_0rp5trg
Start Event	scoring request received	EndEvent_0khk0tq
Task	compute credit score (level 1)	Task_1r15hqs
Exclusive Gateway	score available?	ExclusiveGateway_0rtdod4
Sequence Flow	Yes	SequenceFlow_052bcer
Sequence Flow	No	SequenceFlow_0jh32vv
Task	send credit score	Task_06dqs9t
Message Event	credit score received	IntermediateCatchEvent_0a8iz14
Task	report delay	Task_01ouvha
Task	compute credit score (level 2)	Task_02m68xj
Task	send credit score	Task_07vbn2i
Exclusive Gateway Merging	-	ExclusiveGateway_11dldcm
End Event	scoring request handled	EndEvent_0khk0tq

B. *Searching for Decision Tasks/Decision Points*

After identifying all the elements in a BPMN, FlowForge will determine which elements or tasks are required to develop DMN rules. This involves identifying tasks or elements associated with specific conditions, focusing on decision-making tasks. These are typically represented by decision points such as `ExclusiveGateways` and `InclusiveGateways`, which guide the flow based on predefined conditions.

C. *Generate DMN Rules*

After detecting the decision tasks, FlowForge create specific DMN rules based on these decision points. For each task and gateway extracted on the previous step, the incoming and outgoing flows, along with all associated conditions, are analyzed. Subsequently, any branching points with defined conditions are identified to establish the corresponding DMN rules. This involves defining the decision logic that governs how each decision task operates within the business process. For instance, in the decision task "Compute Credit Score", we must check whether the score is available or not before continuing to the next task. Table III provides an example of DMN rules generated by FlowForge from the "Compute Credit Score" task. This table illustrates the decision logic and criteria used to evaluate credit scores systematically. In the decision task "Compute Credit Score," an `ExclusiveGateway` represents a condition that determines whether the credit score is available. Each `SequenceFlow` from this gateway activates a specific task, leading to distinct `IntermediateEvent` triggered by message catch. Two tasks depend on the outcome of this decision: if the score is available, the "Send Credit Score" task is executed; if the score is unavailable, the "Report Delay" task is performed. These decision rules are typically organized into DMN decision tables, such as the DMN Decision Table for "Compute

TABLE III
DMN RULE OF "COMPUTE CREDIT SCORE (LEVEL 1)"

Task	SourceRef	Sequence Flow	TargetRef
Compute Credit Score (Level 1)	ExclusiveGateway	Yes	Send credit score
Compute Credit Score (Level 1)	ExclusiveGateway	No	Report delay

Credit Score," providing a structured and consistent framework for decision-making based on varying conditions.

D. Generate Path

The process of generating test paths from DMN rules and BPMN files follows a methodical approach that combines graph traversal techniques with data filtering. Initially, DMN rules are transformed into a DataFrame, encapsulating the decision logic that guides the path selection. Simultaneously, the BPMN file is parsed to extract key components such as sequence flows, start and end events, and node names, creating a graph-like structure where nodes represent tasks, gateways, and events. To explore all potential paths from start to end events, a depth-first search (DFS) algorithm is employed, efficiently traversing the graph and recording each route while ensuring that cycles are avoided [12][13]. Afterward, the generated paths are filtered through the DMN rules, selecting those that encompass relevant decision points and their corresponding outcomes. Finally, the selected test paths, along with detailed descriptions, are exported to an Excel file for documentation and analysis. This structured methodology ensures that the generated test paths are both comprehensive and aligned with business logic, facilitating robust testing and validation of business processes. Table IV provides an example of the generated test paths, showcasing each element along with its unique identifier.

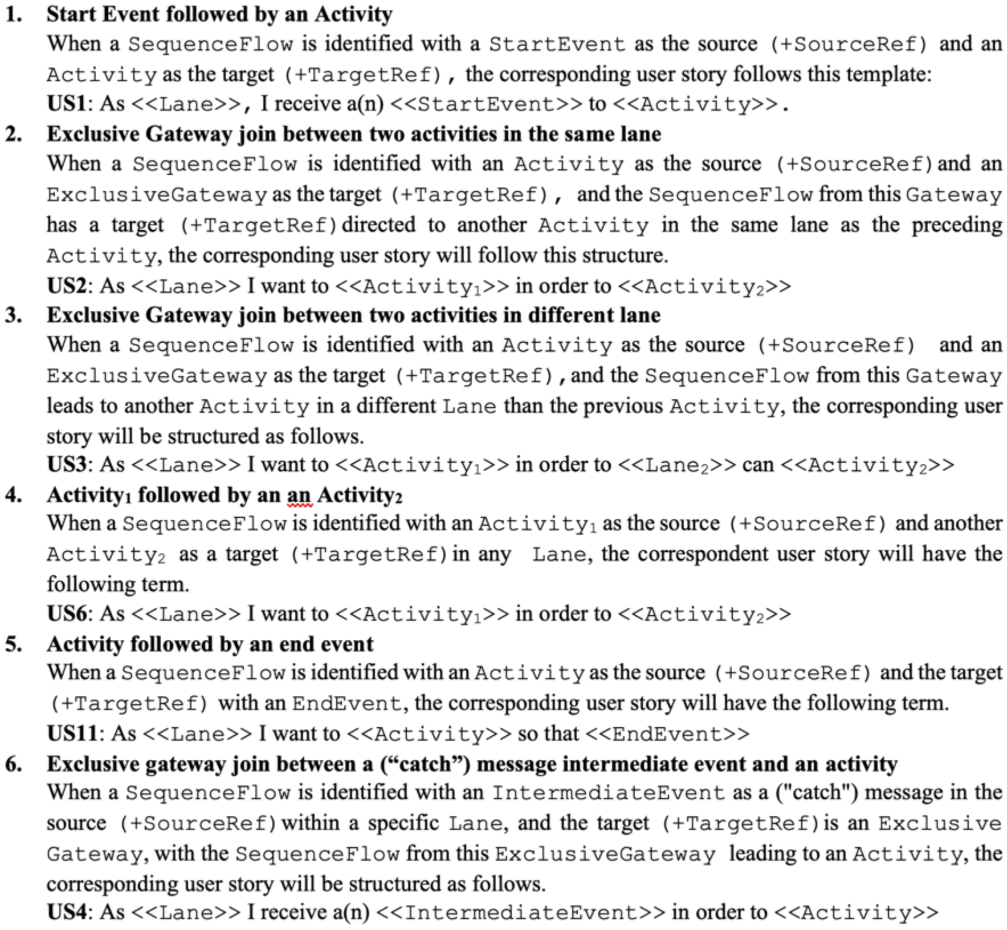
TABLE IV
GENERATED PATH OF "CREDIT SCORING ASYNCHRONOUS"

