Automated testing in robotic process automation projects

Andres Jiménez-Ramírez[1] | Jesús Chacón-Montero[1] | Tomasz Wojdynsky[2] | José González Enríquez[1]

[1]Computer Languages and Systems Department, University of Seville, Seville, Spain
[2]Faculty of Management, Economics and Computer Science, The School of Banking and Management, Cracow, Poland

Correspondence
A. Jiménez-Ramírez, Computer Languages and Systems Department, Escuela Ténica Superior de Ingeniería Informática, Avenida Reina Mercedes, s/n, 41012 Seville, Spain.
Email: ajramirez@us.es

Funding information
Centre for Industrial Technological Development, Grant/Award Number: P114-16/E09; Secretaría de Estado de Investigación, Desarrollo e Innovación, Grant/Award Number: TIN2016-76956-C3-2-R

Robotic process automation (RPA) has received increasing attention in recent years. It enables task automation by software components, which interact with user interfaces in a similar way to that of humans. An RPA project life cycle is closely resembling a software project one. However, in certain contexts (e.g., business process outsourcing), a testing environment is not always available. Thus, deploying the robots in the production environment entails high risk. To mitigate it, an innovative approach to automatically generate a testing environment and a test suite for an RPA project is presented. The activities of the humans whose processes are to be robotized are monitored and a UI log is confirmed. On one side, the test environment is generated as a fake application, which mimics the real environment by leveraging the UI log information. The control flow of the application is governed by an invisible control layer that decides which image to show depending on the interface actions that it receives. On the other side, the test case checks whether the robot can reproduce the behaviour of the UI log. Promising results were obtained and a number of limitations were identified such that it may be applied in more realistic domains.

KEYWORDS
automated testing, robotic process automation

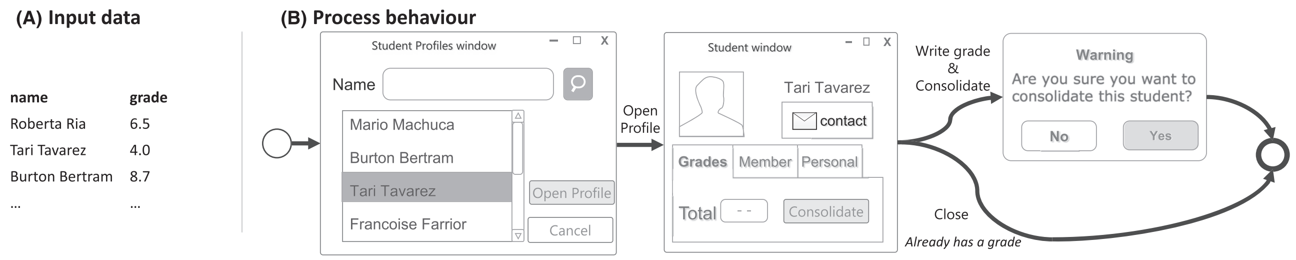
1 | INTRODUCTION

Robotic process automation (RPA) is a software solution for the creation of programs that mimic the behaviour of human workers when performing repetitive and structured tasks with information systems (ISs).[1-3] This solution has been applied in many industries and contexts. However, several authors have acknowledged that the best candidates to conduct a successful RPA project are the processes within the back offices of a company[4,5] since they (a) are highly frequent, (b) lack excessive exception control, (c) require limited cognitive effort, and (d) are prone to human errors.[1]

There are many vendors who offer out-of-the-box solutions to deal with RPA, such as BluePrism,[6] UIPath,[7] or WorkFusion.[8] In general, such tools share a common RPA life cycle although each tool provides different support to each phase. The life cycle starts with the analysis of the candidate process for automation. The processes are then designed to contain elements such as actions or dataflow that must be coded. Subsequently, robots are constructed in accordance with the design. They are then deployed in individual environments (eg, virtual machines) to perform their tasks. During the deployment phase, the robots are controlled and monitored in their operation (eg, to start new robots and stop them in case of serious errors). Finally, the performance and error cases of the robots are evaluated to enable a new analysis for the enhancement of the robots. It is important to bear in mind that the testing phase is missing since it heavily depends on each RPA project.

In the traditional software life cycle, testing is performed in a testing environment before deployment in the production environment. Unlike traditional software, such a testing environment is seldom available in RPA, which entails a high risk for deployment. This is a common setting in the business process outsourcing (BPO) context, where companies execute processes that belong to another company. In these situations, companies about to adopt RPA start performing the processes with human workers rather than robots. For this reason, such employees receive training to learn how to manage cases, and they also learn through solving the exceptions that occur on a daily basis. In a parallel way, the RPA project starts by automating the simplest cases by leveraging the knowledge that the human workers have generated. In this paper, we propose leveraging such information to automatically test the constructed robots. Note that, although the paper is focused on the BPO context, it can be geeralized to any other context where a list of similar tasks is to be done over a system that lacks a testing environment.

To this end, a method to automatically generate a testing environment and a test suite for RPA projects is proposed. The first step that must be executed involves controlling the input data that the user needs to work (eg, a list of emails with a specific subject and a spreadsheet file with



name	grade
Roberta Ria	6.5
Tari Tavarez	4.0
Burton Bertram	8.7
...	...

FIGURE 1 Running example

id	timestamp	action	window
12	2018/01/10-0:4:32	Text “Tari Tavarez”	Student profile
13	2018/01/10-0:4:35	Left click on (120,205)	Student profile
14	2018/01/10-0:4:50	Left click on (10, 240)	Student
15	2018/01/10-0:5:05	Text “Burton Bertram”	Student profile
16	2018/01/10-0:5:09	Left click on (115,206)	Student profile
...	...	...	...

