


Supporting the Automated Generation of Acceptance Tests of Process-Aware Information Systems

Tales M. Paiva¹^a, Toacy C. Oliveira^{1,3}^b, Raquel M. Pillat²^c and Paulo S. C. Alencar³

¹Computer and Systems Engineering Program, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

²Software Engineering Program, Universidade Federal do Pampa, Alegrete, Brazil

³Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada

Keywords: Test Automation, RPA, BPMN, Model-Based Testing.

Abstract: Software quality assurance is a crucial process that ensures software products meet specified requirements and quality standards. Achieving an exhaustive test coverage is essential for quality assurance, particularly in complex and dynamic Process-Aware Information Systems (PAIS) built upon the Business Process Model and Notation (BPMN). Manual testing in such systems is challenging due to many execution paths, dependencies, and external interfaces. This paper proposes a model-based testing strategy that uses BPMN models and build specifications as input to generate a Robotic Process Automation (RPA) script that automates a comprehensive User Acceptance Test procedure. Leveraging on Robotic Process Automation (RPA) to automate user interactions allows for reducing the need for testers to manually input PAIS-related information when handling user forms. We also present a Case Study to demonstrate the feasibility of our approach.

1 INTRODUCTION


Software quality is a systematic process that ensures software products meet specified requirements and quality standards. In a more aspirational definition, it is "an ideal toward which we strive" (Lawrence-Pfleeger and Atlee, 1998). The number and the type of failures detected are a proxy to the assessment of software quality, being an internal direct measure of the testing process and an external indirect measure of the attribute of software quality itself. Therefore, having a proper testing strategy, with a high test coverage, can contribute to the quality assurance of the software developed.


According to (Dumas et al., 2005), a Process-Aware Information System (PAIS) is "a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models". PAISs are based on business process models such as the Business Process Model and Notation (BPMN) (OMG, 2014). Test coverage in PAIS is challenging due to the inherent complexity and dynamism of these systems.


A process typically features multiple execution paths, options, and dependencies among activities, including decision points, parallel execution, and synchronization. The sheer volume of potential combinations makes it challenging for a human to test exhaustively all scenarios, as well as functional and non-functional requirements. Furthermore, these systems often interface with external systems and human interactions, introducing further variability. Maintaining high test coverage becomes increasingly difficult as the system evolves, requiring significant time and resources to keep test suites up-to-date and effective.

This paper proposes a model-based testing strategy that uses BPMN models and build specifications as input to generate a Robotic Process Automation (RPA) script that automates a comprehensive User Acceptance Test procedure. In other words, this work presents a solution that generates RPA scripts to support the execution of the Acceptance Tests, covering all the possible paths within the BPMN of the PAIS under test. We also present a Case Study to demonstrate the feasibility of our approach.

Given a PAIS is fundamentally architected upon process models (Dumas et al., 2005), in this case BPMN (OMG, 2014), it is possible to employ a model-based strategy for test case generation (Utting et al., 2016; de Moura et al., 2017). This systematic

^a <https://orcid.org/0000-0003-2036-3442>

^b <https://orcid.org/0000-0001-8184-2442>

^c <https://orcid.org/0000-0002-5420-6966>

technique can ensure comprehensive test coverage, meticulously accounting for all conceivable paths and dependencies among the activities within the model. By doing so, it could effectively mitigate the aforementioned challenge of ensuring exhaustive test coverage in such complex and dynamic environments.

Considering that the BPMN standard is very explicit about the human interactions within a process, it is feasible to compile a comprehensive set of human interactions in the generated test cases. These interactions can serve as valuable guidance for testers, allowing for increased test coverage and subsequently enhancing the efficiency and effectiveness of the acceptance testing process. Moreover, by strategically implementing Robotic Process Automation (RPA), it is feasible to automate the execution of these user interactions (Enríquez et al., 2020). This strategic deployment of RPA further amplifies the efficiency and consistency of the overall testing procedure, providing added benefits to the testing process as a whole.

A model-based approach can automate the generation of acceptance tests (Ramler and Klammer, 2019), increasing the test coverage of a PAIS based on the business process models. The automated generation of such tests can improve the quality of tests by reducing the possibility of human errors, improving its reliability and consistency.

Continuous delivery (CD) is an evolving software engineering paradigm that refers to the iterative practice of delivering software to end-users in frequent and regular cycles (Makki et al., 2016). (Humble and Farley, 2010) states that adopting acceptance criteria-driven tests in the CD strategy is seen as a progressive step to further improve software quality. According to (Gmeiner et al., 2015), numerous companies are embracing automated testing within their pipelines as an enabler to CD, for it is seen as a strategic capability that offers a competitive advantage.

Automating software tests provide three key benefits: repeatability, leverage, and accumulation (Fenster and Graham, 1999). Automated test generation and execution can significantly reduce the time and effort required to create test cases manually, freeing up the tester's time, allowing them to focus on more complex and higher-value tasks, while enabling a regression testing strategy.

The remainder of this paper is organized as follows: Section 2 presents the background of this research. The proposed solution is presented in Section 3, and a case study is demonstrated in Section 4 and discussed in Section 5. Related work is shown in Section 6, while Section 7 concludes the paper and presents future work.

2 BACKGROUND

The Software Development and Testing are vast domains of software engineering, with several different frameworks and approaches. Given the focus of the present work on automated generation and execution of acceptance tests of BPMN-based PAIS through the usage of RPA, it is instrumental that these concepts are well defined within its scope.

The use of MBT-related techniques in a BPMN-based PAIS allows for the derivation of the User Acceptance Test (UAT) suite from the process model, since it encapsulates the expected interactions between end-users and the system within process. These generated test cases in the test suite then facilitate the UAT, allowing users to validate whether the system aligns with their expectations and fulfills the requirements, while also presenting a good opportunity for the automated testing of the user interfaces through the use of Robotic Process Automation (RPA) (Enríquez et al., 2020).