Path Id	Path Description
1	scoring request received (StartEvent_0o849un), compute credit score (level 1) (Task_1r15hqs), score available? (ExclusiveGateway_0rtdod4), send credit score (Task_07vbn2i), ExclusiveGateway_125lzox, scoring request handled (EndEvent_0khk0tq)
2	scoring request received (StartEvent_0o849un), compute credit score (level 1) (Task_1r15hqs), score available? (ExclusiveGateway_0rtdod4), report delay (Task_01ouvha), compute credit score (level 2) (Task_02m68xj), send credit score (Task_06dqs9t), ExclusiveGateway_125lzox, scoring request handled (EndEvent_0khk0tq)
3	scoring request received (StartEvent_1els7eb), request credit score (Task_16winvj), EventBasedGateway_02s95tm (EventBasedGateway_02s95tm), credit score received (IntermediateCatchEvent_0yg7cuh), ExclusiveGateway_11dldcm (ExclusiveGateway_11dldcm), send credit score (Task_1fzfxey), scoring request handled (EndEvent_0rp5trg)
4	scoring request received (StartEvent_1els7eb), request credit score (Task_16winvj), EventBasedGateway_02s95tm, delay information received (IntermediateCatchEvent_0ujob24), report delay (Task_0l942o9), credit score received (IntermediateCatchEvent_0a8iz14), ExclusiveGateway_11dldcm, send credit score (Task_1fzfxey), scoring request handled (EndEvent_0rp5trg)

E. Generate User Stories

The process of generating user stories from test paths and BPMN files involves a systematic approach to extract relevant information and construct narratives that reflect the user's interaction with a process. Initially, the BPMN file is parsed to identify lanes, which represent different user roles, and to associate tasks and gateways with their corresponding names. This mapping facilitates the understanding of which user is responsible for each task. Concurrently, the test paths, represented in JSON format, are loaded, providing a structured overview of the various routes through the process, each associated with a specific description.

Next, the user stories are crafted by iterating through each path in the JSON data. For each step in the path, the corresponding role, task or gateway name, and any conditions are retrieved from the earlier mappings. This information is then woven into user stories described in the BDD context. We will use the template for specifying user stories, shown as follows [18]: "As `<type of user>`, I want to `<desired goal>` so that `<achieved value>`." This narrative style not only clarifies the user's perspective but also highlights the specific tasks they are expected to perform under varying conditions. Finally, the generated user stories are saved to a text file, providing a comprehensive documentation of user stories that can be utilized for testing,



1. **Start Event followed by an Activity**
   When a SequenceFlow is identified with a StartEvent as the source (+SourceRef) and an Activity as the target (+TargetRef), the corresponding user story follows this template:
   US1: As <<Lane>>, I receive a(n) <<StartEvent>> to <<Activity>>.
2. **Exclusive Gateway join between two activities in the same lane**
   When a SequenceFlow is identified with an Activity as the source (+SourceRef) and an ExclusiveGateway as the target (+TargetRef), and the SequenceFlow from this Gateway has a target (+TargetRef) directed to another Activity in the same lane as the preceding Activity, the corresponding user story will follow this structure.
   US2: As <<Lane>> I want to <<Activity$_1$>> in order to <<Activity$_2$>>
3. **Exclusive Gateway join between two activities in different lane**
   When a SequenceFlow is identified with an Activity as the source (+SourceRef) and an ExclusiveGateway as the target (+TargetRef), and the SequenceFlow from this Gateway leads to another Activity in a different Lane than the previous Activity, the corresponding user story will be structured as follows.
   US3: As <<Lane>> I want to <<Activity$_1$>> in order to <<Lane$_2$>> can <<Activity$_2$>>
4. **Activity$_1$ followed by an an Activity$_2$**
   When a SequenceFlow is identified with an Activity$_1$ as the source (+SourceRef) and another Activity$_2$ as a target (+TargetRef) in any Lane, the correspondent user story will have the following term.
   US6: As <<Lane>> I want to <<Activity$_1$>> in order to <<Activity$_2$>>
5. **Activity followed by an end event**
   When a SequenceFlow is identified with an Activity as the source (+SourceRef) and the target (+TargetRef) with an EndEvent, the corresponding user story will have the following term.
   US11: As <<Lane>> I want to <<Activity>> so that <<EndEvent>>
6. **Exclusive gateway join between a ("catch") message intermediate event and an activity**
   When a SequenceFlow is identified with an IntermediateEvent as a ("catch") message in the source (+SourceRef) within a specific Lane, and the target (+TargetRef) is an Exclusive Gateway, with the SequenceFlow from this ExclusiveGateway leading to an Activity, the corresponding user story will be structured as follows.
   US4: As <<Lane>> I receive a(n) <<IntermediateEvent>> in order to <<Activity>>