FIGURE 2 An example interaction with the system

client information). The computers of the human workers who are performing the processes to robotize and gather this behaviour in the form of a UI log[9] are monitored. This log contains the user interactions with screen captures for each case of the input data. The test environment can then be generated as a fake application, which mimics the real IS, that consists of a set of images and an invisible control layer over such images. This invisible control layer oversees capturing robot inputs (ie, mouse clicks and keystrokes) and deciding the next image to show depending on such inputs. In addition, this layer controls the state of the test, ie, *pass* or *fail*.

Once the generated testing environment is ready, the constructed robots are deployed therein and stimulated with the input data of the test cases (this step lies outside the scope of this paper). Finally, a test report is generated by the control layer.

A preliminary prototype has been evaluated in controlled scenarios, and the results seem promising. This innovative idea opens several research lines regarding the current limitations.

This paper is an extension of a previous work[10] incorporating (a) the formalization of test cases and test suites, (b) the modification of the approach and algorithm to include test suites, (c) the conducting of an evaluation of the platform, and (4) an extensive related work section. The rest of the paper is organized as follows: Section 2 describes a running example that will be used to guide and illustrate the various points of this proposal. Section 3 contextualizes the proposal, provides details of the RPA paradigm, and focuses on how it affects the aforementioned BPO context. Section 4 details the approach by taking the running example described in Section 2 as a reference. Section 5 describes the preliminary evaluation that has been conducted. Section 6 presents the closest studies related to the proposal of this paper found in the literature. Finally, Section 7 summarizes the conclusions obtained after developing this approach and proposes several new research lines.

2 | RUNNING EXAMPLE

To contextualize the approach, this section describes a simplification of the process of a teacher that has to perform the consolidation of the results of the exams that she has marked on her institution website. To this end, a set of student marks are needed (such as the spreadsheet in Figure 1A), which can be considered as the input data. Each student is dealt with as a different case. For each student, the teacher behaves as follows (Figure 1B). First, she looks for the student in the Student Profiles window. She then has to open the student profile and check whether this student already has a mark (eg, another teacher did it before). If it is not the case, the mark must be entered and then consolidated. A warning window is shown to confirm the action. In the case that the student already has a mark, the teacher simply closes the window. In both cases, the application returns to the student profile window.

Figure 2 shows an excerpt of the interaction of the teacher with the system. As can be observed, in interactions 12 to 14, the teacher is dealing with the case of the student “Tari Tavarez.” She first enters the student name, then left-clicks on the coordinates 120, 205 (ie, the “Open Profile” button), and finally closes the Student window (ie, clicks on 10, 240) after having seen that this student already has a mark. The teacher then proceeds with the subsequent student case.

This simplified process is repetitive and is, therefore, a perfect candidate for robotization since the interactions with the user interface can be performed by a software robot. Following the construction of such a robot (which lies outside the scope of this paper), its behaviour must be tested. However, the only platform available for said testing is the real-life system and, testing in this environment, therefore, entails a high risk. For this reason, our approach shows how to generate a testing platform for robotics projects.

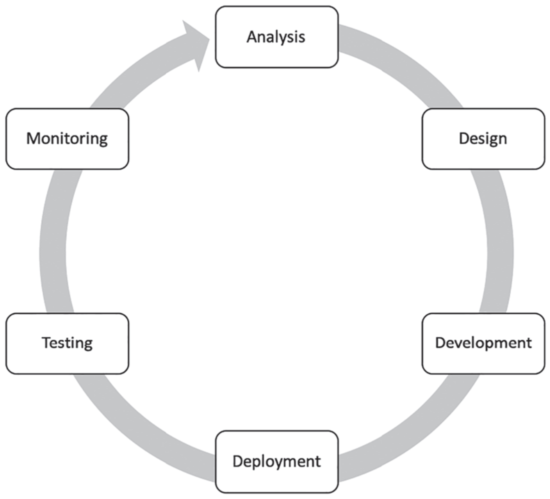


FIGURE 3 Typical robotic process automation (RPA) life cycle

3 | BACKGROUND

3.1 | Robotic process automation

The ever-expanding RPA concept is built upon three main ideas: robot, process, and automation.[11] While *robot* means whatever software machine can be programmed to execute a specific job, the *process* is about a succession of steps to reach a particular objective or result. Lastly, *automation* means a transformation of a manual operation with the purpose of enabling it to be performed in an autonomous way.

Broadly speaking, a robot neither needs to be autonomous nor to follow a process (eg, it could be commanded by a human). When those three elements join together, then RPA arises. Therefore, in this context, a robot is a software product created to perform a specific task. This task is a process that can be performed autonomously by the robot. The main objective is to minimize costs, provide agility, and improve the quality[12] of processes that have hitherto been performed by a human team.[13]

For the construction of such robots, an RPA project must be conducted following a life cycle that involves six main steps (cf Figure 3): (a) analysis, (b) design, (c) development, (d) deployment, (e) testing, and (f) maintenance and operation. In the analysis step, various techniques are applied to successfully understand the business process to be robotized. In the design and development steps, the developers must define the process in order to translate it and build the robot. This robot would carry out the process that has been formalized in the analysis step. Although a test environment is desirable for testing purposes and may be available in certain contexts, it should be borne in mind that general RPA contexts (eg, the BPO context) are characterized as lacking a test environment and only the production environment is available for such purposes.[14] Therefore, when the robot is implemented, it is deployed in production and then tested to determine whether it is behaving correctly. Once checked, the robot is then maintained and operated. In this phase, its performance is measured, and the robot can be stored, started, and/or replicated depending on its state and/or on the current demand. Once checked, the robot is maintained, and it operates to measure its performance and to start, stop, or replicate it based on its state or the demand.

As the reader may have noticed, this life cycle presents one main drawback: The testing phase is executed directly in the production environment that places production at high risk.

