Business Process Improvement with the AB-BPM Methodology

Suhrid Satyal[a,b,*], Ingo Weber[a,b,*], Hye-young Paik[a,b], Claudio Di Ciccio[c], Jan Mendling[c]

[a] *Data61, CSIRO, Sydney, Australia*
[b] *University of New South Wales, Sydney, Australia*
[c] *Vienna University of Economics and Business, Vienna, Austria*

Abstract

A fundamental assumption of Business Process Management (BPM) is that redesign delivers refined and improved versions of business processes. This assumption, however, does not necessarily hold, and any required compensatory action may be delayed until a new round in the BPM life-cycle completes. Current approaches to process redesign face this problem in one way or another, which makes rapid process improvement a central research problem of BPM today. In this paper, we address this problem by integrating concepts from process execution with ideas from DevOps. More specifically, we develop a methodology called AB-BPM that offers process improvement validation in two phases: simulation and AB tests. Our simulation technique extracts decision probabilities and metrics from the event log of an existing process version and generates traces for the new process version based on this knowledge. The results of simulation guide us towards AB testing where two versions (A and B) are operational in parallel and any new process instance is routed to one of them. The routing decision is made at runtime on the basis of the achieved results for the registered performance metrics of each version. Our routing algorithm provides for ultimate convergence towards the best performing version, no matter if it is the old or the new version. We demonstrate the efficacy of our methodology and techniques by conducting an extensive evaluation based on both synthetic and real-life data.

Keywords: Business process management, DevOps, AB Testing, Trace simulation, Process performance indicators

*Corresponding authors
Email addresses: suhrid.satyal@data61.csiro.au (Suhrid Satyal), ingo.weber@data61.csiro.au (Ingo Weber), hpaik@cse.unsw.edu.au (Hye-young Paik), claudio.di.ciccio@wu.ac.at (Claudio Di Ciccio), jan.mendling@wu.ac.at (Jan Mendling)

1. Introduction

Various lifecycle approaches to Business Process Management (BPM) have a common assumption that a process is incrementally improved in the redesign phase [1, Ch.1]. While this assumption is hardly questioned in BPM research, there is evidence from the field of AB testing that improvement concepts often do *not* lead to actual improvements. For instance, work on business improvement ideas found that 75 percent did not lead to improvement: half of them had no impact while approximately a quarter turned out to be even harmful [2]. The results are comparable to that of a study of the Microsoft website, in which only one third of the ideas observed had a positive impact, while the remaining had no or negative impact [3]. The same study also observed that customer preferences were difficult to anticipate before deployment, and that customer research did not predict customer behaviour accurately.

If incremental process improvement can only be achieved in a fraction of the cases, there is a need to rapidly validate the assumed benefits. Unfortunately, there are currently two major challenges for such an immediate validation. The first one is *methodological*. Classical BPM lifecycle approaches build on a labour-intensive analysis of the current process, which leads to the deployment of a redesigned version. This new version is monitored in operation, and if it does not meet performance objectives, it is made subject to analysis again. All this takes time. The second challenge is *architectural*. Contemporary Business Process Management Systems (BPMSs) enable quick deployment of process improvements, but they do not offer support for validating improvement assumptions. A performance comparison between the old and the new version may be biased since contextual factors might have changed at the same time. How a rapid validation of improvement assumptions can be integrated in the BPM lifecycle and in BPMSs is an open research question.

We address this question by extending the business process lifecycle and providing techniques for these extensions. Our AB-BPM methodology integrates business process execution concepts with the idea of AB testing from DevOps, and supports the design of AB tests with simulation. The methodology and supporting techniques as a whole provide support for validating improvement assumptions inherent in new process versions.

AB testing compares two versions of a deployed product (e.g., a Web page) by observing users' responses to versions A and B, and determines which one performs better [4]. We implement this technique in such a way that two versions (A and B) of a process are operational in parallel and any new process instance is routed to one of them. Through a series of experiments and observations, we have developed an instance routing algorithm, $LTAvgR$, which is adapted to the context of executing business processes. The routing decision is guided by the observed performance metrics of each version at runtime.

To manage the risks of exposing even a few customers to clearly inferior versions during AB tests, we propose a technique to simulate new versions of business processes beforehand, using the execution logs and performance data of the old version. For the purpose of this simulation, we devise a data struc-

ture, the *Transition Simulation Tree (TST)*, which summarizes decisions and performance metrics available in the event log of a process. The TST allows the simulator to extrapolate historical observations from the existing version of a process to the new version, with minimal assumptions about how the process is implemented. The results of this simulation can be used for preliminary analysis of potential improvements, e.g., to rule out performing AB testing with a clearly inferior new version. They can also help in the designing rewards and configuring parameters of *LTAvgR*.

