Shadow Testing for Business Process Improvement

Suhrid Satyal[1,2], Ingo Weber[1,2], Hye-young Paik[1,2],
Claudio Di Ciccio[3], and Jan Mendling[3]

[1] Data61, CSIRO, Sydney, Australia
[2] University of New South Wales, Sydney, Australia
{suhrid.satyal,ingo.weber}@data61.csiro.au , hpaik@cse.unsw.edu.au
[3] Vienna University of Economics and Business, Vienna, Austria
{claudio.di.ciccio,jan.mendling}@wu.ac.at

Abstract. A fundamental assumption of improvement in Business Process Management (BPM) is that redesigns deliver refined and improved versions of business processes. These improvements can be validated online through sequential experiment techniques like AB Testing, as we have shown in earlier work. Such approaches have the inherent risk of exposing customers to an inferior process version during the early stages of the test. This risk can be managed by offline techniques like simulation. However, offline techniques do not validate the improvements because there is no user interaction with the new versions. In this paper, we propose a middle ground through *shadow testing*, which avoids the downsides of simulation and direct execution. In this approach, a new version is deployed and executed alongside the current version, but in such a way that the new version is hidden from the customers and process workers. Copies of user requests are partially simulated and partially executed by the new version as if it were running in the production. We present an architecture, algorithm, and implementation of the approach, which isolates new versions from production, facilitates fair comparison, and manages the overhead of running shadow tests. We demonstrate the efficacy of our technique by evaluating the executions of synthetic and realistic process redesigns.

Keywords: Shadow Testing, Business Process Management, DevOps, Live Testing

1 Introduction

Business process improvement ideas often do *not* lead to actual improvements. Works on business improvement ideas found that only a third of the ideas observed had a positive impact [11,14,13]. If improvements can only be achieved in a fraction of the cases, there is a need to rapidly validate the assumed benefits.

Simulation and AB testing techniques for business processes provide incremental validation support [18,20]. A new process version can be simulated using historical data from the old version. If the performance projections from the

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

simulation are satisfactory, the two versions can be deployed simultaneously on their production system such that each version receives a portion of customer requests. This method of simultaneous live-testing in production, called AB testing, is a method from DevOps [2], and compares two versions (A and B) in a fair manner. The speculative projections from the simulation are validated through performance data from the production system. This approach treats off-line simulation as a sanity-check before deployment of the new version, which reduces the risk of deploying versions with problems that can be anticipated beforehand. Nevertheless, the risk of exposing a significant number of customers to a bad version during the early stages of AB testing still remains. In addition, with these techniques, the performance of the two versions can only be compared on a collective level. There is no one-to-one mapping between process instances of two versions: each instance is executed on either version A or B.

In this paper, we propose a middle ground between simulation and AB testing with the idea of shadow testing. A new version of a process is deployed alongside the current version in the production system, but it is hidden from the customers and the process workers. A copy of user requests on the current version is forwarded to this hidden *shadow* version. When a process instance of the current version runs to completion, a corresponding shadow version in instantiated in *shadow mode*. This shadow process instance is partially executed as if it were the current version, and partially simulated with the execution information obtained from the completed process instance. This approach allows us to take a particular customer request as a reference and observe the performance and behaviour differences between the corresponding instances of the current version and the new version. We implemented and evaluated the approach.

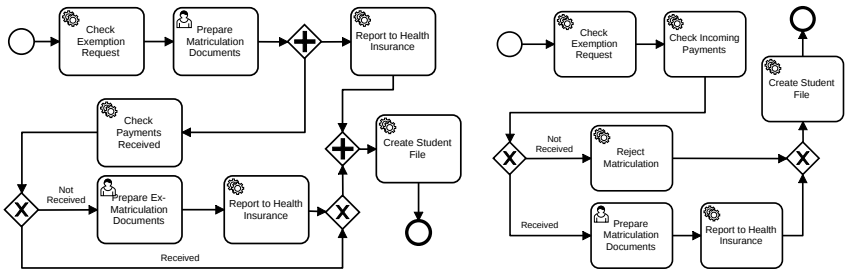
The remainder of this paper starts with a discussion of the requirements and related work in Section 2. Section 3 describes our approach to the requirements and the design trade-offs. In Section 4, we discuss the implementation architecture and the details of shadow test execution. We evaluate our approach in Section 5, discuss the strengths and weaknesses in Section 6, and draw conclusions in Section 7.

2 Background

There are essentially two broad approaches to process improvement in business process management. First, *business process re-engineering* (BPR) offers a methodology for redesigning processes from a clean slate [10,5]. Second, approaches to *business process improvement* take a more cautious and incremental method [12]. The BPM lifecycle integrates process improvement into a continuous management approach [7]. This lifecycle puts a strong emphasis on modelling and analysis before engaging with redesign, and also assumes that processes models are incrementally improved. Using these approaches, the old version of a process is replaced by a new version in the production system.

The AB-BPM [20] methodology provides an extended BPM lifecycle that facilitates rapid validation of improvement assumptions without needing to replace

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9



(a) P_1 (b) P_2

Fig. 1: Two versions of student matriculation process model: P_1 (as-is) and P_2 (to-be). Adapted from [8].

the old version. To date, this is the only approach that supports the idea of using the principles of DevOps [2] to address this need for business processes. Rapid validation builds on the existence of a newly redesigned process versions, which are typically modelled using the BPMN notation. Figure 1 shows as example of an *as-is* version of process and a redesigned *to-be* version of a student matriculation process originating from a case study in the literature [8]. In AB-BPM, such versions are progressively validated by simulation and AB testing.

In the following, we describe the problems with the AB-BPM methodology, derive key requirements for an improved solution, and describe the related work.