3.2 | RPA in the BPO context

Several authors, through practical experience, acknowledge that the best process candidates to guarantee a successful RPA project are processes within the back office of a company[4,5] since they are highly repeatable, with a clear workflow, require low-cognitive effort, and prone to errors[1](eg, the running example stated in Section 2).

RPA becomes particularly challenging in BPO scenarios since processes are outsourced, in that, the back office and the ISs to interact with may be geographically dispersed. An outsourced process implies that the company remains unaware of how the process will be executed. The BPO company is responsible for deciding whether a robot should be developed for any particular process. However, when a robot is to be implemented, there are some cases where the robot development team have no access to a test environment.

In this context, a partnership with the Servinform S.A.[*] company has been established to improve the current RPA life cycle.[15] In Servinform and some BPO setting in general, the back office consists of both human and robot teams. They typically perform the following process when an outsourced process is to be managed (cf Figure 5A). First, human workers are trained (cf Figure 5B) to perform such a process. These workers then start to deal with the real ISs according to their training (cf Figure 5C). It is at this moment when the RPA life cycle starts. To leverage the knowledge that the human workers have acquired, their computers are monitored to obtain a log of the interactions with the user interfaces (ie,

[*] Servinform is a Spanish BPO company with an IT consulting area.

start_ts	event_type	keystrokes	click_type	click_coords	img_name
2018/01/10-0:4:32	keystroke	“Tari Tavarez”			img_12
2018/01/10-0:4:35	click		Left	120,205	img_13
2018/01/10-0:4:50	click		Left	10,240	img_14
2018/01/10-0:5:05	Keystroke	“Burton Bertram”			img_15
2018/01/10-0:5:09	click		Left	115,206	img_16
...	...	...			...

FIGURE 4 Running UI log. Only some relevant columns are shown

TABLE 1 Event attributes captured in the UI log

Name	Description
app_name	This attribute refers to the name of the application that is being used.
focus	This attribute indicates whether the application has just gained the focus.
event_type	This attribute refers to the kind of events that can be produced $\{click, keystroke\}$.
keystrokes	This attribute refers to the sequence of keystrokes pressed when *event_type* is a keystroke.
click_type	This attribute refers to the kind of click that has been produced $\{left, right, middle\}$ when *event_type* is *click*.
click_coords	This attribute refers to the coordinates of the click when *event_type* is *click*.
start_ts	This attribute refers to the start timestamp.
end_ts	This attribute refers to the end timestamp.
img_name	This attribute refers to the name of the file that contains the screen capture.

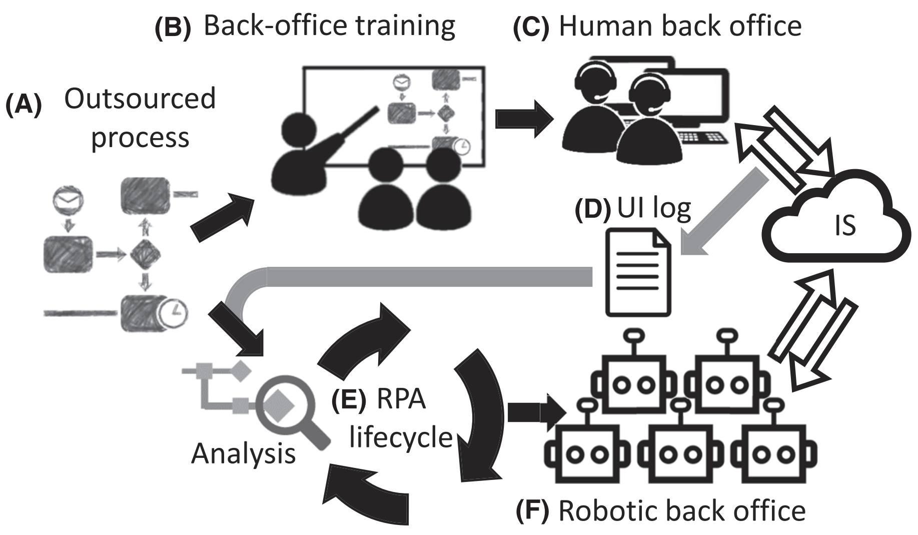


FIGURE 5 Robotic process automation (RPA) in the business process outsourcing (BPO)

a UI log)[9,15] (cf Figure 5D). Such a UI log consists of a stream of events, which contains clicks, keystrokes, screen captures, and further relevant information (cf Table 1) for the UI log information proposed by Servinform (cf Example 1).

Example 1. Considering the running example and the interaction of Figure 2, Figure 4 shows the information of the corresponding UI log. Note that the information related to the window, which is shown in Figure 2, is lost in Figure 4, but the image of the screen capture appears (cf img_name column).

This log provides invaluable information for the analysis of the underlying outsourced process and the enhancement of a successful RPA life cycle (cf Figure 5F). As a result, a robotic team is deployed to interact with the same ISs in parallel with the human team.

4 | APPROACH

This section describes our proposal to automatically generate a testing platform for an RPA project in the early phases of the RPA life cycle.

As evidenced in Figure 6, this method changes the original life cycle in certain new aspects. Essentially, it considers the inclusion of two new steps in the life cycle: the test environment construction (cf Section 4.1) and the automatic testing steps (cf Section 4.2).

4.1 | Test environment construction

The first step of our proposal consists of building a platform that mimics the behaviour of the real system with which the robots will interact. To this end, a generic test environment is constructed as a UI controller.