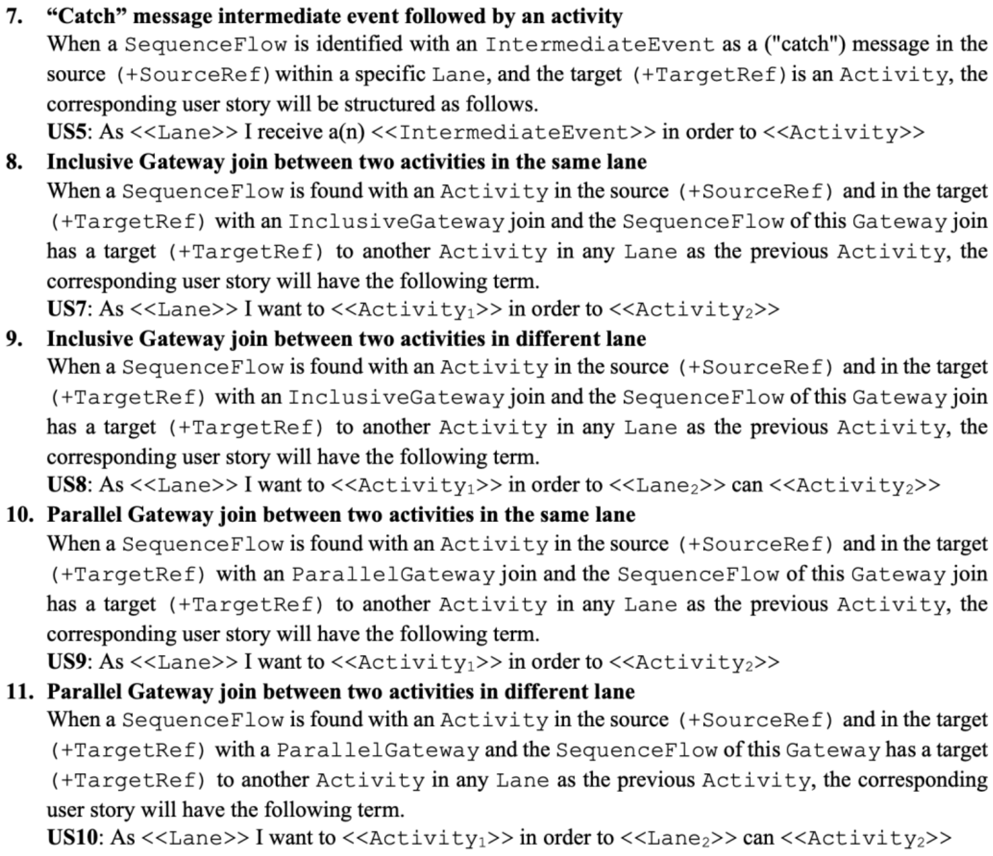
Fig. 5. User Stories Pattern by Mateus et al. [2]

validation, or stakeholder communication. This method ensures that the generated user stories are directly aligned with both the BPMN model and the defined test paths, enhancing the relevance and usability of the resulting documentation.

To convert the generated paths into user stories, we utilized certain patterns from prior research [2] as outlined in Fig. 5, which identified patterns through an analysis of various business models, resulting in a non-exhaustive pattern list. Moreover, we made additional patterns to align them with the structure of our BPMN, as outlined in Fig 6.

F. Generate Gherkin

The process of generating Gherkin test cases from user stories begins by analyzing narrative user stories to extract key components such as the user role, action, and conditions. These components are then transformed into USC (User Stories with Condition) rules, which include IDs and names for tasks along with conditions or SequenceFlows. USC rules are stored in a JSON file and used to generate the User Stories with Conditions. Alongside, MR (Message Rules) are developed to account for IntermediateEvent elements, ensuring the test cases align with the predefined conditions. These MR rules include the ID of the event and associated tasks, similar to USC rules.



7. **"Catch" message intermediate event followed by an activity**
When a `SequenceFlow` is identified with an `IntermediateEvent` as a ("catch") message in the source `(+SourceRef)` within a specific `Lane`, and the target `(+TargetRef)` is an `Activity`, the corresponding user story will be structured as follows.
US5: As <<Lane>> I receive a(n) <<IntermediateEvent>> in order to <<Activity>>

8. **Inclusive Gateway join between two activities in the same lane**
When a `SequenceFlow` is found with an `Activity` in the source `(+SourceRef)` and in the target `(+TargetRef)` with an `InclusiveGateway` join and the `SequenceFlow` of this `Gateway` join has a target `(+TargetRef)` to another `Activity` in any `Lane` as the previous `Activity`, the corresponding user story will have the following term.
US7: As <<Lane>> I want to <<Activity$_1$>> in order to <<Activity$_2$>>

9. **Inclusive Gateway join between two activities in different lane**
When a `SequenceFlow` is found with an `Activity` in the source `(+SourceRef)` and in the target `(+TargetRef)` with an `InclusiveGateway` join and the `SequenceFlow` of this `Gateway` join has a target `(+TargetRef)` to another `Activity` in any `Lane` as the previous `Activity`, the corresponding user story will have the following term.
US8: As <<Lane>> I want to <<Activity$_1$>> in order to <<Lane$_2$>> can <<Activity$_2$>>

10. **Parallel Gateway join between two activities in the same lane**
When a `SequenceFlow` is found with an `Activity` in the source `(+SourceRef)` and in the target `(+TargetRef)` with an `ParallelGateway` join and the `SequenceFlow` of this `Gateway` join has a target `(+TargetRef)` to another `Activity` in any `Lane` as the previous `Activity`, the corresponding user story will have the following term.
US9: As <<Lane>> I want to <<Activity$_1$>> in order to <<Activity$_2$>>

11. **Parallel Gateway join between two activities in different lane**
When a `SequenceFlow` is found with an `Activity` in the source `(+SourceRef)` and in the target `(+TargetRef)` with a `ParallelGateway` and the `SequenceFlow` of this `Gateway` has a target `(+TargetRef)` to another `Activity` in any `Lane` as the previous `Activity`, the corresponding user story will have the following term.
US10: As <<Lane>> I want to <<Activity$_1$>> in order to <<Lane$_2$>> can <<Activity$_2$>>