2.1 Problem Description and Requirements

Customer preferences are difficult to anticipate before deployment and there is a need to carefully test improvement hypotheses in practice [14]. If an improvement hypothesis has to be validated through user interaction, the new versions have to be exposed to the users. However, exposing inferior versions to the user can have an adverse effect on the business. We call this problem *risk of exposure*.

In AB testing, the risk of exposure can be reduced by gradually allocating more user requests to a better version during the tests [18,20,19]. However, this risk is not completely eliminated.

A better solution should eliminate this risk completely while still providing support for validating process improvements. Based on the above analysis, such a system should satisfy the following requirements:

R1 Fair Comparison: The system should compare execution of process versions under similar circumstances and reduce bias that may result from the execution environment and the behaviour of process workers.

R2 Minimal Impact from Overhead: The process version under test should have minimal impact on the performance of the production version.

R3 Isolation: The process version under test should not be visible to users, and should not change the process in the production system.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

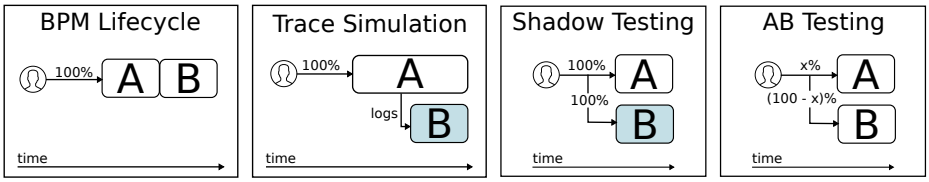


Fig. 2: Comparison of process improvement approaches. Process versions shown in white are executed in production, and those shown in colour are executed in test environments.

One way to eliminate this risk of exposure is by a form of live testing known as *shadow testing*. This approach is commonly used to observe functional issues that were undetected in earlier testing phases, and non-functional properties that can be known only after deployment. A copy of a request from the production system is forwarded to a *shadow version*, which runs in the background.

Our research adopts this idea of shadow testing for validating the improvement assumption inherent in new versions of a business process. Following the discussion of related work below, we devise dedicated architecture and execution techniques that addresses requirements R1-R3.

2.2 Related Work

We start this section with discussion on existing works on business process simulation, AB testing, and shadow testing in general. Then, we compare these techniques from a lifecyle perspective. Finally, we describe the gaps in shadow testing literature and how we address that gap.

Process simulation techniques can be useful for analyzing process redesigns. Advanced simulation techniques can extract knowledge from the historical logs of a process and predict performance of new versions [1,17,16,20]. These off-line techniques make many assumptions about process execution, and do not execute the code associated with activities. Therefore, the results of such techniques are not always reliable.

AB testing is a live testing approach that overcomes this limitation of simulation. Typically, a pool of user are selected for test, and their requests are evenly allocated to the two deployed versions (A and B) [2,4,14]. The *risk of exposure* in AB testing can be reduced by gradually allocating more users to the version that is estimated to best serve them as the test progresses [19].

Another form of live testing is *shadow* (or *dark*) *testing*. A copy of a request from the production system is forwarded to a hidden *shadow version*. Such tests are typically executed for a shorter duration than AB tests because these tests are not dependent on getting statistically significant user behaviour data [21]. Shadow testing can also be found in the area of aircraft operation control where a shadow version collects real time data and performs simulations [6].

Referring to the as-is version of a process as A and the to-be version as B, Fig. 2 illustrates how user requests for process instantiation are allocated to the

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

two versions. In the traditional approach, all requests are allocated to version A and then to version B when the version A is phased out. In simulation, all requests are allocated to version A. After the execution of some process instances, the logs of version A are used offline to estimate the performance of version B. In shadow testing, a single user request instantiates both the version A, which serves the user, and version B, which runs in the shadow. Finally, in AB testing, requests are dynamically routed to the two versions which run in production.

The practice and effects of shadow testing for non-functional testing of web applications are documented in surveys and industry reports [9,21]. However, there is a lack of scientific literature on dedicated architectures and techniques. Furthermore, shadow testing has not been adopted in business process management. This paper adds to this body of knowledge by providing an architecture design and execution mechanism for shadow tests for business processes.

3 Solution Approach

In this section, we analyse how two process versions can differ and classify these differences. We then discuss how to address the requirements, and the encountered challenges and trade-offs.

3.1 Classification of processes, changes, and activities

We need two process versions to execute shadow tests: the customer facing *production version* and a new *shadow version*. In order to understand the scope of shadow tests, we classify the types of changes between the versions as follows: *(i)* process model, *(ii)* back-end implementation, and *(iii)* user interface. Validation of user interface changes requires customer exposure. So, shadow tests are targeted towards the other two types of changes.

Processes models are composed of elements such as activities, gateways, and sequence flows. We classify these activities in three categories: *(i)* automatic tasks, *(ii)* user tasks, and *(iii)* manual tasks. They respectively represent fully automated, semi-automated, and unautomated activities. Service, script, send, receive, and business-rule tasks from the BPMN standard [7] belong to the first category. The difference between user and manual task is that manual tasks are performed without the aid of the process engine, according to definitions in the BPMN standard [4] . Therefore, we limit our scope to user tasks. Tasks from the shadow version can be further classified into two types: tasks that are shared between the two process models, henceforth *common* tasks, and *new* tasks introduced in the shadow version.

3.2 Conceptual Design

To facilitate fair comparison (R1), it is desirable to gather data from the full execution of process instances of the new version. However, this is at odds with our

[4] BPMN 2.0 Specification, http://www.omg.org/spec/BPMN/2.0/PDF/, Retrieved 25-07-2018

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

requirement for minimizing overhead (R2). If we instantiate the shadow version for each instance of the production version using the concept of shadow testing, we double the total work because the activities are enacted twice. Scheduled tasks, especially user tasks, require the involvement of valuable resources (machines and workers). This inevitably affects the performance of the production version. Furthermore, executing the same task twice for the same inputs can confuse the process workers.