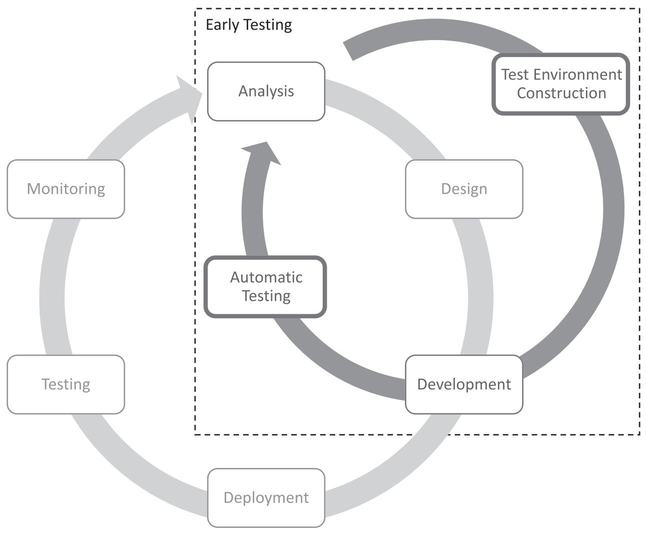


FIGURE 6 Modified robotic process automation (RPA) life cycle

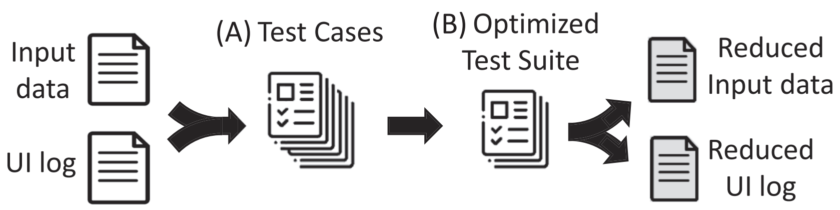


FIGURE 7 Process to select the subset of test cases

Definition 1. A **UI controller** is a software system that allows for two main functionalities:

1. Showing a view: The UI controller shows a view (ie, images) working as the background of the application.
2. Waiting for events: The UI controller waits until a UI event occurs (ie, clicks and keystrokes). For this, in front of the views, an invisible layer captures all the UI events.

The test environment leverages the UI controller to simulate any application, by showing images, and checking if the behaviour is correct, by checking the occurred events. To this end, the screen captures and the event details of the UI log (cf Table 1), which is provided in the analysis phase, are used as the basis.

However, a UI log contains events related to several cases, eg, the UI log of the running example (cf Figure 4) contains events for uploading the grades of two different students. Generally, many of these cases may imply executing a similar behaviour of the application and, hence, of the robot code. Since testing is a frequent activity in software development, it is important to keep just the relevant test cases in the test suites to avoid slowing doing the development process[16]; ie, a subset of the input data and the UI log must be selected.

Figure 7 depicts the process, which is followed to perform this selection. First, the UI log should be analysed to separate it in different cases (cf Figure 7A). In this paper, we base upon Jimenez-Ramirez et al[15] to discover the process behaviour that is behind the UI log. At a glance Jimenez-Ramirez et al,[15] uses process mining techniques to assign a *case_id* and a *variant_id* to each event. The *case_id* identifies the events that are related to the same case; eg, all the events of uploading the grade of "Tari Tavarez" will share the same *case_id*. And the *variant_id* indicates process variant that follows each case; eg, for the running example of Figure 1, two different variants exist—the two branches of the process—and, therefore, each case—student—will follow one of these variants. Then, an optimized test suit might consist of only one test case for each variant[†] (cf Figure 7 B).

Thereafter, the reduced subset of the input data and the UI log related to such an optimized test suit are gathered.

4.2 | Automatic testing

When the developer team completes the construction of robots, our proposal can be used to automatically test these robots before continuing with the life cycle. To this end, a robot must be deployed to interact with the test environment described above with the selected input data as that used while creating the UI log (cf Figure 8), ie, the test suite.

The automatic testing process, which is described in Algorithm 1, receives the aforementioned UI log and the UI controller (ie, the test environment). The objective of this algorithm is to compare the actions stored in the UI log against each action that a robot performs. The result of such a comparison produces a test report that collects all existing differences. The comparison between the data of two events is detailed in

[†]This selection corresponds to the minimum set of tests, which covers all the behaviour; however, different selection policies can be followed.

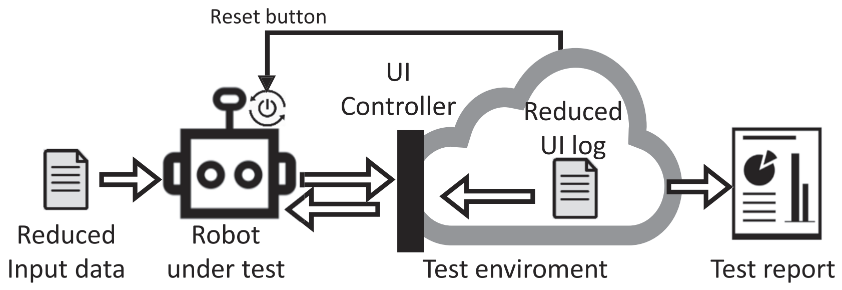


FIGURE 8 Automatic testing process

Algorith 2. This function receives two events and returns a boolean, which indicates if the events are considered equals or not. The first step of the function *similarEvents* is to check if the robot event (*cf recEvent*) is a keystroke or a click. Relying on the kind of event, the comparison will be between two string—which is trivial—or between two clicks. In general, there are two main ways to check if a click is similar to another one. The first one is simply by checking if the coordinates are the same. This is the simplest way to do it but also the strictest one. This strictness could be improved adding a threshold (eg, a given number of pixels). The second one is by recognizing where the clicks do the action (eg, the bounding box of a button) and check if they are the same or equivalent. Algorithm 2 describes the pseudocode of such a procedure; however, the details for detecting the click area on a general basis lies outside of the scope of this paper.[‡]