Fig. 6. Additional User Stories Pattern

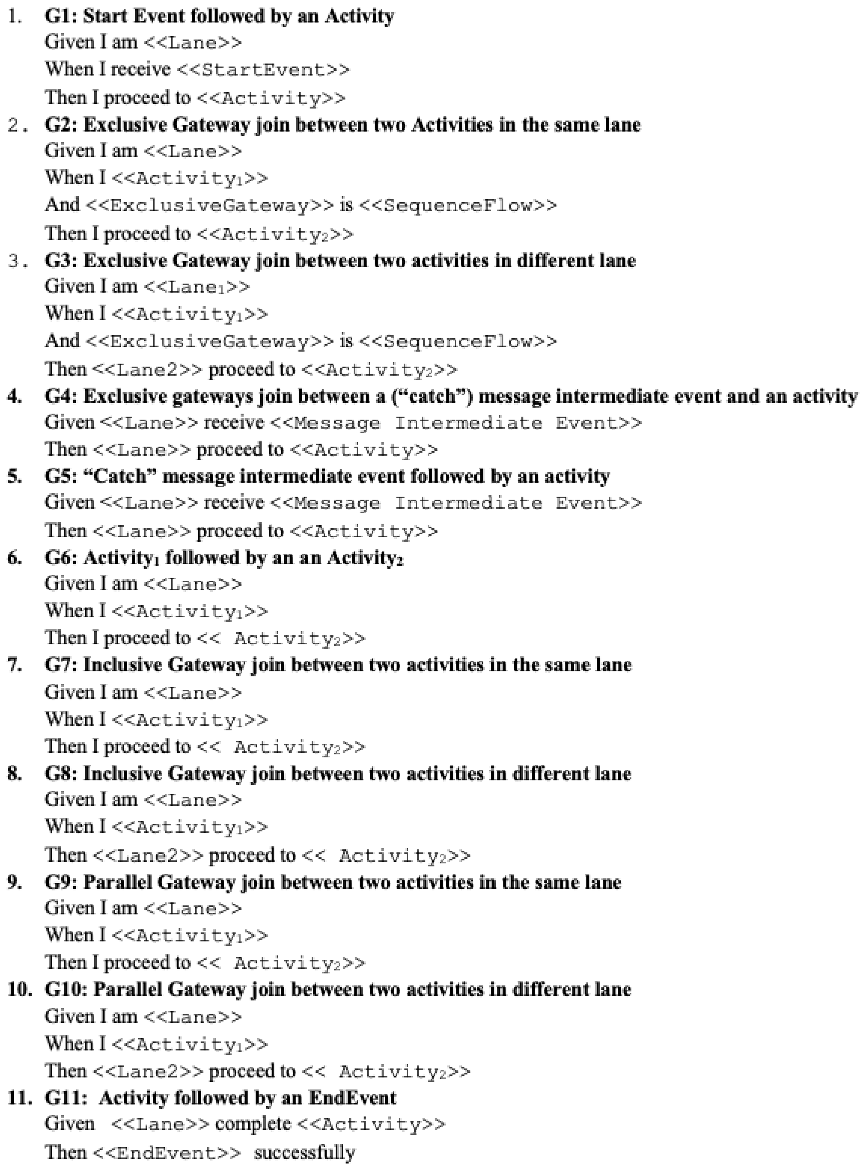
Once the USC and MR rules are defined, the Gherkin syntax is used to convert the user story into test cases. Gherkin utilizes a straightforward syntax that is easy to learn and consists of only a limited number of operators. There are the following Gherkin operators [1]:

1. `FEATURE`: represents a specific functionality of the software.
2. `SCENARIO`: identifies a sequence of events that implement a portion of the `FEATURE`
3. `GIVEN`: specifies the preconditions or initial state required for the `SCENARIO`
4. `WHEN`: describes the actions or events that occur during the execution or testing process
5. `AND`: acts as a logical connector used to link multiple elements of complex preconditions, actions, or outcomes.
6. `BUT`: functions similarly to AND but is used to express negative conditions

The resulting Gherkin test cases preserve the original intent of the scenarios while adhering to Behavior-Driven Development (BDD) conventions. This ensures clarity for both technical and non-technical stakeholders. The test cases are stored in a dedicated file for use in automated testing, ensuring alignment with the defined user requirements. Fig. 7 depicts the pattern developed in this research to generate the Gherkin test cases.

IV. RESULTS AND DISCUSSION

This section presents four distinct evaluations. First, we compare our Gherkin test case generation approach with that of other relevant studies, focusing on its ability to capture key elements such as Given-When-Then



1. **G1: Start Event followed by an Activity**
   Given I am <<Lane>>
   When I receive <<StartEvent>>
   Then I proceed to <<Activity>>
2. **G2: Exclusive Gateway join between two Activities in the same lane**
   Given I am <<Lane>>
   When I <<Activity$_1$>>
   And <<ExclusiveGateway>> is <<SequenceFlow>>
   Then I proceed to <<Activity$_2$>>
3. **G3: Exclusive Gateway join between two activities in different lane**
   Given I am <<Lane$_1$>>
   When I <<Activity$_1$>>
   And <<ExclusiveGateway>> is <<SequenceFlow>>
   Then <<Lane2>> proceed to <<Activity$_2$>>
4. **G4: Exclusive gateways join between a ("catch") message intermediate event and an activity**
   Given <<Lane>> receive <<Message Intermediate Event>>
   Then <<Lane>> proceed to <<Activity>>
5. **G5: "Catch" message intermediate event followed by an activity**
   Given <<Lane>> receive <<Message Intermediate Event>>
   Then <<Lane>> proceed to <<Activity>>
6. **G6: Activity$_1$ followed by an an Activity$_2$**
   Given I am <<Lane>>
   When I <<Activity$_1$>>
   Then I proceed to << Activity$_2$>>
7. **G7: Inclusive Gateway join between two activities in the same lane**
   Given I am <<Lane>>
   When I <<Activity$_1$>>
   Then I proceed to << Activity$_2$>>
8. **G8: Inclusive Gateway join between two activities in different lane**
   Given I am <<Lane>>
   When I <<Activity$_1$>>
   Then <<Lane2>> proceed to << Activity$_2$>>
9. **G9: Parallel Gateway join between two activities in the same lane**
   Given I am <<Lane>>
   When I <<Activity$_1$>>
   Then I proceed to << Activity$_2$>>
10. **G10: Parallel Gateway join between two activities in different lane**
    Given I am <<Lane>>
    When I <<Activity$_1$>>
    Then <<Lane2>> proceed to << Activity$_2$>>
11. **G11: Activity followed by an EndEvent**
    Given <<Lane>> complete <<Activity>>
    Then <<EndEvent>> successfully

Fig. 7. Gherkin Pattern

constructs, events, tasks, lane, gateways, and sequence flows. Second, we assess the generated test cases by evaluating both the completeness of the elements and the accuracy of the extracted paths. Element completeness ensures that all BPMN elements are captured, while path accuracy verifies that the extracted paths align with those present in the BPMN model, ensuring comprehensive test coverage. Lastly, we evaluate the execution time to analyze the efficiency of the approach, measuring the time taken from inputting the BPMN model to generating the Gherkin scenarios.