2.1 Software Development and Testing Life Cycles

The Software Development Life Cycle (SDLC) and Software Testing Life Cycle (STLC) are integral processes in software development. The SDLC is a methodological framework employed within the domain of software engineering that encompasses the systematic stages involved in developing software, including requirements gathering, design, coding, testing, deployment, and maintenance.

The SDLC framework aims to maximize the quality of software products, manage project timelines and costs effectively, and minimize the potential risks associated with software development. By employing SDLC, organizations strive to deliver software products that meet specified requirements, ensure functional precision, and provide value to end-users.

On the other hand, the STLC focuses specifically on testing activities within the SDLC, involving planning, designing, executing, and evaluating tests to verify software functionality and quality. Both processes work hand in hand, with STLC ensuring that software meets requirements through comprehensive testing. Together, they facilitate structured and efficient software development, resulting in high-quality products that meet stakeholder expectations.

According to (Leung and Wong, 1997), software testing consists of:

- Unit Test: testing new functionalities for correctness, usually tested against function and code coverage requirements;

- **Integration Test:** designed to test the unit interfaces and the interactions among the units;
- **System Test:** checks for defects in the functionalities of the system, typically using a black-box approach;
- **Acceptance Test:** test the software against the user requirements, both functional and non-functional, to demonstrate the software readiness for operational use.

In the STLC, although testing consists on an important aspect (Padmini et al., 2016), most of the existing testing strategies are focused on the developer rather than the final user (Leung and Wong, 1997).

2.1.1 Acceptance Testing

As defined in (IEEE, 2017), Acceptance Testing is *”testing conducted to determine whether a system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system”*.

In that context, Acceptance Tests, also referred as User Acceptance Tests (UAT) when conducted with the presence of or even by the customer, assess whether a feature is working from the customer’s perspective and ensuring the customer’s satisfaction with the final product (Melnik et al., 2004), while enhancing their confidence to accept the system (Padmini et al., 2016).

One key difference between the test modalities is that Unit and Integration Tests are modeled and written by developers, System Tests are modeled and written by developers or testers, while the Acceptance Tests, specially UATs, are usually modeled by the customers, and possibly even written by them, with the aid of a developer or tester (Melnik et al., 2004). Conventionally, the end-users enumerate a set of acceptance test cases, covering the *”major functions, user interface, and capabilities in handling invalid input and exceptions in operation”*, with the main objective of evaluating the system readiness for operational use (Leung and Wong, 1997).

2.1.2 Model-Based Testing

Model-based testing (MBT) constitutes a specialized testing approach predicated on the utilization of explicit behavior models. These models encapsulate the projected behaviors of the system under test (SUT) or potentially, its corresponding environment. Test cases originate from either one of these behavioral representations or a synthesis of them, and are subsequently deployed for execution on the designated SUT (Utting et al., 2011).

MBT embodies a testing paradigm in which test cases are wholly or partially generated from a model. For the successful deployment of this approach, it necessitates that the software’s behavioral or structural characteristics are explicitly delineated by models crafted with well-defined rules. These models may take the form of formal models, finite state machines, UML diagrams, among others (Utting and Legeard, 2006).

The benefits of MBT, according to (Dias-Neto and Travassos, 2010):

1. Lower cost and effort for testing planning/execution and shorter testing schedule;
2. Improvement of the final product quality, because the models are used as an oracle for testing;
3. Testing process can be automated;
4. Ease of communication between the development and testing teams;
5. Capacity of automatically generating and running large sets of useful and non-repetitive (non-redundant) tests;
6. Ease of updating the test cases set after the software artifacts used to build the software model changes;
7. Capacity of evaluating regression testing scenarios.

2.2 Process-Aware Information Systems

Process-Aware Information Systems (PAIS) are software systems that automate and support business processes, integrating process modeling, execution, monitoring, and analysis (Dumas et al., 2005; Aalst, 2009). PAIS enable organizations to streamline workflows, track progress, handle exceptions, and improve operations. They provide a framework for aligning information systems with business processes, enhancing efficiency, and facilitating effective management of complex workflows.

Various providers offer PAIS solutions, including commercial vendors and open-source platforms (Telemaco et al., 2022). Among them, Camunda¹ stands out as an open-source provider with comprehensive PAIS capabilities. Camunda offers a flexible and scalable platform for modeling, executing, and monitoring complex processes. It is known for its extensive feature set and robust integration capabilities, making it a popular choice for organizations seeking an open-source PAIS solution.

¹<https://camunda.com/platform-7/>

The AKIP Process Automation Platform² is an open source project devoted to facilitate Process/Workflow Automation initiatives based on code generation techniques, aiming to promote the construction and dissemination of PAISs utilizing established technologies such as BPMN, Java, Javascript and Camunda, enabling the creation of modern web applications which are named *KIPApps* (Telemaco et al., 2022).

2.3 Business Process Model and Notation

Business Process Model and Notation³ (BPMN) is a standardized graphical notation used for modeling business processes, developed by the Object Management Group (OMG) and first introduced in 2004. It provides a visual representation that enables organizations to document, analyze, and communicate their business processes effectively (OMG, 2014).

The notation incorporates symbols and elements to represent various aspects of a process, intending to *“standardize a business process model and notation in the face of many different modeling notations and viewpoints”* (OMG, 2014). It has become widely adopted in the industry due to its clarity, versatility, and ability to bridge the gap between all stakeholders involved, providing *“a simple means of communicating process information to other business users, process implementers, customers, and suppliers”* (OMG, 2014).

In BPMN-based PAIS, such as *KIPApps* (Telemaco et al., 2022), the human points of interaction within a business process play a crucial role in assessing software quality. These interactions occur when tasks require human input or decision-making. BPMN provides specific elements to model these interactions. Start events have a form associated with it, and user tasks represent activities that involve human participants.