Algorithm 1 Test execution (**input:** UILog suite, UIController c, **output** TestReport)

```
1:  report ← Ø
2:  for each case : cases(suite) do
3:      succEvents ← Ø
4:      failFlag ← false
5:      for each expEvent : case do
6:          c.showView(expEvent.getImage())
7:          recEvent ← c.waitForEvent()
8:          if similarEvent(expEvent,recEvent) then
9:              succEvents.add(recEvent)
10:         else
11:             report.add(createFailReport(succEvents,expEvent,recEvent))
12:             failFlag ← true
13:             break
14:         end if
15:     end for
16:     if failFlag == false then
17:         report.add(createPassReport(succEvents))
18:     end if
19:     resetRobot()
20: end for
21: return report
```

Algorithm 2 Similar events (**input:** Event expEvent, Event recEvent, **output** boolean)

```
1:  similar ← false
2:  if isKeystroke(expEvent) then
3:      similar ← (expEvent.keystrokes == recEvent.keystrokes)
4:  else
5:      if expEvent.clickType == recEvent.clickType then
6:          clickArea1 ← getClickArea(expEvent.clickCoords)
7:          clickArea2 ← getClickArea(recEvent.clickCoords)
8:          similar ← equivalent(clickArea1, clickArea2)
9:      end if
10: end if
11: return similar
```

[‡]The current prototype implements the click area comparison just for text boxes and buttons using a simple edge detector algorithm. For the rest of UI elements (ie, when the edge detector fails), a 15-pixels threshold is used for the comparisons.

Test nº: 1	**Date: 2019/02/06**	**Duration: 12s.**	**Status: FAILED**
Trace:	2019/02/16-0:8:20	*Keystroke "Tari Tavarez"*	
	2019/02/16-0:8:32	***Failure***	
Cause:	Expected	*Click: left, 120, 205*	
	but received:	*Keystroke "Burton Bertran"*	

FIGURE 9 Example of a test report

The algorithm starts with an empty report where the successful and fail test reports will be stored (cf line 1). Then, each case log is obtained from the suite log (cf *cases* function in line 2). This case log contains a subset of the full UI log where all the events belong to the same case.

For each case log, an empty set of successfully performed events (cf line 3), and a flag indicating that there are no errors (cf line 4) are created. Then, the algorithm goes through the events in the case log (cf line 5) and requests the UI controller to show the image corresponding to the current event (cf line 6).

Example 2. Regarding the UI log of Figure 4, *img_12* is shown first in the automatic testing since it is the image of the first event.

The algorithm then waits until the robot performs any action[§] and, after that, the received event is stored (cf line 7) and it is verified whether it is similar to the current event in the UI log (cf line 8). In the case when the events are similar (eg, clicking on similar regions or introducing the same keystrokes, cf Algorithm 2), the received event is stored in the set of successful events (cf line 9), and the algorithm continues with the subsequent event in the case. In turn, if the event diverges, a *fail report* is created and stored in the reports, which details (a) which events have been successfully performed (ie, *succEvents*), (b) which events have failed (ie, *recEvent*), and (c) which events were expected (ie, *expEvent*, cf line 11). In addition, the fail flag of the case is set to indicate that the test case has failed (cf line 12).

Example 3. Regarding the running example, the first expected event is a text action containing "Tari Tavarez." Assuming the correct behaviour of the robot, the test environment will show the image of the subsequent event (ie, *img_13*) and wait for an event of the robot. In case the robot introduces a text, eg, "Burton Bertman", the test case will then be completed by producing the failure report of Figure 9 since a click event was expected instead of a text. Thanks to this report, the RPA developers can improve the behaviour of the robot to avoid wrong actions.

The algorithm continues until all the expected events of the case log are processed. When a case log is finished with no failures (cf line 16), a *pass report* is produced using the data stored in *succEvents* and included in the reports(cf line 17).

Since the robots under test might be left in an unknown state after performing a test case, a global function *resetRobot* is invoked. This abstract function would require some specific code out of the testing environment to link this invocation to an actual reset of the robots under test.

When no more cases remain in the suite, the report of the test suite is returned. Such a report may contain both types of case reports:

1. If all the events of a case are validated, it means that the robot has done all its work correctly and this fact can be seen in the pass report. At the same time, it means that the robot is ready to be deployed in the real environment and hence, the life cycle may continue. This pass report includes just (a) the number of the test, (b) the date, (c) the duration of the test, and (d) the status of the test (ie, SUCCESS).
2. If any event stops the process because it fails to match the expected events, then a fail report is included for this case stating that the robot is not ready for deployment and hence, the robot must be repaired before continuing the life cycle. In this case, the fail report includes: (a) the number of the test, (b) the date, (c) the duration of the test until the failure, (d) the status of the test (i.e., FAILED), (e) the trace of the robot under test, which states all the events that have performed, and (f) the cause of the failure.

5 | EVALUATION

To test the effectiveness of the proposal presented in this work, a set of test cases have been performed. On these tests, a robot and a human used the developed test platform. The aim of this experiment is to assess whether the proposal can effectively deploy a platform that can (a) simulate a real platform being able to cheat humans and robots and (b) detect both successful and failed execution of tests. Section 5.1 describes how these test cases have been designed, while Section 5.2 shows the data analysis and presents the conclusions of this evaluation.

5.1 | Setup

For this evaluation, a traditional web platform has been used. More precisely, it was used Moodle, an open-source learning web platform that is currently in version 3.7.2. Moodle offers a demo site[¶] with populated data, which is reset every hour so the content can be considered stable to perform repeatable tests. The process to be simulated using the proposal presented is based on the example shown in Section 2, ie, a process to include grades of several students in the system. To this end, these steps are followed:

[§]The *waitForEvent* function may return null if it waits too long for a robot interaction (eg, if the robot freezes).
[¶]https://school.moodledemo.net/, accessed 30/09/2019.