This study utilizes four BPMN diagrams: Credit-Scoring-Asynchronous (Fig. 4), Dispatch-of-Goods, Recourse, and Self-Service Restaurant. These diagrams were selected based on their complexity, which effectively captures the common challenges encountered in BPMN modeling. The Credit-Scoring-Asynchronous diagram includes an `IntermediateEvent`, illustrating asynchronous communication, and features an exception flow represented by an `ExclusiveGateway`. Dispatch-of-Goods diagram incorporates a `ParallelGateway`, requiring tasks to be executed concurrently. Additionally, the Recourse and Self-

Service Restaurant BPMN diagrams, included in the repository used for this study, further contribute to the variety and complexity of the scenarios explored. These characteristics make the selected diagrams particularly suitable for evaluating the proposed approach in diverse and intricate BPMN contexts.

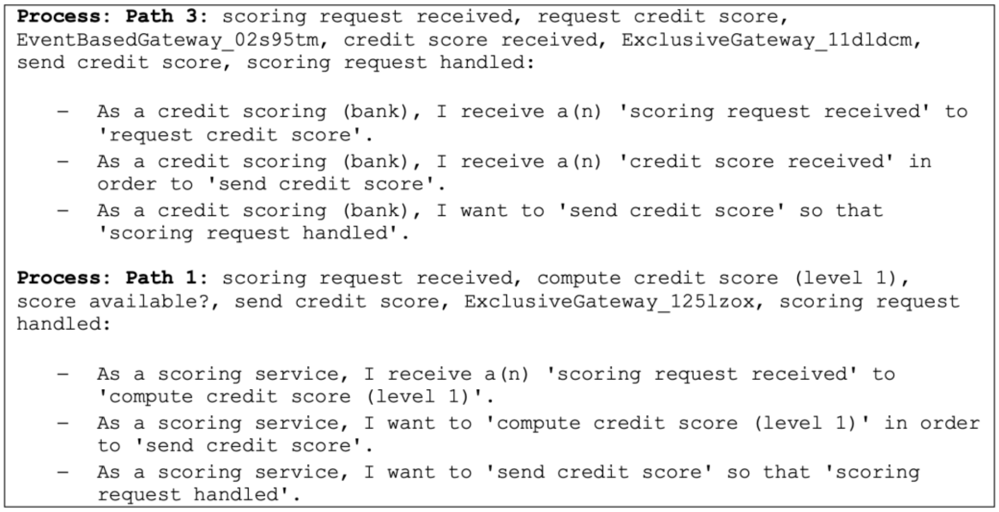
A. *Evaluation of the Gherkin test case generated*

We selected several relevant studies to compare with our approach, focusing exclusively on papers that generate Gherkin test cases. These include [2], [9], [11], and [12]. Table V summarizes the comparison, evaluating these approaches based on their support for Gherkin structures and the BPMN elements captured in their Gherkin test cases.

TABLE V
COMPARISON OF GHERKIN TEST GENERATED

Category	Criteria	De Moura et al. [11]	Paiva et al. [12]	Mateus et al. [2]	Von Olberg & Strey [9]	FlowForge
Gherkin structure	Given	✓	✓	✓	✓	✓
	When	✓	✓	✓	✓	✓
	Then	-	✓	✓	✓	✓
BPMN Elements	Activity	✓	✓	✓	✓	✓
	Lane	-	-	✓	✓	✓
	Event	-	✓	Unclear	Unclear	✓
	Gateway	✓	✓	Unclear	✓	✓
	Sequence Flow	-	✓	Unclear	✓	✓

In terms of Gherkin structures, [2], [9], [12] and FlowForge fully support all three components of Gherkin syntax ("Given," "When," and "Then"), while [11] lacks support for the "Then" component. An example of a Gherkin test case generated by our approach is shown in Fig. 9, which presents the test case for a BPMN Credit Scoring Asynchronous process.

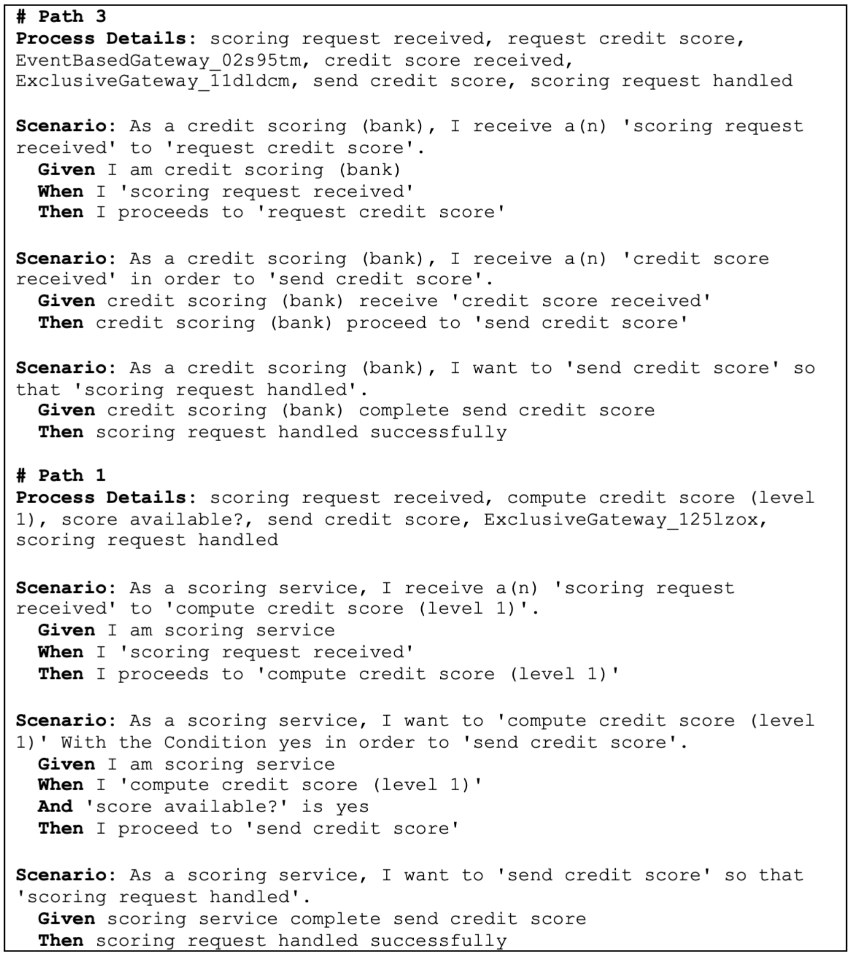


```
Process: Path 3: scoring request received, request credit score,
EventBasedGateway_02s95tm, credit score received, ExclusiveGateway_11dldcm,
send credit score, scoring request handled:

   –  As a credit scoring (bank), I receive a(n) 'scoring request received' to
      'request credit score'.
   –  As a credit scoring (bank), I receive a(n) 'credit score received' in
      order to 'send credit score'.
   –  As a credit scoring (bank), I want to 'send credit score' so that
      'scoring request handled'.

Process: Path 1: scoring request received, compute credit score (level 1),
score available?, send credit score, ExclusiveGateway_125lzox, scoring request
handled:

   –  As a scoring service, I receive a(n) 'scoring request received' to
      'compute credit score (level 1)'.
   –  As a scoring service, I want to 'compute credit score (level 1)' in order
      to 'send credit score'.
   –  As a scoring service, I want to 'send credit score' so that 'scoring
      request handled'.
```