As a compromise, we can instantiate the shadow version for every instance of the production version, but only partially execute the shadow instances. The executable parts of shadow version should run in isolation (R3), separate from the production version. We can execute *new* tasks in the shadow version, and estimate *common* tasks by copying information from the corresponding instance from production. This eliminates the problem of double execution but introduces a degree of speculation.

Partial execution can be ineffective for reducing overhead in scenarios where shadow versions have many new tasks. Execution of these new tasks inevitably affects the performance of the the production version. In such cases, we can reduce the amount of work related to the shadow version. One way to do this is to instantiate the shadow version when the production workload is low. However, this approach inhibits fair comparison because the production and the shadow instances may not execute under similar circumstances. Another approach is to instantiate the shadow version only when the load is expected to be low. Creating less instances of the shadow version in this way reduces the need to execute new tasks. However, this deviates from the concept of shadow testing where one shadow instance is created for every production instance thus establishing a one-to-one mapping between them.

To find a balance between fair comparison through one-to-one mapping and overhead reduction, we create a coupling between the rate of shadow version instantiation with the performance degradation. When the observed performance degradation is below a user defined threshold, we instantiate the shadow version for every instance of the production version. We reduce the instantiation rate when the degradation is above the given threshold, and increase the instantiation rate when the degradation is below the threshold.

One can consider the shadow process instance executions as experiments. Knowing that a certain task is being executed solely for the experiment may prompt the process workers to perform their work differently to influence the outcome. Therefore, another way in which we strive for fairness (R1) is by hiding the experiment. We propose a unified user interface that aggregates scheduled tasks from both versions and hides the internals of the experimentation platform. The limitation of this design choice is that user interface changes cannot be evaluated in the shadow mode. Such changes could be evaluated through separate AB tests for the user interface changes.

Operations on the test environment should be safe, i.e., tasks in shadow process instances should not override data in the production system and external services. The design choice of avoiding double scheduling puts isolation (R3) at

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

odds with fairness (R1) and overhead management (R2). Consider the example of using a global budget stored in the application database, where a new budget drawing task in the shadow version is followed by a common task. In this example, the common task and every other task that follows afterwards have to be executed so that the effect of this new budget drawing task is reflected on the execution of the instance of the shadow version. This is only possible if we execute the common tasks, which introduces the problem of double execution.

We devise a solution that lets the shadow version independently interact with a separate test database. We can manage test database in three ways: by synchronizing with production database, by keeping it un-synchronized, or by replacing it with snapshots of production during the tests. If we synchronize or overwrite with snapshots, we cannot see the impact of shadow version at the data level. If we don't synchronize, we cannot see the true impact because common tasks are not executed. Therefore, we chose the option of not synchronizing.

4 Conceptual Solution and Architecture

In this section, we present the conceptual solution by first describing the modular architecture we adopted. Then we discuss how the modules operate internally and how they interact.

4.1 Architecture

We propose a layered architecture which uses a single process engine to execute the production and shadow versions. The architecture is divided into four layers: the User Interface, the BPMS Layer, the Service Layer, and the Persistence Layer. Figure 3 shows the modular architecture diagram. The components shown in colour facilitate shadow testing.

The BPMS Layer is responsible for instantiating, executing, and estimating process instances. The shadow version and the production version are deployed alongside each other. The Process Engine enacts process instances and manages their state. The Execution Controller determines when the shadow version can be instantiated, and how tasks can be executed. User tasks are allocated to process workers. Automatic tasks are delegated to the Service Layer. If a task in the shadow version can be estimated, the Execution Controller delegates the task to the Estimator instead. The Estimator mimics the execution of the task and estimates its performance.

The Service Layer is responsible for executing automatic tasks. These tasks are implemented as services. Common Services and New Services are implementations of common and new tasks respectively. External Services represent third party services. Test Doubles are services that replace other services for testing purposes. These services are exposed to the BPMS Layer through a Service API.

The Persistence Layer handles database operations. The Application Database hosts the application data. The BPMS Database hosts the configuration, the process instance mapping, and the execution data for both versions. The Test

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

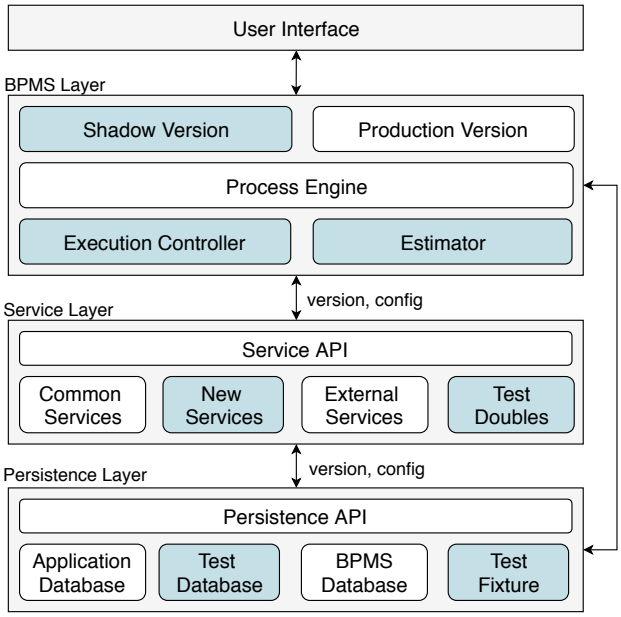


Fig. 3: Modular architecture for shadow testing. Components shown in colour facilitate shadow testing.

Database is a copy of the Application Database. The Test Fixture is a database loaded with a known set of data used specifically to make tests repeatable. These databases are exposed through a Persistence API.

A unified user interface hides the presence of two process versions. Process version and configuration information are passed down to the Service Layer and the Persistence Layer during the execution of process instances. Based on this information, these layers decide which services and data operations should be invoked. Test Doubles, Test Database, and Test Fixture are optional. However, either a Test Database or a Test Fixture is required for the execution of shadow versions that contain new automatic tasks.