TABLE 2 Test cases specification

Test Case	No. of Events	Description
1	10	Access to the list of students and include three grades of them.
2	12	Grade 3 students navigating through the student profiles by using the save and continue functionality.
3	17	Grade 3 students navigating through the student profiles by saving grade and then going the next student.
4	21	Grade 3 students by looking for each one on the list of students and accessing to their student profile.

TABLE 3 Performance results

Test Case	Actor	%True+	*%True–*
1	Robot	100%	100%
	Human	86%	100%
2	Robot	100%	100%
	Human	84%	100%
3	Robot	100%	100%
	Human	79%	100%
4	Robot	100%	100%
	Human	65%	100%

1. Four test cases are designed through the Moodle website. Each of them consists of a set of actions (ie, clicks and keystrokes) over the Moodle site as detailed in Table 2.
2. Then, a UI log for each test case is generated by manually executing the case. An external tool is used for this aim.
3. After that, the generated UI logs are introduced in the proposed test platform.
4. Finally, the four test cases are executed in the platform by four different actors: (a) a *flawless* robot that is programmed to simply follow the UI log, (b) a *faulty* robot that is programmed to intentionally fail at some point of the process, (c) a *reliable* human who tries to fulfil the process in correctly, and (d) a *faulty* human who is instructed to intentionally fail at some point of the process.

To assess the effectiveness of our proposal, each test was run 30 times to evaluate the same metrics for both human and robot, ie, whether or not the test has passed. Specifically, for each test case and actor—robot or human—it was measured:

1. *%True+*: That is, the percentage of tests that have passed when they were executed by the *flawless* robot or by the *reliable* human.
2. *%True-*: That is, the percentage of tests that have failed when they were executed by the *faulty* robot or by the *faulty* human.

For the sake of simplicity, the *False+* and *False–* are not shown since they are the complementary values of *True+* and *True–*, respectively.

5.2 | Results and conclusion

Table 3 shows the results obtained after performing the executions of each test case using a robot.

As can be seen, the results obtained by the robots are merely perfect. As expected, the *flawless* robot (ie, the robot that is implemented to follow the UI log) passes the 100% of the tests (cf *%True+* column). Similarly, the *faulty* robot does not pass any test, which is the expected behaviour (cf *%True–* column). These two results mean that, independently of the size of the number of events (cf *test case* column), the proposed testing platform can easily (a) recognize the sequence of events that strictly follows the UI log and (b) detect any deviations from it.

Regarding the human behaviour, results show a similar behaviour related to the *faulty* human (cf *%True–* column); ie, all the test cases where the human intentionally fail were not passed. However, different results are obtained when testing the *reliable* human. Although this actor tried to correctly conduct the process, there were cases that did not pass. It can be noticed that the longer the case is, the higher the failures are too. That behaviour makes sense since the human has more opportunities to commit a failure. After analysing these test case reports, it has been detected that involuntary interactions against the fake UI trigger the failure of the tests in most of the cases, eg, clicking on the screen without any functional consequences. In some other cases, the human tried to fulfil the procedure in a slightly different way although it was functionally correct. Not surprisingly, the test case #4 achieved the lowest *%True+* since this case includes a step to look for the student by the name which entails producing more keystroke events than the rest of the cases.

Looking at these results, we can conclude that, due to the strict nature of the system, our proposal is very effective classifying actors as erroneous. However, some misclassifications have been detected when evaluating the human who tried to do well. Nonetheless, the platform is not intended to be used by humans although this result shows clear flexibility issues that might be addressed.

6 | RELATED WORK

The scope of application of RPA is diverse since it can be applied in environments such as front office and back office,[17-19] the aforementioned BPO,[19-21] health systems,[22,23] eCommerce or facial recognition,[23] and finance.[12,24] However, to the best of our knowledge, there are just a few proposals that speak about how to test the robots implemented.

Le Clair et al[25] summarize several best practices and provide guidance for the prioritization of processes for the analysis phase. However, this work lacks any technical support for the testing phase. From a more technical point of view, Leopold et al[26] propose the use of natural language processing and machine learning for the detection of candidate activities from the textual description of processes. In turn, Linn et al[27] introduce the concept of desktop-activity mining. In this work, the authors combine monitoring techniques with process mining. However, it would not be feasible to apply this in complex settings, such as the BPO domain, since it requires access to actual UI elements. Similarly, Leno et al[9] suggest that using process mining for the discovery of local process models (ie, frequent patterns) from UI logs may be useful in training robots. Furthermore, Van der Aalst et al[28] describe how RPA may leverage techniques from the process-mining paradigm.

In the work presented by Kämäräinen,[14] the author explains the complicated structure that currently exists for the development of RPA. The author points out that, in the early stages of development, the robots work well on the development servers; however, the transition to the production phase becomes overly complicated. This is usually due to the fact that developers lack permission to access the test servers, which means they cannot test their implementations in the preproduction environment, and therefore, the clients are often requested to test the implemented robots. In this context, Jimenez-Ramirez et al[15] propose a modification of the process that has been being used in the analysis and design steps of RPA lifecyle at the Servinform company. This modification, and the resulting improvement, was based in use the back-office personal, who were training in the process that the company wants to automatize, to monitor, and build a log with all the key and mouse interaction in order to discover the process the robot must follow.