Fig. 8. Example of Generated User Stories from Credit-Scoring-Asynchronous Path 3 and Path 1

These test cases represent user stories derived from paths generated through the extraction of BPMN elements integrated with DMN. Fig. 8 illustrates a user story example from the Credit-Scoring-Asynchronous process (Paths 3 and 1), showcasing the inclusion of an `IntermediateEvent` and `ExclusiveGateway`, which play critical roles in process flow. This scenario highlights how events and gateways are integrated into workflows, capturing the conditional logic and transitions necessary for processing credit scoring requests. Fig. 9 demonstrates the corresponding Gherkin scenarios for the same paths, adhering to predefined rules to ensure the test cases reflect the process flow and decision logic accurately, including conditions tied to the `IntermediateEvent` and `ExclusiveGateway`.

Regarding BPMN elements captured in the Gherkin test cases, FlowForge stands out as the only approach that fully supports all listed basic BPMN elements. In contrast, the Gherkin generated by [11] captures only `Activities` and gateways, omitting other elements. For [12], all elements are supported except for the `Lane` element, which represents process participants. The support provided by [2] for `Event, Gateway`, and `SequenceFlow` remains unclear due to the lack of Gherkin examples and a defined pattern in the paper. The Gherkin generated by [9] is the only paper that nearly captured all BPMN element, with the exception of one, an `Event`, which can't be definitely identified as captured. This is because they only provide one example, and it is important to note that it is merely an example and has not been implemented yet. Therefore, the output consists solely of the plan in Gherkin format. Overall, FlowForge demonstrates the most comprehensive support, capturing both complete Gherkin syntax and all basic BPMN elements.



```
# Path 3
Process Details: scoring request received, request credit score,
EventBasedGateway_02s95tm, credit score received,
ExclusiveGateway_11dldcm, send credit score, scoring request handled

Scenario: As a credit scoring (bank), I receive a(n) 'scoring request
received' to 'request credit score'.
  Given I am credit scoring (bank)
  When I 'scoring request received'
  Then I proceeds to 'request credit score'

Scenario: As a credit scoring (bank), I receive a(n) 'credit score
received' in order to 'send credit score'.
  Given credit scoring (bank) receive 'credit score received'
  Then credit scoring (bank) proceed to 'send credit score'

Scenario: As a credit scoring (bank), I want to 'send credit score' so
that 'scoring request handled'.
  Given credit scoring (bank) complete send credit score
  Then scoring request handled successfully

# Path 1
Process Details: scoring request received, compute credit score (level
1), score available?, send credit score, ExclusiveGateway_125lzox,
scoring request handled

Scenario: As a scoring service, I receive a(n) 'scoring request
received' to 'compute credit score (level 1)'.
  Given I am scoring service
  When I 'scoring request received'
  Then I proceeds to 'compute credit score (level 1)'

Scenario: As a scoring service, I want to 'compute credit score (level
1)' With the Condition yes in order to 'send credit score'.
  Given I am scoring service
  When I 'compute credit score (level 1)'
  And 'score available?' is yes
  Then I proceed to 'send credit score'

Scenario: As a scoring service, I want to 'send credit score' so that
'scoring request handled'.
  Given scoring service complete send credit score
  Then scoring request handled successfully
```

Fig. 9. Example of Generated Gherkin Test Case from Credit-Scoring-Asynchronous Path 3 and Path 1

B. Element Completeness

From the transformation of the four BPMN diagrams utilized in this research, we proceed to analyze completeness of element generated by FlowForge to determine whether the generated test cases effectively encompass all BPMN elements. This evaluation ensures that each element within the BPMN diagrams is represented in the test cases, verifying that all possible execution paths are accounted for. As illustrated in Table VI, the analysis compares the extracted elements and extracted paths with those present in the original BPMN file, highlighting the completeness of the transformation process.

$$Completeness = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)} \tag{1}$$

As observed in Equation (1), the formula presented is utilized in Table VI to calculate the Element Completeness. This equation serves as the foundational method for evaluating the degree to which all elements within the BPMN models are accurately identified and extracted. By applying this formula, the study ensures a standardized approach to measuring the completeness of elements across different BPMN models, providing a consistent basis for comparison and analysis.

TABLE VI
ELEMENT COMPLETENESS RESULT

BPMN Name	Total Element from BPMN	Total Element Extracted	Element Completeness
Dispatch-of-Goods	15	15	100%
Credit-Scoring-Asynchronous	19	19	100%
Recourse	20	20	100%
Self-Service Restaurant	30	28	93%
Average			**98.25%**

As shown in Table VI, among the four BPMN models utilized in this study, the BPMN for the Self-Service Restaurant achieved an Element Completeness score of only 93%. This lower score is attributed to a bug present in one of the prototype's functions, specifically during the process of extracting elements from the BPMN. The algorithm employed failed to detect an element with a looping structure in the BPMN, resulting in incomplete element recognition.