UATs play a significant role in BPMN-based PAIS to ensure the quality and usability of the implemented processes. In BPMN-based PAIS, UATs focus on validating the process models and their execution within the system. By aligning UATs with the BPMN diagrams, organizations can assess if the implemented processes adhere to the intended logic and achieve the desired outcomes. This validation process helps to identify and address any discrepancies or issues early on, improving user satisfaction and system reliability.

Moreover, since BPMN serves as the underlying

model for these systems, MBT leverages this model to automate the testing process. This approach enables comprehensive coverage of user interactions and early detection of potential issues. By utilizing MBT, organizations can improve the effectiveness and efficiency of UATs, ensuring robust testing and higher software quality within BPMN-based PAIS.

BPMN plays a crucial role in PAIS by providing a standardized language for modeling and executing business processes. By utilizing BPMN, PAIS can represent complex process logic, control flow, and data dependencies in a visual and standardized manner.

(Lübke and van Lessen, 2017) states that the adoption of BPMN as a universal modeling language facilitates effective communication among all project participants and encourages the reuse of existing editors and repositories. Although their work was focused on service-based processes, the same argument can be derived for human-oriented processes and its User-Acceptance Tests.

2.4 Robotic Process Automation

Robotic Process Automation (RPA) refers to the use of software robots or “bots” to automate repetitive, rule-based tasks within business processes. These bots mimic human interactions with various software systems and perform tasks such as data entry, form filling, and screen navigation. Its roots can be traced back to screen scraping tools and macros, which were later refined into more advanced automation capabilities.

RPA finds utility in UATs by automating repetitive and manual testing activities (Enríquez et al., 2020). Since UATs involve end-users or stakeholders validating a system’s functionality, user experience, and compliance with requirements, RPA can streamline and accelerate this testing cycle by automating the execution of test cases, data input, and result verification. Bots can interact with the system’s user interface, simulate user actions, and compare expected outcomes with actual results. This automation reduces manual effort, increases test coverage, and enhances the efficiency of the UAT process. However, it’s crucial to emphasize that the current solution is not designed to replace the end user or eliminate the acceptance testing phase. Instead, its goal is to enhance test coverage and facilitate a smoother acceptance testing process.

When it comes to open-source RPA solutions, one notable option is Robot Framework⁴ (RF). RF is a generic test automation framework that supports not

²<https://agilekip.github.io/pap-documentation/about>

³<https://www.omg.org/bpmn/>

⁴<https://robotframework.org/>

only RPA but also other types of software testing. It provides a simple and readable syntax, making it accessible for both technical and non-technical users. RF allows the creation of test cases using keywords, which can be extended through libraries or custom implementations. Its open-source nature fosters a collaborative community and provides flexibility for customization and integration with various tools and technologies.

3 SOLUTION OVERVIEW

Given the goal of improving the software quality assurance process, and that the system under test (SUT) is a BPMN-based PAIS, which has clear definitions about the human interactions points throughout a process (Telemaco et al., 2022; OMG, 2014), it would be beneficial to use an MBT approach for generating the UAT test cases.

A scheme of the proposed solution can be seen in Figure 1, and has the following procedure:

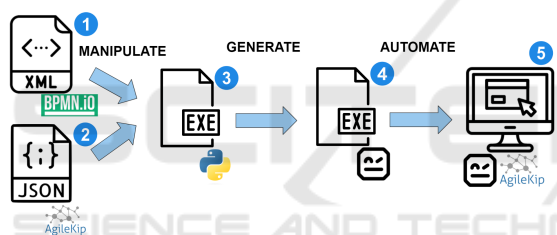


Figure 1: Scheme of the solution.

1. From the chosen process' BPMN, i.e. its XML, extract all the points within the process in which there are human interaction according to the AKIP's definitions (Telemaco et al., 2022), which are:
 - the Start Event, which has a form related to it;
 - the User Tasks present in the process definition, each of which is associated with a corresponding form;
2. From the AKIP entity JSONs⁵ that scaffold the process forms and complex entities utilizing the JHipster Domain Language (JDL)⁶, gather the specification of the fields and data of each human interaction point:
 - the JSON related to the form present in the Start Event;

⁵<https://agilekip.github.io/pap-documentation/tutorials/getting-started>

⁶<https://www.jhipster.tech/jdl/entities-fields>

- the JSONs for the forms of each User Task present in the BPMN;
3. Manipulate the aforementioned files using a Python script;
 4. Generate the RPA scripts of the Acceptance Tests as seen in Algorithm 1;
 5. Execute the automated Acceptance Tests of the said process defined in the BPMN using the RPA tool and generate a minimalist report of the execution;

Data: define the amount n of tests to be run.

Result: a report of all n executed tests with their process patterns identified.

Open the browser;

Go to the platform URL;

Log in the platform;

for $i = 0; i < n; i = i + 1$ **do**

Log the start of the execution of one test;

Generate mock data;

Execute the Start Form;

while *there is a User Task available*

within the current process execution do

Generate mock data;

Execute the User Task;

Log the User Task execution;

end

Log the end of the execution of one test;

end

Generate the tests execution log file;

Algorithm 1: Automated testing algorithm.

3.1 Implementation

The described solution implements a random execution of a given amount n of process instances, defined by the tester, and generates a minimal report of the process paths taken in the test cases executed, alongside with RF's standard thorough report. Each execution generate its own set of mock data utilizing the Faker library⁷. The UI automation is implemented using the RPA.Browser.Selenium library⁸.

A brief overview of the relationship between the fields declared in the AKIP entity JSONs from Step 2, and the mock data to be generated in Step 4 can be seen in Table 1.

The steps of the algorithm generated by the Python script can be seen in Algorithm 1. Each identified point of human interaction (Start Form and User

⁷<https://pypi.org/project/robotframework-faker/>

⁸<https://pypi.org/project/rpaframework/>

Table 1: Examples of data types relation.

JDL type	Faker type
String	Word or Sentence
Integer	Random Int
LocalDate	Date
Boolean	Boolean
Many-to-one String	Word or Sentence

Tasks) has its own interface automation implementation from the information present in the AKIP entity JSON. The execution time of the Python script that generates the RPA scripts is negligible.