In an earlier version of this work [5], we proposed the AB-BPM approach. In this paper, we expand on this idea and show how it fits into a methodology that provides validation support for process improvement assumption. We also introduce a simulation technique that complements the AB testing approach.

The remainder of this paper starts with a discussion of the requirements and prior work in Section 2. Section 3 describes the lifecycle, techniques, and the framework that facilitate the AB-BPM methodology. In Section 4, we evaluate our AB testing and simulation approach. In Section 5, we discuss the strengths ans weaknesses of our approach, and finally draw conclusions in Section 6.

2. Background

This section discusses the background of our research. Section 2.1 identifies requirements for rapid validation of process improvements. Section 2.2 discusses in how far these requirements are addressed by prior research and outlines the general idea of AB-BPM.

2.1. Requirements for Rapid Improvement Validation

"Actions speak louder than words" is a proverb that emphasizes the need to take action once the right way to act is identified. Approaches to process redesign follow exactly this idea when they suggest that a specific weakness should be addressed by reusing an established heuristic [6]. The problem in this context is that such approaches deny the uncertainty about the understanding of the factors that influence process performance, and potentially ignore the non-deterministic behaviour of the persons that engage in the process [7].

An improvement hypothesis is neither self-evident nor fully understood in many cases, as highlighted by an anecdote of a leading European bank (EB). The EB improved their loan approval process by cutting its turnaround time down from one week to a few hours as a means to boost their business. What happened though was a steep decline in customer satisfaction: customers with a negative notice would complain that their application might have been declined unjustifiably; customers with a positive notice would inquire whether their application had been checked with due diligence. This anecdote emphasizes the need to carefully test improvement hypotheses in practice because the customers and the process participants might not act as anticipated by the process analyst.

Information systems such as BPMSs are prime candidates for supporting the validation of improvement hypotheses. However, current BPMSs are not

designed for this purpose. Taking inspiration from existing work in the literature [2, 3, 7] and based on the arguments on risk mitigation, we identify a list of requirements for supporting the validation of improvement hypotheses by means of a BPMS. These include rapid validation, fair comparison, and rapid adjustment.

R1 Rapid validation: If it is uncertain whether an improvement hypothesis holds, the hypothesis should be tested immediately after deployment and within a short time frame.

R2 Fair comparison: An ad-hoc comparison of old and new process version is biased towards the specific conditions of the two time intervals $[t(n-1), t(n)]$ and $[t(n), t(n+1)]$, where $t(n)$ symbolizes the point in time when the old version was replaced with the new one. A fair comparison should avoid biases resulting from the characteristics of different time intervals.

R3 Rapid adjustment: The units of analysis should be different versions of a process model, an old and a new version in the simplest case. The system should rapidly adjust the allocation of customers to the version that has the best performance in the current context.

2.2. Prior Research

Prior research in the area of BPM and operations management proposes various approaches for improving business processes. They identify various focal points of analysis, but typically share the idea that the right analysis will yield an actual improvement. Most of the approaches do not consider a potential uncertainty of the improvement hypotheses. Here, we discuss some prominent approaches for process improvement to illustrate this point.

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 [8, 9]. BPR promotes radical changes that exploit new IT capabilities, overcoming the limitations of the old process design, and indeed throwing away the old design. BPR hardly discusses issues of uncertainty about the improvement hypothesis, but rather points to various managerial, technological, and contextual factors [10]. Second, approaches to *business process improvement* take a more cautious and more incremental approach [11]. The BPM lifecycle integrates process improvement into a continuous management approach [1]. This lifecycle puts a strong emphasis on modeling and analysis before engaging with redesign, for instance by reusing so-called best-practises [12, 13]. Once new process variants are implemented and rolled out, they are monitored. In case of unsatisfactory performance, there will be a new iteration to correct it. Extensions that enhance lifecycle phases independently [14] run into the problem of long iterations inherent in the lifecycle. The BPM lifecycle also assumes redesigns to be mostly driven towards fixing issues, which means that an incremental improvement is assumed.