Considering the testing phase of RPA, which is the key point of this work, some related work have been found. Regarding to automatic testing, Morán et al[29] proposes new testing techniques with the aim of detecting faults by simulating different infrastructure configurations in MapReduce applications. Petković et al[30] present a methodology of automatic testing and its specific steps during testing of web applications based on the combination of Groovy programming language, Geb web driver, and Spock test framework. A new automatic GUI test method based on a three-layer script, showing in detail the steps necessary for its implementation, is presented by Huang and Zeng.[31] Yatskiv et al[32] present a method for software test automation that allows executing tests in a faster and more reliable way. As described by Moffitt et al[33] and Rozario and Vasarhelyi,[34] RPA can be used for different tasks of audit processes such as internal control or attribute matching. Audit procedures for the revenue account involve manual and repetitive tasks. In this sense, RPA can improve audit quality by testing the population of revenue transactions and allowing the auditor to more precisely evaluate and address the risk of revenue material misstatement.

Summarizing, RPA is being used in many different contexts, and it is increasing attention by the community. Although it has been used in testing domains, related work presented differ in some aspects in comparison with the subject matter dealt with in this paper, eg, the use of logs. Most of the works are focused on using RPA for software testing and not for testing the implemented robots. To the best of our knowledge, there are not proposals that cover this kind of testing being a key aspect to guarantee the reliability of the expected results of RPA projects.

7 | CONCLUSIONS AND FUTURE WORK

Throughout this paper, a proposal to automatically generate test suites for RPA projects has been presented. To this end, a test environment, which simulates the behaviour of the real system, is constructed based on UI log information, ie, UI information that has been monitored from human people working with the same system as the robots will have to work with. Robots are then deployed to work against this synthetic system instead of the real system. They are requested to perform some of the tasks that were monitored, and their behaviour is then checked against behaviour that was monitored. Testing robots in this environment is, on the one hand, more secure since it reduces the risk of damaging a real system and, on the other hand, more convenient since it is done in an automatic way.

This proposal is currently being validated in a real R&D project in the Servinform S.A. company. Thanks to this validation, a prototype has been developed and the preliminary results obtained seem promising.

Nonetheless, during the development of this proposal, certain limitations have been detected that could open new research lines. First, this paper presents a *straight* testing process that checks if every step has been reproduced in exactly the same way. For example, if the human people who are monitored has performed a task less efficiently than the robot, the test will not pass. Therefore, there is only one way to execute the operation although, in fact, it can be done in many ways. For instance, moving down on a component can be done by scrolling over that component or clicking on the "down arrow" till finding the desired element, or filling input fields in a form can be carried out in different orders without affecting the result. Moreover, it is important to note that in case that additional environments (ie, device configuration nd specific software versions) or test cases want to be generated, such environments and behaviours must be generated due to this limitation.

Second, this proposal is based on the fact that a UI log exists. What is more, this log is considered to contain information about the execution of a set of processes that are performed by a human. Although this is a strong assumption and some companies have acknowledged that recording their employees' behaviour might entail some privacy issues, this kind of logs are not so unusual in the BPO context.

Third, the current proposal relies on process mining applied over the UI logs, which are generated through the monitoring of human computers. Therefore, the results are highly sensitive to changes in user interfaces. When such monitoring requires long time periods—to gather a log with an appropriate size—there exists an increased risk related to potential changes due to continuous software evolution that user interfaces use to suffer.

Finally, the generated test environment comprises several variants; the running example process presented in this paper (cf Figure 1) has two alternatives: The student already has a grade, or the student has no grade. Our proposal only tests one case for each alternative to minimize the number of tests. However, the testing policy would depend on the kind of test that is performed. For example, developers might have developed a new version of a robot that affects only some part of the process, thus, testing all the possible alternatives could mean a waste of time and resources.

As a consequence of the limitations identified, certain interesting research lines are opening up.

1. One of the most significant lines is the inference of the behaviour of the UI components to make the test more flexible. This can be carried out, for example, using machine learning techniques. This arrangement will allow the components of the windows to be treated as black boxes, by focusing on the result that is obtained after its execution and not on the behaviour at a low-level view, ie, mouse clicks and keystrokes.
2. To improve the efficiency of the test execution, different testing policies can be deployed according to the different development state, eg, regression tests and deep tests on a variant.
3. It is necessary to perform an empirical validation with a variety of contexts. Although, as mentioned above, this proposal is currently being validated in a real project performing a case study that will provide more rigour to the evaluation.
4. This new evaluation will include wrong test cases too, eg, generating mutants of test cases. Therefore, these test cases can be used to check the potential ability to detect errors. On the one hand, its results must be reported, and, on the other hand, other case studies would be necessary to further corroborate the results in a more reliable way.

ACKNOWLEDGEMENTS

This research has been supported by the Pololas project (TIN2016-76956-C3-2-R) of the Spanish Ministry of Economy and Competitiveness and the RAIL project (P114-16/E09) of the Centre for Industrial Technological Development (Centro para el Desarrollo Tecnológico Industrial, CDTI) of Spain.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

ORCID

Andres Jiménez-Ramírez https://orcid.org/0000-0001-8657-992X
José González Enríquez https://orcid.org/0000-0002-2631-5890