Given the random nature of the data generated by the Faker library, the execution of each process instance is "blind", meaning that there's no previous knowledge or planning about the path that a specific process instance will have, and the User Tasks are executed in a "first come, first served" basis.

After executing n process instances, logs are provided and the tester can evaluate the executions and make decisions regarding the results.

4 CASE STUDY

An example is given utilizing a simple Customer Feedback Service process in a generated KIPApp (Telemaco et al., 2022). This PAIS is a scaled down version, for didactic purposes, of a realistic model used in the industry. The procedures can all be executed by the same developer, or they can be split throughout the team, given the following roles, for example:

- Process Analyst: model the BPMN, as in item 1 from Figure 1;
- Developer: create the AKIP's entity JSONs, as in item 2 from Figure 1, and scaffold the KIPApp⁹;
- Tester: gather the aforementioned artifacts, feed the Python script, generate the RPA scripts, execute the RPA and evaluate the results of the execution, as in items 3 to 5 from Figure 1.

The Customer Feedback Service is a business process in which a customer can submit a Request Form with a Complaint, a Suggestion or a Compliment. The process was modeled using the BPMN standard as seen in Figure 2, and is explained in Algorithm 2.

The web application (KIPApp) was scaffolded following the instructions present in (Telemaco et al., 2022), and an example of the content of a AKIP entity JSON, in this case the form associated with the Start Event, can be seen in Figure 3. The resulting user

⁹<https://agilekip.github.io/pap-documentation/about>

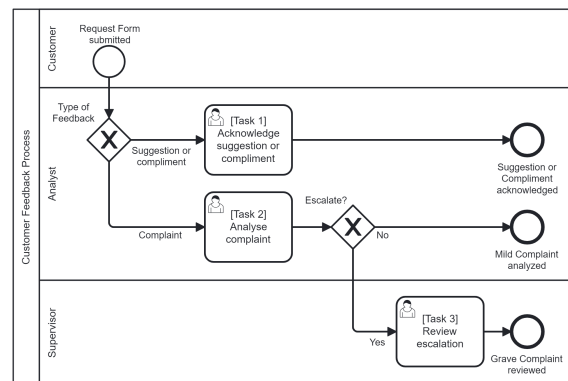


Figure 2: A BPMN of a simple business process model representing a Customer Feedback process.

Data: Customer feedback
 Customer submits the feedback;
if feedback is Compliment or Suggestion **then**
 | Analyst reviews positive feedback;
else
 | Analyst reviews negative feedback and determines its gravity;
 | **if** gravity is high **then**
 | | Escalates process to supervisor;
 | | Supervisor reviews feedback;
end
end
 End of process;

Algorithm 2: A simple Customer Feedback service.

interface scaffolded by the KIPApp for the Request Form can be seen in Figure 7.

The entity JSONs and the BPMN are fed into the Python script as in item 3 of Figure 1, and the Robot Framework's .robot files are generated by the Python script indicated in item 4. The Python script, as well as examples of the resulting files can be accessed in the following GitHub repository: <https://github.com/talesmp/aat4pais>.

The analogous version of the Algorithm 1, generated by the Python script utilizing the Robot Framework syntax can be seen in Figure 4.

From the AKIP entity JSONs, the `fieldName` is used to derive the XPath Locators for the RF implementation of the UI automation, and the `fieldType` is used to generate the proper mocked data through the Faker library, as described in Table 1.

An example regarding the "Generate mock data" step described in Algorithm 1 is given. The Python script manipulates the BPMN and the AKIP entity JSONs and generates a test file with the .robot extension, following RF's syntax. A code snippet related to this specific operation within the

```

1  {
2    "fields": [
3      {
4        "fieldName": "description",
5        "fieldType": "String"
6      },
7      {
8        "fieldName": "date",
9        "fieldType": "LocalDate"
10     }
11  ],
12  "relationships": [
13    {
14      "relationshipName": "babblingCharacterization",
15      "otherEntityName": "babblingCharacterization",
16      "relationshipType": "many-to-one",
17      "otherEntityField": "type"
18    }
19  ],
20  "entityType": "start-form",
21  "name": "CustomerFeedbackStartForm",
22  "processBpmnId": "CustomerFeedbackProcess",

```

Figure 3: A snippet from the AKIP Start Event form entity JSON.

```

36  TC_BlindBatch
37    ${kw_executed}= Create List
38    kwFakerDataSetup
39    kwLogin
40    FOR ${i} IN RANGE 30
41      ${inner_list}= Create List
42      kwFakerDataSetup
43      kwRequestForm
44      Append To List ${inner_list} Start of Execution #${i}
45      Append To List ${inner_list} kwRequestForm
46      WHILE $processRunning == True
47        kwFindFirstAvailableTask
48        IF $found_task == "TaskAnalyseComplaint"
49          kwFakerDataSetup
50          kwTaskAnalyseComplaint
51          Append To List ${inner_list} kwTaskAnalyseComplaint
52        ELSE IF $found_task == "TaskReviewEscalation"
53          kwFakerDataSetup
54          kwTaskReviewEscalation
55          Append To List ${inner_list} kwTaskReviewEscalation
56        ELSE IF $found_task == "TaskAcknowledge"
57          kwFakerDataSetup
58          kwTaskAcknowledge
59          Append To List ${inner_list} kwTaskAcknowledge
60        ELSE IF $found_task == "No task available."
61          ${processRunning}= Set Variable ${False}
62          Set Test Variable ${processRunning}
63        BREAK
64      END
65    END
66    Append To List ${inner_list} End of Execution #${i}
67    Append To List ${kw_executed} ${inner_list}
68  END
69  ${json_string}= Evaluate json.dumps(${kw_executed}, indent=4)
70  Create File executedKeywords.json ${json_string}

```

Figure 4: The RF syntax version of the Algorithm 1.