4.2 Process Instance Execution

The shadow version is instantiated after the completion of an instance of the production version. A one-to-one mapping is created between these two instances so that they can be executed similarly and then compared after the experiment.

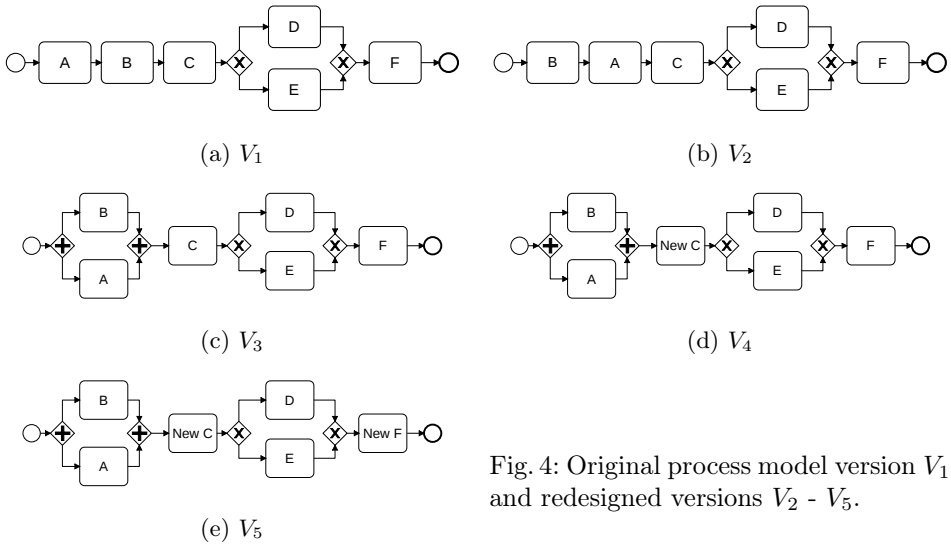
We can execute an instance of shadow version by progressively copying over the execution data of its corresponding instance of production version. We capture this data in the form of *snapshots*. A snapshot is composed of the task name, the start and end timestamps of the task, other metrics such as cost, and the values of all process variables at the time of completion of the task. A snapshot is created when a task in a process instance of the production version runs to

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

completion. These snapshots are stored in the BPMS database, and retrieved during execution of corresponding shadow instances for estimation.

Consider the case of tasks reordering shown in Fig. 4 for versions V_1 to V_2. Execution sequences for V_1 begin with $Old = \langle A, B, C \rangle$. Snapshots are created at the completion of tasks *A, B,* and *C*. The shadow version begins with $New = \langle B, A, C \rangle$. We can estimate B by copying over snapshot data from *Old* even though the trace does not conform with V_2. Since the same tasks are in different order, we cannot copy over raw timestamps from the snapshot. Instead, we aggregate the timestamps into duration and waiting time metric and make an estimate of execution times of these tasks.

Since we avoid executing common tasks and copy over aggregated information from snapshots, shadow process instances do not need to be executed concurrently with their corresponding process instances.



(a) V_1 (b) V_2

(c) V_3 (d) V_4

(e) V_5

Fig. 4: Original process model version V_1 and redesigned versions V_2 - V_5.

Adjusting task estimates using redesign information. When the shadow version is deployed in production, common tasks may run differently than those in the old production version. These differences cannot be captured by copying over snapshots. Therefore, we allow analysts to adjust estimates available in the snapshots through user defined functions. These user defined functions are executed by the Estimator.

Information on redesigns can be used in making better estimates of duration and waiting times of tasks. Consider some redesigns chosen from known best practices [15,7] illustrated in Fig. 4. In the example of parallelisation, from V_1 to V_4, the redesign may make a process instance faster, but it can consume more resources in the given time-frame. Analysts can make explicit assumptions about

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

how duration and waiting times of the parallelised tasks are affected and provide a function that adjusts the values retrieved from snapshots.

Managing performance overhead. In the presence of numerous new tasks, the execution of shadow process instances can slow down the production instances, because those new tasks have to be enacted. We approach this challenge by reducing the rate at which the shadow versions are instantiated. We instantiate shadow versions only when there is evidence that doing so does not degrade the quality of instances in production. Our method to adjust the rate of instantiation was inspired by the congestion control algorithm of TCP/IP [3]: its Additive Increase Multiplicative Decrease (AIMD) algorithm exponentially decreases the instantiation rate when performance degradation is observed, and linearly increases the instantiation rate when performance is acceptable.

Before we define performance degradation, we need to observe the baseline performance of the production version. Let $s \in \mathbb{Z}^+$ be the number of process instances that need to be observed for this purpose. The base measure is the average duration of these s instances, $\mu_{1,s} \in (0, +\infty) \subset \mathbb{R}$. Let $\theta \in (1, +\infty) \subset \mathbb{R}$ be the threshold for acceptable degradation, in form of a factor. The acceptable average duration of production version instances during the test is thus $\theta \times \mu_{1,s}$.

Let n be the count of current user requests. We observe the average duration of the production version instances in a window of $w \in \mathbb{Z}^+$ requests and calculate the mean $\mu_{n-w+1,n}$. Let $a \in \mathbb{Z}^+ \cup \{0\}$ and $b \in (0, 1) \subseteq \mathbb{R}$ be the user-defined additive increase and multiplicative factor respectively. The shadow process instantiation rate $i(n) \in (0, 1) \subseteq \mathbb{R}$ for user request number n is adjusted using the base measures and above parameters as shown in Eq. (1).

$$i(n) = \begin{cases} i(n) + a & \text{if } \mu_{n-w+1,n} < \theta \times \mu_{1,s} \\ i(n) \times b & \text{if } \mu_{n-w+1,n} >= \theta \times \mu_{1,s} \\ 1 & \text{if } i(n) + a > 1 \\ & \text{and } \mu_{n-w+1,n} < \theta \times \mu_{1,s} \end{cases} \quad \text{where} \quad \begin{array}{l} a >= 0, \\ 0 < b < 1, \text{ and} \\ \theta > 1 \end{array} \tag{1}$$

Task Execution. The layered architecture and configuration information dictate how and where a task is executed. Figure 5 summarizes how these layers decide the way of handling tasks.