REFERENCES

1. Fung HP. Criteria, use cases and effects of information technology process automation (ITPA). *Advances in Robotic and Automation*. 2014;3:1-11. https://10.4172/2168-9695.1000124
2. Willcocks L, Lacity M, Craig A. Robotic process automation: strategic transformation lever for global business services? *J Inform Technolo Teach Cases*. 2017;7(1):17-28.
3. Slaby JR. Robotic Automation emerges as a threat to traditional low-cost outsourcing. https://www.blueprism.com/wpapers/robotic-automation-emerges-threat-traditional-low-cost-outsourcing[Online; accessed September 2019]; 2018.
4. Geyer-Klingeberg J, Nakladal J, Baldauf F, Veit F. Process mining and robotic process atomation: a perfect match. In 1–8; 2018.
5. Penttinen E, Kasslin H, Asatiani A. How to choose between robotic process automation and back-end system automation?In; 2018.
6. Blue Prism. https://www.blueprism.com, [Online; accessed September 2019].
7. UiPath. https://www.uipath.com, [Online; accessed September 2019].
8. WorkFusion RPA Express. https://www.workfusion.com/rpaexpress[Online; accessed September 2019].
9. Leno V, Dumas M, Maggi FM, La Rosa M. Multi-perspective process model discovery for robotic process automation. In: CEUR Workshop Proceedings, 2018;2114:37-45.
10. Chacón-Montero J, Jiménez-Ramírez A, Enríquez JG. Towards a method for automated testing in robotic process automation projects. In: AST '19: Proceedings of the 14th International Workshop on Automation of Software Test IEEE Press; 2019:42-47.
11. Anagnoste S. Robotic automation process—the next major revolution in terms of back office operations improvement. In: Proceedings of the International Conference on Business Excellence, Bucharest, Romania. 2017;11:676-686. https://10.1515/picbe-2017-0072
12. Asatiani A, Penttinen E. Turning robotic process automation into commercial success—case OpusCapita. *J Info Technol Teach Cases*. 2016;6(2):67-74.
13. Madakam S, Holmukhe RM, Jaiswal DK. The future digital work force: robotic process automation (RPA). *JISTEM - J Inf Syst Technol Manag*. 2019;16(1):1–16.
14. Kämäräinen T, et al. Managing robotic process automation: opportunities and challenges associated with a federated governance model. *Master's Thesis*: School of Business; 2018.
15. Jimenez-Ramirez A, Reijers HA, Barba I, Del Valle C. A method to improve the early stages of the robotic process automation lifecycle. In: P. Giorgini, B. Weber, eds. *Advanced Information Systems Engineering*, CAiSE 2019. Lecture Notes in Computer Science, vol. 11483. Cham: Springer; 2019:446-461.
16. Spillner A, Linz T, Schaefer H. *Software testing foundations*. Germany: dpunkt.verlang; 2006.

17. Hultin J, Trudell C, Vashistha A, Glover T. Implications of technology on the future workforce. tech. rep., Defense Business Board Washington United States; 2017.
18. Naik VK, Garbacki P, Mohindra A. Architecture for service request driven solution delivery using grid systems. In: SCC '06: Proceedings of the IEEE International Conference on Services Computing IEEE; 2006:414-422.
19. Lacity M, Willcocks L. Robotic process automation: the next transformation lever for shared services. London School of Economics Outsourcing Unit Working Papers 7; 2015.
20. Aguirre S, Rodriguez A. Automation of a business process using robotic process automation(RPA): a case study. In: J. Figueroa-Garcá, E. López-Santana, J. Villa-Ramírez, R. Ferro-Escobar, eds. *Applied Computer Sciences in Engineering. WEA 2017. Communications in Computer and Information Science*, Vol. 742. Cham: Springer; 2017:65-71.
21. Barnett G. Robotic process automation: adding to the process transformation toolkit. White paper IT0022-0005, Ovum Consulting; 2015.
22. Mijović P, Giagloglou E, Todorović P, Mačužić I, Jeremić B, Gligorijević I. A tool for neuroergonomic study of repetitive operational tasks. In:ACM; 2014.
23. Lu H, Li Y, Chen M, Kim H, Serikawa S. Brain intelligence: go beyond artificial intelligence. *Mob Netw Appl*. 2018;23(2):368-375.
24. Lamberton C, Brigo D, Hoy D. Impact of robotics, RPA and AI on the insurance industry: challenges and opportunities. *J Financ Perspect: Insurance ed*. 2017;4(1):8–20.
25. Le Clair C, Cullen A, King M. The forrester wave^{TM}: robotic process automation. q1 2017; 2017.
26. Leopold H, Aa v dH, Reijers HA. Identifying candidate tasks for robotic process automation in textual process descriptions. In: J. Gulden, I. Reinhartz-Berger, R. Schmidt, S. Guerreiro, W. Guédria, P. Bera, eds. *Enterprise, Business-Process and Information Systems Modeling. BPMDS 2018, EMMSAD 2018. Lecture Notes in Business Information Processing*, Vol. 318. Cham: Springer; 2018:67-81.
27. Linn C, Zimmermann P, Werth D. Desktop activity mining—a new level of detail in mining business processes. Köllen Druck+ Verlag GmbH; 2018.
28. van der Aalst WMP, Bichler M, Heinzl A. Robotic process automation. *Bus Inf Syst Eng*. 2018;60:269-27.
29. Morán J., Bertolino A, de la Riva C, Tuya J. Automatic testing of design faults in mapreduce applications. *IEEE Trans Reliab*. 2018;67(3):717-732.
30. Petković M., Čandrlić Sanja, Ašenbrener Katić M. Automatic testing of web applications with the support of Geb web driver. *Zbornik Veleučilišta u Rijeci*. 2019;7(1):185-207.
31. Huang M, Zeng L. An automatic testing method for GUI using the framework of three-layer test script. In:Atlantis Press; 2019.
32. Yatskiv S, Voytyuk I, Yatskiv N, Kushnir O, Trufanova Y, Panasyuk V. Improved method of software automation testing based on the robotic process automation technology. In: 2019 9th International Conference on Advanced Computer Information Technologies (ACIT). IEEE; 2019:293-296.
33. Moffitt KC, Rozario AM, Vasarhelyi MA. Robotic process automation for auditing. *J Emerging Technol Account*. 2018;15(1):1-10.
34. Rozario AM, Vasarhelyi MA. How robotic process automation is transforming accounting and auditing. *The CPA J*. 2018;88(6):46-49.