Python script can be seen in Figure 5, and the resulting generated RPA script can be seen in Figure 6.

For ease of use, the robot can be executed utilizing the Visual Studio Code extension¹⁰ provided by Robocorp. This makes the logging and debugging clearer, as well as the management of the necessary libraries easier.

¹⁰<https://robocorp.com/docs/developer-tools/visual-studio-code>

```

### Implementing the Faker for each interactable field ###
### An interactable field is a non-fieldReadOnly in the AgileKip entity JSON ###
test.write('kwFakerDataSetup\n')
for uif in unique_interactable_fields:
  if uif[1] == 'Boolean':
    test.write(' ${faker-'+uif[0]+'} FakerLibrary.Boolean\n')
    test.write(' Set Test Variable ${faker-'+uif[0]+'}\n')
  if uif[1] == 'LocalDate':
    test.write(' ${faker-'+uif[0]+'} FakerLibrary.Date\n')
    test.write(' Set Test Variable ${faker-'+uif[0]+'}\n')
  if uif[1] == 'String':
    test.write(' ${faker-'+uif[0]+'} FakerLibrary.Sentence nb_words=8\n')
    test.write(' Set Test Variable ${faker-'+uif[0]+'}\n')
  if uif[1] == 'Integer':
    test.write(' ${faker-'+uif[0]+'} FakerLibrary.Random Int min=1 max=10\n')
    test.write(' Set Test Variable ${faker-'+uif[0]+'}\n')
  if uif[1] == 'many-to-one':
    inputSubString = "processInstance."+domainNameInBpmnExpressions+"."+uif[0]
    collectionResult = extract_many_to_one_col(crudeOutgoingGatewayCond, inputSubString)
    test.write(' ${faker-'+uif[0]+'} FakerLibrary.Word ext_word_list="+str(colRes)+"\n')
    test.write(' Set Test Variable ${faker-'+uif[0]+'}\n')
test.write(' ${processRunning}= Set Variable ${True}\n')
test.write(' Set Test Variable ${processRunning}\n')
test.write('\n')

```

Figure 5: A snippet from the Python script that implements the mock data keyword in RF syntax.

```

124 kwFakerDataSetup
125   ${faker-log} FakerLibrary.Boolean
126   Set Test Variable ${faker-log}
127   ${faker-gravity} FakerLibrary.Random Int min=1 max=10
128   Set Test Variable ${faker-gravity}
129   ${faker-date} FakerLibrary.Date
130   Set Test Variable ${faker-date}
131   ${faker-description} FakerLibrary.Sentence nb_words=8
132   Set Test Variable ${faker-description}
133   ${faker-response} FakerLibrary.Sentence nb_words=8
134   Set Test Variable ${faker-response}
135   ${faker-babblingCharacterization.type} FakerLibrary.Word
136   ... ext_word_list=['suggestion','compliment','complaint']
137   Set Test Variable ${faker-babblingCharacterization.type}
138   ${processRunning}= Set Variable ${True}
139   Set Test Variable ${processRunning}

```

Figure 6: A snippet from the Robot Framework RPA script regarding the generation of the mock data.

The robot, mimicking a user and following the steps described in the Algorithm 1, opens a Request Form as a customer, fills a set of fields and submits the process, as seen in Figure 7.

Figure 7: The form generated for the Request Form.

Given the data about the Type of Request selected by the customer, i.e. a Complaint, a Suggestion or a Compliment, the process is directed to an analyst to either acknowledge the suggestion or compliment [Task 1], or to analyse the complaint [Task 2], which can be seen in Figure 8, while the AKIP entity JSON that generated the said form can be seen in Figure 9, with its specification regarding the read-only

fields, as seen in rows 06, 11 and 28, which reflect in non-editable fields in the user interface of that specific user task form.

The form is titled '#19251 - [Task 2] Analyse complaint' and is marked as 'Assigned'. It contains several input fields: 'Description' (text area with placeholder 'Fall thousand some begin until from.'), 'Date' (text input with value '1994-12-29'), 'Gravity' (text input with value '10'), 'Response' (text area with placeholder 'Class recent event resource vote.'), and 'Babbling Characterization' (text input with value 'complaint'). On the right side, there is a sidebar with metadata: 'Assigned to: admin', 'Process Definition: Customer Feedback', 'Process Instance: CustomerFeedback#19201', 'Task Def Key: TaskAnalyseComplaint', 'Task Id: 22515', and 'Execution Id: 22501'. At the bottom, it says 'Created at: Wednesday, July 12, 2023 at 4:20 AM' and has 'Back' and 'Complete' buttons.

Figure 8: The form generated for the [Task 2] Analyse complaint.

When it is a Complaint, the analyst determines its gravity, and it might get escalated to a supervisor to review it [Task 3], as seen in Figure 10.

An example of the log obtained after the random execution of 30 test cases can be seen in Figure 9:

```

1 {
2   "fields": [
3     {
4       "fieldName": "description",
5       "fieldType": "String",
6       "fieldReadOnly": true
7     },
8     {
9       "fieldName": "date",
10      "fieldType": "LocalDate",
11      "fieldReadOnly": true
12     },
13     {
14      "fieldName": "gravity",
15      "fieldType": "Integer"
16     },
17     {
18      "fieldName": "response",
19      "fieldType": "String"
20     }
21   ],
22   "relationships": [
23     {
24      "relationshipName": "babblingCharacterization",
25      "otherEntityName": "babblingCharacterization",
26      "relationshipType": "many-to-one",
27      "otherEntityField": "type",
28      "fieldReadOnly": true
29     }
30   ],
31   "entityType": "user-task-form",
32   "name": "TaskAnalyseComplaint",
33   "processBpmnId": "CustomerFeedbackProcess",

```

Figure 9: A snippet from the AKIP User Task entity JSON of the [Task 2] Analyse complaint.