Tasks created by a production process instance are executed as per the norm. Automatic tasks are invoked; user tasks are scheduled. Execution and application data are written to the corresponding databases. Tasks created by instances of a shadow version can only be executed if either the Test Fixtures or the Test Database is available. This is essential for providing isolation (R3) because test data can be read from but should not be written to the production database. Test Doubles can be executed in place of automatic tasks if they are available. New user tasks are always scheduled.

Common user tasks can be executed as if they were new tasks, if they do not have a *side-effect* to the business. The tasks have to be marked as such, and it is up-to the analyst to determine what constitutes a side-effect. For example, a task

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

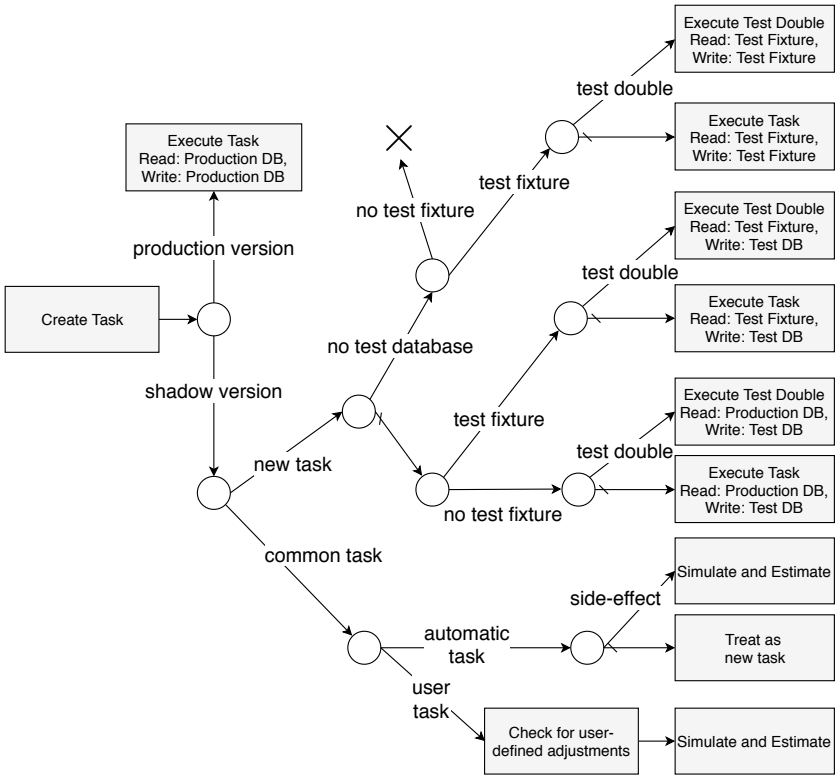


Fig. 5: Decision tree for task execution

that requires workers to make phone calls to a third party may be said to have a side-effect. Common automatic tasks are treated as new tasks and executed by default. However, these tasks can also be configured to be estimated.

Consider the example of student matriculation process versions shown in Fig. 1 where we execute P_1 in production and P_2 in shadow. Assume that we provide a Test Database and say that user tasks do not produce side-effects. Using the decision tree in Fig. 5, we see that all tasks in P_1 will be executed in the same way as if P_1 was the only version that was deployed. For P_2, only the tasks with new labels (e.g. "Reject Matriculation") will be executed. The implementation of these tasks will write data to the configured Test Database. Tasks that are available in P_1 but not in P_2 (e.g. "Prepare Ex-Matriculation Documents") do not need to be executed in the shadow version.

4.3 Non-compliant Shadow Instances

Some redesigns can impede process instances of the shadow version to complete. This is an effect of our design choice of executing only new tasks in the shadow

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

instance. In the redesign of the production process version V_3 into shadow version V_4 (see Fig. 4), task C is substituted with *New* C before an exclusive-choice gateway. The outcome of these tasks dictates whether we perform task D or E next. Consider a situation where the outcome of C in an instance of the production version and *New* C in the corresponding instance of the shadow version differ, leading to different choice of the next activity. From this point onward, the shadow version V_4 cannot be executed because the snapshot for the next task is not available. Since the execution path is determined by the outcome of the previous task, such cases can only be resolved at runtime.

When activities in instances of the shadow version are executed in a different order than those in the production version, some process variables can be overwritten. Consider the example of redesign of the production version V_1 into the shadow version V_2 in Fig. 4, where the re-ordering of tasks A and B takes place. Process variables after the completion of tasks from V_1 would then be copied over in the reverse order using snapshots. Therefore, it is possible for task A in the shadow version to *undo* the changes to process variables made by the task B. If process variables are overwritten, then new tasks that are to be executed in the shadow instance may produce different results. This can also lead to the non-compliance. issue describe above.

If the proportion of such incomplete cases exceeds a pre-defined threshold, we stop instantiating the shadow version. This ensures that we do not waste resource on fruitless executions in the future.

5 Evaluation

In this section, we evaluate our approach using two sets of redesigned process models: the five versions created by using redesign strategies, as shown in Fig. 4, and the two versions of the realistic process models in Fig. 1. Using synthetic data, we first investigate how the instantiation rate of a shadow version changes in response to varying production load. Then we test if shadow tests can accurately estimate the performance of new versions in terms of execution time, by comparing the estimated execution time from shadow testing to the execution time of the same version when it is deployed in production. For this purpose, we use both the synthetic and realistic models (Sec. 5.2 and Sec. 5.3 respectively). Finally, we compare simulation and shadow tests qualitatively.