From the area of operations management, we refer to approaches to quality management and lean management. *Quality management* as a neighbouring

discipline of BPM puts a strong emphasis on controlling process instances by the help of inspection, statistical process control, and other approaches to quality assurance [15, 16]. Quality management shares an analytical focus in order to identify root causes of insufficient quality, and then improve on them. Root cause analysis acknowledges the fact that there can be several hypothetical causes under investigation, from which the right one has to be singled out [1]. *Lean management*, invented for the Toyota Production System [17], also has an analytical emphasis focusing on a broad interpretation of the term "waste" [18]. Several techniques can be used to identify waste, though mostly with an emphasis on qualitative analysis [19, 1]. The success of process improvement using lean management is attributed, among others, to management support and communication [20]. Also, there is an implicit assumption that the right analysis leads to the right redesign decisions.

Approaches from computer science, and software engineering more specifically, are more cautious about the need for testing. A prominent example in this regard is DevOps [21], which aims to better integrate the processes of software development (Dev) and operations (Ops). One DevOps practice is live testing, where new versions of the software are tested in production with actual users of the system. The most popular form of live testing is *AB testing*, where two versions (A and B) are deployed side by side and both receive a share of the production workload while being monitored closely. The versions in production are monitored and their data is then used to draw conclusions about the effectiveness of one version over the other, for instance in the form of increased revenue from higher click-through rates. So far, AB testing has mostly been used for micro changes of websites, like changing the color of a button [4, 3]. The effectiveness of this technique is surveyed by Kohavi et al. for the user interfaces of web applications [22, 3]. Testing techniques on Service Oriented Architectures (SOAs), especially on regression testing and testing for violations of quality of service [23, 24], can be useful in identifying issues that can only be detected after deployment. However, unlike AB tests, these approaches do not test whether changes lead to process improvements.

Beyond these more general approaches, there are several recent techniques that may inform the requirements of rapid validation, fair comparison and rapid adjustment. Rapid validation builds on the existence of a newly redesigned process version. Typically, such new versions are created by an analyst. It is also possible to automatically generate process versions by the help of the technique presented in [25] and deploy them for validation. The validation might also benefit from recent monitoring techniques such as [26, 27] or predictive analytics [28, 29, 30]. The quality of existing processes can be improved at runtime by monitoring and dynamically adjusting service selection, resource allocation, and relevant parameters [31]. Process simulation techniques can also be useful for the validation task. Tools like BIMP[1] use parameters for modeling workload, resources, timings, branching probabilities, and other relevant metrics for

[1] http://bimp.cs.ut.ee/ Accessed: 15-01-2018

Table 1: Support for requirements in prior research

Approach	R1	R2	R3
Business Process Re-engineering (BPR) [8, 9]	-	-	-
Process Improvement [11, 14]	-	-	-
Quality Management [15, 16, 1]	-	-	-
Lean Management [17, 18]	-	-	-
Process Lifecycle [1, 14]	+/-	-	-
Monitoring and Predictive Analytics e.g. [26, 28, 29, 30]	-	-	+
Process Simulation [32, 34]	-	-	-
AB testing e.g. [21, 3]	+	+	-
AB-BPM (this work)	+	+	+

generating traces. Advanced simulation techniques can semi-automatically extract some of this information from the historical logs of a process and construct models that can be used for simulation [32, 33] or prediction [34].

Table 1 summarizes the support of the requirements by prior research. In this paper, we adopt the idea of AB testing on the process level in order to address requirements R1- R3 in a suitable way. Our technique is called AB-BPM and addresses the research gap related to the explicit testing of improvement hypotheses in BPM-related research and the lack of an explicit consideration of business processes in the works on AB testing in software engineering.

3. AB-BPM Approach and Methodology

In this section, we present the AB-BPM methodology and the technical solutions that enable it. The first of these solutions is simulation, for which we discuss how we extract decisions and metrics from the event log of a process and use that to simulate new versions. Then we discuss the mapping of the instance routing problem to algorithms from the literature. Based on an experiment, we choose one algorithm and adapt it to the context of business processes. Finally, we present our high-level framework, architecture, and implementation.

3.1. The AB-BPM Methodology

The AB-BPM methodology extends the redesign, implementation, and execution and monitoring phases of the business process life-cycle. This extension aims to provide support for rapid validation of process improvement ideas. Fig. 1 summarizes this methodology.