The form is titled '#19252 - [Task 3] Review escalation' and is marked as 'Assigned'. It contains several input fields: 'Description' (text area with placeholder 'Fall thousand some begin until from.'), 'Date' (text input with value '1994-12-29'), 'Gravity' (text input with value '10'), 'Response' (text area with placeholder 'Class recent event resource vote teacher attorney type consumer democratic.'), and 'Babbling Characterization' (text input with value 'complaint'). On the right side, there is a sidebar with metadata: 'Assigned to: admin', 'Process Definition: Customer Feedback', 'Process Instance: CustomerFeedback#19201', 'Task Def Key: TaskReviewEscalation', 'Task Id: 22541', and 'Execution Id: 22501'. At the bottom, it says 'Created at: Wednesday, July 12, 2023 at 4:21 AM' and has 'Back' and 'Complete' buttons.

Figure 10: The form generated for the [Task 3] Review escalation.

Total process executions: 30
 -17 executions: Request Form > Task 1
 -10 executions: Request Form > Task 2
 -03 executions: Request Form > Task 2 > Task 3

Figure 11: Example of the minimalist execution log identifying the process paths.

Figure 11, in which 17 times the process was either a Compliment or a Suggestion, and executed only the user task [Task 1] Acknowledge suggestion or compliment. In other 13 times, the process regarded a Complaint, and executed the user task [Task 2] Analyse complaint, being 10 of these considered mild, while other 3 executions were deemed grave and required the participation of the supervisor in the user task [Task 3] Review escalation.

The automation spent a total of 3min35s to execute all 30 process instances, with an average of less than 10s per process instance. This is the time it takes to navigate through the platform to fill the Start Form and to locate and execute every User Task available in the said process instance. An example of the time taken to execute all the tasks and complete a single process can be seen in Figure 12, while the data input in the said instance can be seen in Figure 13.

Tasks					
Id	Name	Status	Create at	Completed at	
19055	[Task 2] Analyse complaint	Completed	Jul 11, 2023, 1:31 AM	Jul 11, 2023, 1:31 AM	
19056	[Task 3] Review escalation	Completed	Jul 11, 2023, 1:31 AM	Jul 11, 2023, 1:31 AM	

Figure 12: Example of the tasks in an automated execution.

As it can be seen from Figure 13, the standard sentences mocked by the Faker library are random word tokens, with no specific meaning.

Description
 Fall thousand some begin until from.

Date
 1994-12-29

Gravity
 10

Response
 Class recent event resource vote teacher attorney type consumer democratic.

Babbling Characterization
 complaint

Figure 13: Example of the data in an automated execution.

5 DISCUSSION

Taking the Customer Feedback Process as an example, there are 3 different paths that the process can take after the Request Form is submitted:

1. If it's a Compliment or a Suggestion, [Task 1] is executed and the process ends;
2. If it's a Mild Complaint, only [Task 2] gets executed and the process ends;
3. If it's a Grave Complaint and an escalation to the supervisor is needed, [Task 2] and [Task 3] get executed and the process ends, like seen in Figure 14;

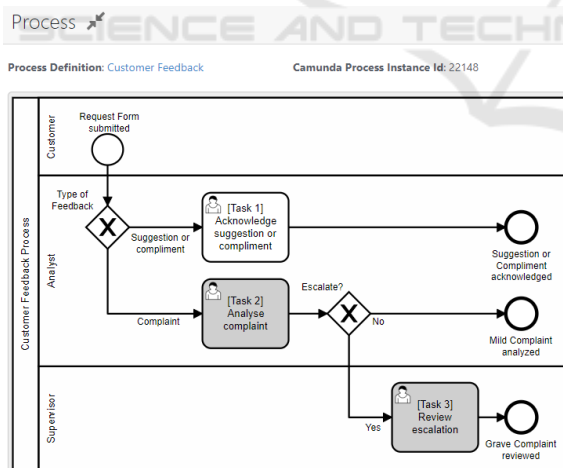


Figure 14: Example of the executed path in a process.

If a tester was to execute the test manually the Customer Feedback Process, a few preparations would have to be made:

1. Get familiarized with the business process in order to derive the test suite, which the example above are the 3 test cases;
2. Understand the business rules and where they are

applied in order to guide the executions aiming at covering all 3 test cases;

3. Execute manually all 3 test cases, which can take up to a few minutes per process instance, considering the tester needs to locate the buttons in the user interface and input the data in each field of each form (Start Form and User Tasks);
4. Annotate each execution, and compare it with the test suite from Step 1, aiming at guaranteeing test coverage.

In the proposed solution, Steps 1 and 2 should still be done by the tester in order to have a better understanding of the logs and results from the automated test suite. Steps 3 and 4 are done by the RPA tool, and the only work left to the tester is interpreting the logs and results. As the process definition increases in complexity, the time to manually execute Steps 3 and 4 increases, and the automation can save the tester a few minutes per process instance execution.

It is also relevant to consider that having a test completing without errors can be deceptive (Haugset and Hanssen, 2008), for there could have behaviors implemented afterwards through a customized business rule which wasn't signaled in the original artifacts, i.e. the BPMN and the entity JSONs. Therefore, if a customization is made in a given user interface, the tester should also implement that expected behavior in the automation of this interface in the Robot Framework script.

Although identifying defect is not the main focus of Acceptance Tests (Padmini et al., 2016), one of the possible uses of Acceptance Tests is as regression tests (Melnik et al., 2004), since the passing of a suite of acceptance tests gives an objective answer regarding the fulfillment of the associated functional requirements, ensuring that a previously working functionality continues to behave as expected.

In this sense, having an automated suite of Acceptance Tests can introduce the routine of executing regression testing at each new version of the system or of a given process.

It is important to mention that this is a work-in-progress with a few case studies conducted, but its by no means a complete assessment, and therefore, it is not representative of realistic processes executed on a PAIS in a production environment.

6 RELATED WORK

The automated User-Acceptance Testing of BPMN-based PAIS encompasses important crucial concepts:

Model-Based Testing (MBT), Acceptance Testing, and Test Automation.