Experiment setup. We have designed our shadow testing experiments in such a way that there is one (emulated) process worker who can only work on one task at a given time. Upon creation of a user task, the task is placed on a First-In-First-Out (FIFO) task queue. The process worker completes items in the queue. We implemented the shadow testing architecture and emulated the process worker on the Java Virtual Machine with Activiti BPMS and PostgreSQL database. We executed the experiments on a Ubuntu 16.04 machine with 8GB RAM and Intel Core i7-6700 CPU at 3.4GHz.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

For the synthetic process models, we execute the new version in shadow mode with V_1 in production and estimate the execution time of the instances of shadow version. We assume that $V_2, \ldots, V_5$ are the new versions designed to replace V_1. Table 1 shows the execution and waiting times for each task in the five versions. For the realistic process models, we execute the as-is version P_1 in production, estimate the execution time of instances of the to-be version P_2, and then compare the results with the execution time of P_2 as deployed as production. The shadow tests are performed without resorting on user defined functions for waiting-time and execution-time adjustment.

Table 1: Waiting and execution times of activities in V_1 - V_5

Activity	Min. Wait Time	Execution Time
A	1 sec	1 sec
B	1 sec	2 sec
C	3 sec	2 sec
D	5 sec	1 sec
E	1 sec	5 sec
F	2 sec	1 sec
New C	1 sec	3 sec
New F	1 sec	2 sec

5.1 Adaptation to Production Load

In this section, we describe how we observed the impact of the shadow process instances on the performance of the execution environment running process instances in production, thus assessing the effectiveness of the overhead management algorithm. In particular we execute V_1 in production and V_5 in shadow mode with one process worker operating with both versions. Since V_5 requires the process worker to complete new tasks, executing a shadow instance slows down the other concurrently active process instances.

We denote with l the number of process instances that are serving a request. We start with the assumption that the workload on production is $l = 1$. We set 5 requests as the bootstrap period during which only the production version is instantiated. The average duration of these 5 instances is taken as the baseline. We set up the overhead management strategy shown in Eq. (1) with arbitrarily chosen parameters $a = 0.1, b = 0.5, w = 5$, and $\theta = 1.1$. If the most recent 5 production instances slow down to 1.1 times the baseline, the shadow version instantiation rate is halved. Otherwise, the rate is increased by 0.1. With this setup, we execute the shadow tests, and intermittently increase the load to test how an increase in the load affects the shadow version instantiation rate $i(n)$.

Figure 6 shows the performance of the production instances under various workloads. It also shows the number of shadow processes instantiated at diverse workloads, and how instantiation rate is adjusted in response to performance degradation. We observe that the shadow version is instantiated less often when the load is high. This is reflected in how the cumulative number of shadow instances change over time. For instance, when the workload increases to $l = 2$ after 10 requests, the execution time of instances surpasses the threshold. As a response to this degradation, the instantiation rate drops by half to 0.5. As a result, the shadow version is instantiated probabilistically and the cumulative shadow instances does not increase linearly as before. We choose the average ex-

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

ecution time in window $w = 5$ for detecting overhead. Therefore, the adjustment in instantiation rate lags the first observation of overhead.

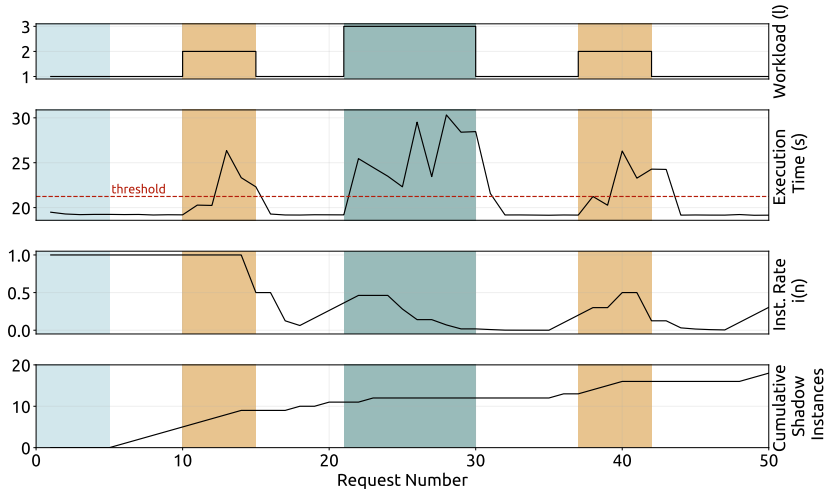


Fig. 6: Comparison of performance of production version V_1 and instantiation rate of shadow version V_5.

5.2 Accuracy of Estimated Execution Time – Synthetic Processes

In this section, we describe our observation on the accuracy of estimates of execution time from shadow tests. Figure 7 shows the differences between the performance of V_1 in production and the respective shadow version. We observe the average performance of for 50 instances of each version. In the case of V_1 vs V_2, the result of the shadow test is accurate because the waiting time and the execution time of the re-ordered tasks are the same. In the case of V_1 vs. V_3, V_3 is faster than estimated because the waiting time between the parallelised tasks is reduced. Since all tasks can be estimated, the waiting time and the execution time are copied over using snapshots. We do not use user defined functions for adjustment of waiting times. This effect of reduction of waiting time can also be seen in V_4 and V_5.