C. *Path Accuracy*

To ensure the accuracy of the generated test cases, we also evaluated the correctness of the paths produced by FlowForge using equation (2). Accuracy is essential to determine whether the generated paths precisely match the expected number of paths, ensuring no paths are missing that could result in incomplete test cases. The accuracy results are presented in Table VII.

$$Accuracy = \frac{True\ Positive\ (TP) + True\ Negative\ (TN)}{Total\ Elements\ \ (TP + TN + FP + FN)} \tag{2}$$

TABLE VII
PATH ACCURACY RESULT

BPMN Name	Expected Total Path from BPMN	Total Path Extracted	Path Accuracy
Dispatch-of-Goods	4	3	75%
Credit-Scoring-Asynchronous	4	4	100%
Recourse	5	5	100%
Self-Service Restaurant	4	3	75%
Average			**87.5%**

For the BPMN model "Dispatch-of-Goods," the process contains 4 expected paths; however, only 3 paths were successfully generated, resulting in an accuracy rate of 75%. This discrepancy is attributed to the implementation of BPMN gateway concepts in FlowForge. For instance, in the case of a `ParallelGateway` that branches into two `Activities`, the generated path is simplified into one sequence of activities instead of separate branches. For the BPMN model "Self-Service Restaurant," the accuracy rate is also 75%, as one path includes a looping structure that fails to reach the `EndEvent`. This limitation arises because valid paths must traverse from the `StartEvent` to the `EndEvent`. For the remaining two BPMN models, all paths were generated accurately, achieving 100% accuracy.

D. Execution Time

The other metric used in this research is measuring the execution time from the moment the BPMN file is uploaded until the process of generating User Stories and Gherkin Test Cases is completed. This procedure is repeated three times, and the average execution time is calculated for each BPMN model utilized in the study. As shown in Table VIII, this approach ensures the reliability and consistency of the results by mitigating potential variations in execution time across multiple trials. By averaging the results, the study provides a more accurate representation of the algorithm's performance and efficiency in handling different BPMN models.

TABLE VIII
EXECUTION TIME OF EVERY BPMN USED

BPMN Name	Run 1	Run 2	Run 3	Average
Dispatch-of-Goods	0.30s	0.30s	0.30s	0.30s
Credit-Scoring-Asynchronous	0.80s	0.30s	0.40s	0.50s
Recourse	0.40s	0.40s	0.30s	0.36s
Self-Service Restaurant	0.30s	0.30s	0.30s	0.30s
	Average			**0.36s**

As shown in Table VIII, the average execution time for the BPMN models Dispatch-of-Goods, Recourse, and Self-Service Restaurant is recorded at 0.3 seconds. In contrast, the longest execution time is observed in the Credit-Scoring-Asynchronous BPMN model, with an average of 0.5 seconds. This discrepancy highlights the varying complexity, and processing demands of different BPMN models, with the Credit-Scoring-Asynchronous model requiring additional time for element extraction and generation processes. The results emphasize the influence of BPMN structure and algorithmic efficiency on overall execution performance.

E. Discussions

The findings from the four evaluations conducted reveal several limitations in our approach. First, the extraction of BPMN elements by our tool shows a notable lack of support for detecting elements involved in looping structures, an issue similarly noted in prior studies [12][13]. This limitation is prevalent in research relying on direct transformation of BPMN models. While such direct transformations ensure completeness without considering model modifications, utilizing intermediate representations, such as state graphs, could address this issue, as demonstrated in [10]. Due to this shortcoming, the generated paths do not entirely reflect the BPMN structure, as two elements, specifically tasks and events, remain undetected.

Second, the generated paths often fail to accurately represent the actual business process flow, particularly for cross-pool or `MessageFlows`, as shown in Table IV. The expected paths, fully reflecting the business flow, are outlined in Table IX, where only two combined paths emerge from the sequences in Table IV. Additionally, a critical limitation is the inability to identify the default path within `InclusiveGateways` in the BPMN model. `InclusiveGateways`, by nature, allow for either a single default path or concurrent execution of multiple paths when outgoing flows lack explicit expressions. Our study does not currently provide mechanisms to discern or validate the default path in such scenarios. This limitation hampers the thorough validation of all business process flows and highlights a gap in the existing body of research, as no prior work has fully addressed this issue. However, it can be confidently asserted that all paths within the BPMN are adequately covered.

Third, regarding test case generation time, our approach demonstrates efficiency, with an average processing time of 0.3 seconds from the BPMN input to Gherkin test case generation. This aligns with the performance observed in related studies[10][11]. However, this metric does not account for the execution time of the test cases themselves, as this aspect lies beyond the scope of our current research. Future studies should analyze the execution of generated Gherkin scripts using platforms such as Cucumber to gain a deeper understanding of execution performance.

TABLE IX
EXPECTED GENERATED PATH OF "CREDIT SCORING"

Path	Combined Path
1 & 3	Scoring request received → Compute credit score (level 1) → Exclusive Gateway → **Report delay**→ **Delay information received** → Report delay → Credit score received → ExclusiveGateway → Send credit score → Scoring request handled
2 & 4	Scoring request received → Compute credit score (level 1) → Exclusive Gateway → **Send credit score**→ **Credit score received** → ExclusiveGateway → Send credit score → Scoring request handled

Lastly, the patterns employed in this study do not encompass all types of BPMN elements. For instance, `Event-basedGateways` that has another type besides message, such as timer, conditional, and signal types are insufficiently addressed. This underscores the need for further development and refinement of patterns, as highlighted in this study and prior research [2]. Expanding these patterns will enhance the comprehensiveness and applicability of the approach.

V. CONCLUSION AND FUTURE WORK

This paper presented FlowForge, which transforms BPMN models into User Story and Gherkin test cases by integrating DMN and pattern matching, as utilized in previous research [2][14]. While the element extraction process achieved high accuracy, with three out of four BPMN models attaining 100% Element Completeness, the Self-Service Restaurant model showed a lower score of 93%, with the total average completeness is 98.25%. Path Accuracy also varied, with most models achieving 100%, but the Dispatch-of-Goods and Self-Service Restaurant models only achieve 75%, with the average accuracy 87.5%. But Execution time averaged 0.36 seconds, showing that time taken to generate test case is efficient.

From that result, it is evident that incorporating DMN allows for the extraction of rules embedded within BPMN diagrams, enabling the identification of exceptions represented by exclusive gateways. This approach addresses the gaps identified in prior research [2]. Moreover, the generated Gherkin includes fully detailed steps—Given, When Then—whereas prior studies, such as [11], lacked the "Then" step. Additionally, our application specifies the roles responsible for performing activities within the user stories, addressing another gap highlighted in earlier research [12]. This paper highlights the challenges in managing complex pathways, exceptions, and evolving BPMN patterns during transformation.