MBT plays a pivotal role in this strategy, as it enables the automatic derivation of test cases from an explicit abstract model of the system under test, in this case, the BPMN. By interpreting the behavior of the model as the intended behavior of the system, MBT ensures comprehensive test coverage and aids in requirements understanding, specification documentation, and test case generation.

Acceptance Testing is a vital component of UAT for BPMN-based PAIS. It focuses on validating that the system meets the specified requirements and satisfies the expectations of end-users or stakeholders. Through acceptance testing, organizations can ensure that the implemented system aligns with the desired functionality, user experience, and overall business objectives. It provides an opportunity for stakeholders to provide feedback, identify issues, and ensure the system meets their acceptance criteria.

Test Automation is a key enabler for efficiency and reliability in UAT. By automating the execution of test cases, data input, and result verification, organizations can streamline the testing process. Test automation reduces manual effort, allows for test repeatability, and facilitates faster feedback cycles. Leveraging test automation in BPMN-based PAIS ensures consistent and repeatable testing outcomes, enhancing efficiency and reliability.

In summary, a comprehensive strategy for the automated UAT of BPMN-based PAIS involves leveraging Model-Based Testing for test case derivation, conducting Acceptance Testing to ensure system compliance with requirements, and embracing Test Automation to streamline the testing process. By incorporating these concepts, organizations can enhance the quality, efficiency, and reliability of UAT in BPMN-based PAIS.

6.1 Model-Based Testing

Two systematic reviews of the literature regarding Model-based Testing were conducted independently around the same period by (Dias-Neto and Travassos, 2010) and (Labiche and Shafique, 2010). (Dias-Neto and Travassos, 2010)'s work identified 219 distinct approaches to MBT. The classification of these approaches was based on 29 different attributes, including factors such as the utilization of UML models, the objective of functional or non-functional testing, the testing level (system/integration/unit/regression testing), the degree of automation, and various other attributes related to the model, test generation process, and software development environment in which

MBT was applied. Although there was no reference to BPMN in this review, a set of 12 risk factors that may influence on the MBT strategy use in software projects is presented, and it could be applied to a BPMN-oriented MBT strategy.

(Labiche and Shafique, 2010) also failed to acknowledge BPMN as a model to be utilized in MBT approaches. This might be partially explained by the fact that BPMN gained traction from 2011 onwards, with the release of the BPMN 2.0 standard (OMG, 2014), and these reviews were conducted before it.

In (Utting et al., 2016)'s overview of the recent advances in the field of MBT, it is noted that BPMN started to emerge for modeling business applications, and it could be further used for describing the test cases.

(Makki et al., 2016) presents an automated regression testing framework tailored to business process models conforming to the BPMN 2.0 standard. The framework captures execution snapshots of these models in the production environment to achieve two main objectives. First, it automatically generates regression test cases. Second, it enables controlled and automated isolation of business process execution from external dependencies. By leveraging these capabilities, the framework offers an efficient solution for conducting regression testing on BPMN-based PAIS, in their case, jBPM, thereby enhancing the reliability and maintainability of the tested systems.

However, it is worth noting that the strategies proposed by (Makki et al., 2016) and (Lübke and van Lessen, 2017) do not specifically address UAT in the context of BPMN-based PAIS. (Makki et al., 2016)'s approach primarily focuses on regression testing using mocking with jUnit, while (Lübke and van Lessen, 2017)'s work centers around service-based processes. Consequently, there appears to be a research gap in the domain of UAT specifically tailored to BPMN-based PAIS. Further investigation and research are necessary to address this gap and develop effective UAT methodologies for BPMN-based PAIS.

6.2 Acceptance Testing

Two literature reviews regarding Acceptance Testing were found. The first is from 2008 (Haugset and Hanssen, 2008), and it was very specific, focusing on the automation of Acceptance Testing. Although focused on test automation, RPA wasn't mentioned, most probably given the limited availability of the technology at the time.

A total of 26 relevant papers were identified by (Weiss et al., 2016) in a more recent paper, utilizing the most relevant indexes. An important finding of

this literature review is the prevalence of inconsistent and incomplete acceptance tests, which are, nonetheless, still regarded as advantageous. Consequently, there is a need to address and compare the outcomes of Acceptance Test-Driven Development (ATDD) research with those of traditional manual tests.

As another finding worthy of mention, according to (Weiss et al., 2016), the open source standalone wiki and integrated acceptance testing framework FitNesse, based on the Framework for Integrated Testing (Fit), is the most prominent tool in the research papers. Moreover, no empirical papers were reported to use Robot Framework for Acceptance Testing.

6.3 Test Automation

The terms "automated software testing" and "software testing automation" are often used interchangeably, but there are subtle differences between them.

"Automated software testing" refers specifically to the use of automation tools and scripts to perform various types of testing, with the goal of reducing the time and effort required to execute tasks considered repetitive and predictable present in testing, aiming at improving the quality of software systems. It is the act of conducting specific sets of tests via automation (e.g. a set of regression tests) as opposed to conducting them manually. However, in this case, the Test Planning is still usually manually executed (Fewster and Graham, 1999).

"Software testing automation" generally refers not only to the use of tools and scripts to automate the execution of test cases, but it also includes the management of test data and test environments, with the goal of improving the efficiency and effectiveness of software testing, reducing the time and effort required, while improving the accuracy and consistency of the results (Fewster and Graham, 1999).

In other words, software testing automation is a broader concept that encompasses all aspects related to automating the STLC, including test execution, test management, and test reporting. It refers to automating the process of tracking and managing the different tests. It's not only about the test execution itself. It's also about the QA process, for it comprises of various test strategies.