V_4 and V_5 replace task C with *New* C. We have implemented these tasks in such a way that task C determines that the branches through D and E are traversed equally at random, whereas *New* C determines D to be enacted in 60% of the cases and E in 40%. Because of these implementation differences, the corresponding instances between production and shadow versions do not always take the same path. Figure 7 shows the estimates of the shadow instances that *matched* the same path as their corresponding production instances. It also shows

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

Table 2: Waiting time execution time of activities in P_1 and P_2

Activity	Min. Wait Time	Execution Time
All automated tasks	0	1 min
Check Payments Received	3 days	2 min
Check Incoming Payments	3 days	2 min
Prepare Matriculation Documents	$\mathcal{N}(\mu = 1\ day, \sigma^2 = 1^2)$	$\mathcal{N}(\mu = 60\ min, \sigma^2 = 5^2)$
Prepare Ex-Matriculation Documents	$\mathcal{N}(\mu = 1\ day, \sigma^2 = 1^2)$	$\mathcal{N}(\mu = 30\ min, \sigma^2 = 2^2)$

how much of the estimate was comprised of the actual execution and how much was based on the estimates from snapshots.

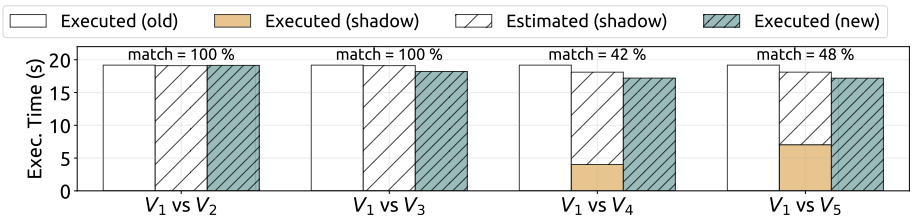


Fig. 7: Performance of old version V_1 in production, and new versions V_2 - V_5 in production and shadow mode

5.3 Accuracy of Estimated Execution Time – Realistic Processes

We perform shadow tests on a student matriculation process from a German university. This process requires eligible students to confirm their enrolment by paying tuition fees. In this experiment, we aim to mitigate unintended bias from using simpler synthetic versions, and to mimic realistic scenario by adding more complexity. We add more complexity by treating processing times as distributions and introducing request context.

We have adopted the *as-is* and the *to-be* versions of these processes, depicted in Fig. 1, based on a case study reported in the business process improvement literature [8]. The goal of this redesign was to reduce the cycle time. In this experiment, we use shadow testing to check if this goal can be achieved.

Since the execution details are not available, we implement the tasks such that they follow the waiting and execution times shown in Table 2. We assume that there are two types of requests: those from graduating students and those from students who wish to continue their studies. For the purpose of illustration, we arbitrarily set the success rate of receiving payments from graduating students to 75%, and from continuing students to 80%. We assume that each request includes a student id. Requests raised during the process execution also include the process generated id sent to the health insurance for report, the

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

matriculation document, the ex-matriculation document, and the resulting matriculation status. In our implementation, we generate unique identifiers and random text to represent this information. We also use a process variable to test whether payment was received. With this implementation, we can compare the performance and resulting data from the production environment with those in the shadow mode.

Figure 8 shows the results upon the execution of 250 process instances of P_1 in production (old), P_2 in shadow mode during the shadow test (shadow), and P_2 in production as a stand-alone version (new). It can be observed that P_2 is estimated to be better for both types of students, and that this estimation reflects real performance. We remark that in the Application Database, we could not see any data related to P_2 during shadow tests as per the design. In the Test Database, the data about the ex-matriculation is not created because this activity is not available in P_2.

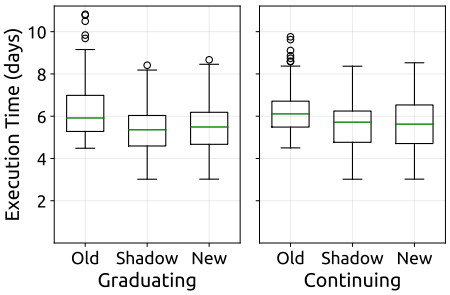


Fig. 8: Comparison of performance of P_1 and P_2 in two contexts

5.4 Qualitative Comparison with Simulation – Realistic Processes

In this scenario, the simulation approach from AB-BPM [20] would fail to produce accurate results. To simulate P_2, trace simulation techniques requires the following inputs from the analyst: *(i)* historical event logs of P_1, *(ii)* the default value for the process variable that indicates whether payment was received in P_2, and *(iii)* estimates for duration of new activities, e.g., "Reject Matriculation". In shadow testing we do not require these inputs because the implementation of $P2$ is available. In trace simulation, we cannot rely on the event log of P_1 after the completion of the "Check Incoming Payments" activity in P_2. Therefore, the simulation uses a predefined user-provided value dictating whether the payment was received or not, to choose the next activity after the exclusive-or gateway.

The trace simulation approach of AB-BPM is unaware of request contexts and implementation details. For every context (graduating or continuing students), a separate simulation using filtered logs should be run. During the simulation, potential implementation bugs regarding data storage cannot be discovered because the new version is not actually executed. For instance, if the implementation of the "Create Student File" task in P_2 depends on an ex-matriculation document created for failed payments, the shadow instances would raise an exception (notice that the dedicated activity is missing in P_2).

6 Discussion

We instantiate the shadow version only when the load is manageable. By design, this approach cannot be used to evaluate shadow versions under high load. If new

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

versions are designed to handle high load, AB testing can be used instead. Our approach instantiates a shadow version after the completion of an instance in the production. This simplifies the execution because we do not have to synchronize the two instances. For fairness, we have to ensure that the corresponding instances execute under similar external conditions.

In this paper, our focus was on architecture design and test execution. We assumed a simplistic approach of resolving data dependencies. Data consistency for general cases was out of scope for this work, and requires further research. In our evaluation, we adopted process models from the redesign literature and implemented them. We cannot use real world log data because of their complexity and lack of implemented code behind the activities. To mitigate the unintended bias from using synthetic redesigns, we also evaluated our approach using a redesign from a case study. Since we propose a new architecture and execution mechanism, we could not find industry implementations for evaluation.