In this position paper, we acknowledge that there are still several opportunities for improvement. A strategic roadmap for future work is proposed to enhance the accuracy, efficiency, and coverage of the BPMN-based test case generation approach. (1) Enhancing BPMN Element Detection: Improve the accuracy of detecting looping structures and complex elements in BPMN models by integrating intermediate representations such as state graphs. This approach will better capture looping relationships and address undetected elements, such as tasks and events. Algorithmic improvements will be validated using diverse BPMN scenarios, particularly those involving loops and branches. (2) Improving Path Representation: Develop mechanisms to handle cross-pool and message flows more accurately while ensuring paths reflect actual business process flows. Focus on identifying and validating default paths in Inclusive Gateways through heuristics or advanced modeling techniques. These enhancements will ensure comprehensive and accurate path generation for all process flows. (3) Expanding the Pattern: Extend the pattern library to include additional BPMN elements, such as various event types (message, timer, conditional, signal) and event-based gateways. Adaptive patterns will be developed to handle evolving BPMN standards, ensuring robustness and applicability to a wide range of models. (4) Validate Test Case Execution Performance: Execute Gherkin-based test scripts on platforms like Cucumber to analyze test execution performance. This will help identify inefficiencies in execution and guide refinements in test case generation.

DATA AND COMPUTER PROGRAM AVAILABILITY

Data used in this paper can be accessed in the following site https://github.com/camunda/bpmn-for-research

REFERENCES

[1] V. Sklyar and V. Kharchenko, "Domain Specific Modelling and Language for Safety-Critical and Security-Critical Requirements Engineering," in *2022 12th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Athens, Greece: IEEE, Dec. 2022, pp. 1–7. doi: 10.1109/DESSERT58054.2022.10018738.

[2] D. Mateus, D. S. Da Silveira, and J. Araújo, "A Systematic Approach to Derive User Stories and Gherkin Scenarios from BPMN Models," in *Business Modeling and Software Design*, vol. 483, B. Shishkov, Ed., in Lecture Notes in Business Information Processing, vol. 483. , Cham: Springer Nature Switzerland, 2023, pp. 235–244. doi: 10.1007/978-3-031-36757-1_15.

[3] G. Nagy and S. Rose, *Discovery: Explore behaviour using examples*. CreateSpace Independent Publishing Platform, 2018.

[4] D. Chelimsky, Ed., *The RSpec book: behaviour-driven development with RSpec, Cucumber, and Friends*. Lewisville, Tex: Pragmatic, 2010.

[5] S. Rose and G. Nagy, *Formulation: Document examples with Given/When/Then*. Independently published, 2021.

[6] Dion Moult and Thomas Krijnen, "Compliance checking on building models with the Gherkin language and Continuous Integration," *Proc. EG-ICE 2020 Workshop Intell. Comput. Eng.*, pp. 294–303, 2020, doi: 10.14279/DEPOSITONCE-9977.

[7] K. Schneid, H. Kuchen, S. Thöne, and S. Di Bernardo, "Uncovering data-flow anomalies in BPMN-based process-driven applications," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, Virtual Event Republic of Korea: ACM, Mar. 2021, pp. 1504–1512. doi: 10.1145/3412841.3442025.

[8] T. Lopes and S. Guerreiro, "Assessing business process models: a literature review on techniques for BPMN testing and formal verification," *Bus. Process Manag. J.*, vol. 29, no. 8, pp. 133–162, Dec. 2023, doi: 10.1108/BPMJ-11-2022-0557.

[9] P. Von Olberg and L. Strey, "Approach to Generating Functional Test Cases from BPMN Process Diagrams," in *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*, Melbourne, Australia: IEEE, Aug. 2022, pp. 185–189. doi: 10.1109/REW56159.2022.00042.

[10] A. Yazdani Seqerloo, M. J. Amiri, S. Parsa, and M. Koupaee, "Automatic test cases generation from business process models," *Requir. Eng.*, vol. 24, no. 1, pp. 119–132, Mar. 2019, doi: 10.1007/s00766-018-0304-3.

[11] J. L. De Moura, A. S. Charao, J. C. D. Lima, and B. De Oliveira Stein, "Test case generation from BPMN models for automated testing of Web-based BPM applications," in *2017 17th International Conference on Computational Science and Its Applications (ICCSA)*, Trieste, Italy: IEEE, Jul. 2017, pp. 1–7. doi: 10.1109/ICCSA.2017.7999652.

[12] A. Paiva, N. Flores, J. Faria, and J. Marques, “End-to-end Automatic Business Process Validation,” *Procedia Comput. Sci.*, vol. 130, pp. 999–1004, Jan. 2018, doi: 10.1016/j.procs.2018.04.104.

[13] K. Schneid, L. Stapper, S. Thone, and H. Kuchen, “Automated Regression Tests: A No-Code Approach for BPMN-based Process-Driven Applications,” in *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*, Gold Coast, Australia: IEEE, Oct. 2021, pp. 31–40. doi: 10.1109/EDOC52215.2021.00014.

[14] B. Boonmepipit and T. Suwannasart, “Test Case Generation from BPMN with DMN,” in *Proceedings of the 2019 3rd International Conference on Software and e-Business*, Tokyo Japan: ACM, Dec. 2019, pp. 92–96. doi: 10.1145/3374549.3374582.

[15] Object Management Group, “Business Process Model and Notation (BPMN), Version 2.0.” 2014. [Online]. Available: https://www.omg.org/spec/BPMN

[16] J. Recker, “Opportunities and constraints: the current struggle with BPMN,” *Bus. Process Manag. J.*, vol. 16, no. 1, pp. 181–201, Feb. 2010, doi: 10.1108/14637151011018001.

[17] C. Nonchot and T. Suwannasart, “A Tool for Generating Test Case from BPMN Diagram with a BPEL Diagram,” *Hong Kong*, 2016.

[18] P. Pokharel and P. Vaidya, *A Study of User Story in Practice*. 2020, p. 5. doi: 10.1109/ICDABI51230.2020.9325670.