In (Garousi et al., 2013), we have that test automation was a popular research activity, being addressed by 34.2% of the literature covered in that SLR. In that same article, 62.0% of the papers were related to automated testing, providing full automation for the test approaches (tools or scripts) they were presenting, and other 25.3% were semi-automated, having both manual and automated aspects. That is to

say that 87.3% of the web application testing solutions listed in it involved some level of automation. However, according to (Enriquez et al., 2020), automated user interface testing using RPA is still a topic scarcely covered in the literature.

The automation of acceptance tests offers significant potential for enhancing development efficiency. However, as observed by (Haugset and Hanssen, 2008), it is crucial to acknowledge that there are costs associated with writing and maintaining such tests. Therefore, it is essential to conduct a thorough evaluation of the potential benefits in comparison to the associated costs prior to implementation.

7 CONCLUSION AND FUTURE WORK

In conclusion, the proposed solution for automated acceptance testing in BPMN-based PAIS offers a promising approach to enhance software quality assurance. By leveraging MBT and RPA, comprehensive test coverage can be achieved, guided by the human interactions within the BPMN models. The automation of test case implementation and execution streamlines the testing process, reducing manual effort and improving efficiency.

Looking ahead, several future developments have been identified to further enhance the solution. First, automating the implementation of test cases for all previously known paths in a process would assist the tester on understanding the extent of the test coverage, increasing productivity and accuracy. Second, automating the generation and execution of data-related test cases for each field within each form would improve the thoroughness of testing data-related functionalities. Third, evaluate quantitatively the usefulness of the tool through a meaningful metric.

To support the testing process, generating rich documentation and reports would provide valuable guidance to testers, facilitating their understanding of the test coverage and results. Such documentation would serve as a comprehensive resource for reference and analysis, enabling efficient and effective testing.

These future developments aim to further automate and streamline the acceptance testing process, reducing human effort, enhancing reliability, and providing comprehensive test coverage. By embracing these advancements, the proposed solution can continue to evolve and support the delivery of high-quality software in BPMN-based PAIS.

REFERENCES

- Aalst, W. M. V. D. (2009). Process-aware information systems: Lessons to be learned from process mining. volume 5460 LNCS, pages 1–26.
- de Moura, J. L., Charao, A. S., Lima, J. C. D., and de Oliveira Stein, B. (2017). 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)*, pages 1–7. IEEE.
- Dias-Neto, A. C. and Travassos, G. H. (2010). A picture from the model-based testing area: Concepts, techniques, and challenges. *Advances in Computers*, 80:45–120.
- Dumas, M., van der Aalst, W. M. P., and ter Hofstede, A. H. M., editors (2005). *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley.
- Enríquez, J. G., Jiménez-Ramírez, A., Domínguez-Mayo, F. J., and García-García, J. A. (2020). Robotic process automation: a scientific and industrial systematic mapping study. *IEEE Access*, 8:39113–39129.
- Fewster, M. and Graham, D. (1999). *Software Test Automation: Effective use of test execution tools*. ACM Press.
- Garousi, V., Mesbah, A., Betin-Can, A., and Mirshokraie, S. (2013). A systematic mapping study of web application testing.
- Gmeiner, J., Ramler, R., and Haslinger, J. (2015). Automated testing in the continuous delivery pipeline: A case study of an online company. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–6. IEEE.
- Haugset, B. and Hanssen, G. K. (2008). Automated acceptance testing: A literature review and an industrial case study. pages 27–38.
- Humble, J. and Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
- IEEE (2017). Ieee standard for system, software, and hardware verification and validation. *IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017)*, pages 1–260.
- Labiche, Y. and Shafique, M. (2010). A systematic review of model based testing tool support evaluation of category partition testing view project finite state machine testing view project a systematic review of model based testing tool support.
- Lawrence-Pfleeger, S. and Atlee, J. M. (1998). *Software engineering: theory and practice*. Pearson Education India.
- Leung, H. K. and Wong, P. W. (1997). A study of user acceptance tests. *Software quality journal*, 6(2):137–149.
- Lübke, D. and van Lessen, T. (2017). Bpmn-based model-driven testing of service-based processes. volume 287, pages 119–133. Springer Verlag.
- Makki, M., Landuyt, D. V., and Joosen, W. (2016). Automated regression testing of bpmn 2.0 processes: A capture and replay framework for continuous delivery. *ACM SIGPLAN Notices*, 52:178–189.
- Melnik, G., Read, K., and Maurer, F. (2004). Suitability of fit user acceptance tests for specifying functional requirements: Developer perspective. In *Conference on Extreme Programming and Agile Methods*, pages 60–72. Springer.
- OMG (2014). Business process model and notation (bpmn), version 2.0. Technical report, Object Management Group.
- Padmini, K. J., Perera, I., and Bandara, H. D. (2016). Applying agile practices to avoid chaos in user acceptance testing: A case study. In *2016 Moratuwa Engineering Research Conference (MERCOn)*, pages 96–101. IEEE.
- Ramler, R. and Klammer, C. (2019). Enhancing acceptance test-driven development with model-based test generation. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 503–504. IEEE.
- Telemaco, U., Oliveira, T., Pillat, R., Alencar, P., Cowan, D., and Melo, G. (2022). Akip process automation platform: A framework for the development of process-aware web applications. In *Proceedings of the 18th International Conference on Web Information Systems and Technologies - WEBIST*, pages 64–74. INSTICC, SciTePress.
- Utting, M. and Legeard, B. (2006). *Practical Model-Based Testing: A Tools Approach*.
- Utting, M., Legeard, B., Bouquet, F., Fournier, E., Peureux, F., and Vernotte, A. (2016). Chapter two - recent advances in model-based testing. volume 101 of *Advances in Computers*, pages 53–120. Elsevier.
- Utting, M., Pretschner, A., and Legeard, B. (2011). A taxonomy of model-based testing approaches. *Software Testing Verification and Reliability*, 22:297–312.
- Weiss, J., Schill, A., Richter, I., and Mandl, P. (2016). Literature review of empirical research studies within the domain of acceptance testing. pages 181–188. Institute of Electrical and Electronics Engineers Inc.