7 Conclusion

Business process improvement ideas do not necessarily yield actual improvements. Simulation and AB testing techniques can be utilized for validating process improvements. However, the results of simulation are speculative and AB tests introduce risk of exposure. In this paper, we propose a shadow testing approach that provides a middle ground between these techniques.

A new version of a process is deployed alongside the current version in such a way that the new version is hidden from the customers and the process workers. User requests that instantiate the current version are also forwarded to the new version. If the performance overhead falls under a user defined threshold, the new version is instantiated. Process instances of the new version are partially executed and partially estimated by copying over *snapshots* from executing the current version. This way of shadow testing facilitates fair comparison between the two versions, isolates the new version, and limits the overhead of running tests. Using synthetic and realistic process redesigns, we demonstrate that shadow testing provides accurate performance estimates of new versions.

In future work, we plan to include shadow testing as a step in the AB-BPM methodology, and run case studies where we compare the costs and benefits of the methodology as a whole.

Acknowledgements. The work of Claudio Di Ciccio and Jan Mendling has received funding from the EU H2020 programme under MSCA-RISE agreement 645751 (RISE_BPM).

References

1. van der Aalst, W.M.P.: Business process simulation survival guide. In: Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed., pp. 337–370 (2015)

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

2. Bass, L., Weber, I., Zhu, L.: DevOps - A Software Architect's Perspective. SEI series in software engineering, Addison-Wesley (2015)
3. Chiu, D., Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Computer Networks 17, 1–14 (1989)
4. Crook, T., Frasca, B., Kohavi, R., Longbotham, R.: Seven pitfalls to avoid when running controlled experiments on the web. In: KDD. pp. 1105–1114 (2009)
5. Davenport, T.H.: Process innovation: reengineering work through information technology. Harvard Business Press (1993)
6. Denery, D.G., Erzberger, H.: The center-tracon automation system: Simulation and field testing (1995)
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
8. Falk, T., Griesberger, P., Leist, S.: Patterns as an artifact for business process improvement - insights from a case study. In: Design Science at the Intersection of Physical and Virtual Design. DESRIST. pp. 88–104 (2013)
9. Feitelson, D.G., Frachtenberg, E., Beck, K.L.: Development and deployment at facebook. IEEE Internet Computing 17(4), 8–17 (2013)
10. Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution. HarperCollins (1993)
11. Holland, C.W.: Breakthrough Business Results With MVT: A Fast, Cost-Free "Secret Weapon" for Boosting Sales, Cutting Expenses, and Improving Any Business Process. John Wiley & Sons (2005)
12. Kettinger, W.J., Teng, J.T.C., Guha, S.: Business process change: A study of methodologies, techniques, and tools. MIS Quarterly 21(1), 55–98 (1997)
13. Kevic, K., Murphy, B., Williams, L.A., Beckmann, J.: Characterizing experimentation in continuous deployment: A case study on Bing. In: ICSE-SEIP. pp. 123–132. IEEE Press (2017)
14. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. 18(1), 140–181 (2009)
15. Reijers, H.A., Mansar, S.L.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega 33(4), 283–306 (2005)
16. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Inf. Syst. 54, 1–14 (2015)
17. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.J.: Workflow simulation for operational decision support. Data Knowl. Eng. 68(9), 834–850 (2009)
18. Satyal, S., Weber, I., Paik, H., Di Ciccio, C., Mendling, J.: AB-BPM: performance-driven instance routing for business process improvement. In: BPM. pp. 113–129 (2017)
19. Satyal, S., Weber, I., Paik, H., Di Ciccio, C., Mendling, J.: AB Testing for process versions with contextual multi-armed bandit algorithms. In: CAiSE (2018)
20. Satyal, S., Weber, I., Paik, H., Di Ciccio, C., Mendling, J.: Business process improvement with the AB-BPM methodology. Information Systems (2018)
21. Schermann, G., Cito, J., Leitner, P., Zdun, U., Gall, H.C.: We're doing it live: A multi-method empirical study on continuous experimentation. Information and Software Technology (2018)

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-030-02610-3_9

References

Satyal, Suhrid, Ingo Weber, Hye-young Paik, Claudio Di Ciccio, and Jan Mendling (2018). "Shadow Testing for Business Process Improvement". In: *CoopIS*. Ed. by Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman. Vol. 11229. Lecture Notes in Computer Science. Springer, pp. 153–171. ISBN: 978-3-030-02609-7. DOI: 10.1007/978-3-030-02610-3_9.

BibTeX

```
@InProceedings{   Satyal.etal/CoopIS2018:ShadowTestingBusinessProcess,
  author        = {Satyal, Suhrid and Weber, Ingo and Paik, Hye{-}young and
                   Di Ciccio, Claudio and Mendling, Jan},
  title         = {Shadow Testing for Business Process Improvement},
  booktitle     = {CoopIS},
  year          = {2018},
  pages         = {153--171},
  crossref      = {OTM2018},
  doi           = {10.1007/978-3-030-02610-3_9},
  keywords      = {Shadow testing; Business process management; DevOps; Live
                   testing}
}
@Proceedings{     OTM2018,
  title         = {On the Move to Meaningful Internet Systems. {OTM} 2018
                   Conferences - Confederated International Conferences:
                   CoopIS, C{\&}TC, and {ODBASE} 2018, Valletta, Malta,
                   October 22-26, 2018, Proceedings, Part {I}},
  year          = {2018},
  editor        = {Herv{\'{e}} Panetto and Christophe Debruyne and Henderik
                   A. Proper and Claudio Agostino Ardagna and Dumitru Roman
                   and Robert Meersman},
  volume        = {11229},
  series        = {Lecture Notes in Computer Science},
  publisher     = {Springer},
  isbn          = {978-3-030-02609-7}
}
```