First, the redesign goal and the Process Performance Indicators (PPIs) are defined, followed by the design of the new version. Ideally this is followed by simulating the new version, using data from the old version. Simulation provides rapid feedback on the effect of the changes. However, it is not always possible to have a meaningful simulation: that requires the models to be reasonably similar to one another. If the models are too different (which is assessed in

the step "Compare versions" in Fig. 1), we advance to the AB testing stage directly. If the models are similar enough for simulation, the simulation can have satisfactory results or not. In the latter case, the new version is further improved. It should be noted that simulation is always approximate, and the fuzziness of the results should be taken into account in this decision. Once the results are satisfactory, we advance to the AB testing stage. For AB testing, the PPIs are summarized in a numerical value that acts as a feedback, or reward, which helps the instance routing algorithm make routing decisions during AB tests. Next, the new process version is deployed alongside the old version so that they run simultaneously in production. Finally, AB tests are configured and executed. The best performing version is automatically found by the instance routing algorithm. If the old version was found to be better than the new version, the new version is further improved and tested.

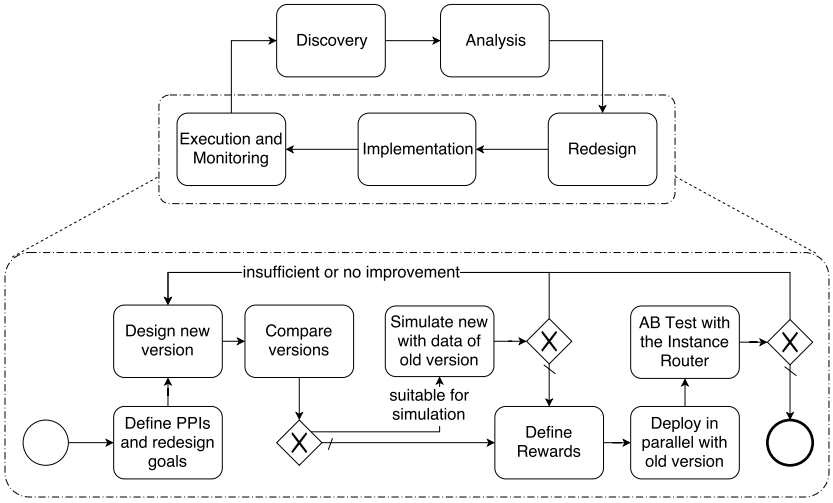


Figure 1: The AB-BPM methodology for process improvement. Adapted from [35].

3.2. Trace Simulation

Simulating the new process version is a good first step towards not only finding potential flaws in the version, but also realizing the design and configuration of the AB testing experiments. The simulation step in the AB-BPM methodology should estimate the performance of the new process version without making explicit assumptions about the workload, the customers, or the resources. Insights from the execution of the old version can help us answer how the new version would have performed under the same circumstances with minimal explicit assumptions.

Process runs are stored in the form of event sequences, namely *traces*, where events bear the information about activity executions (activity name, and other attributes like timestamp, operating resource, etc.). Collections of traces are

Table 2: An example of event log

Case id	Activity Name	Cost	Start Time	End Time
1	a	100	01-01-2018 00:00	01-01-2018 01:00
	b	50	01-01-2018 02:00	01-01-2018 04:00
	c	150	01-01-2018 04:00	01-01-2018 05:00
	b	25	01-01-2018 05:00	01-01-2018 05:10
	c	75	01-01-2018 05:10	01-01-2018 05:25
2	a	150	01-01-2018 10:00	01-01-2018 12:00
	d	50	01-01-2018 12:00	01-01-2018 13:00
	c	100	01-01-2018 13:00	01-01-2018 14:00

called *event logs* [36, Ch. 2]. The sequence of activity names in a trace will be henceforth called *activity sequences*. Table 2 shows a sample event log. We say that $\langle a, b \rangle$ is an activity sequence of the trace corresponding to case id 1. We use decisions and metrics extracted from the traces of the old version of a process and progressively generate traces of the new version.

3.2.1. Inferring Decisions and Metrics

We construct a Transition Simulation Tree (TST), a rooted tree data structure that summarizes decisions and metrics available in an event log. Figure 2 shows an example of TST constructed using the traces from Table 2.

The design of TST is motivated by two key observations. First, the decisions and metrics depend on what activities were executed previously. For instance, when a model allows for loops, the second iteration could on average be faster than the first. Second, a new version of a process can produce traces that are highly similar to the original process, but these traces may not match exactly. For example, a re-sequencing of two activities on the new process model can produce traces that never match with traces from the original version. The TST allows us to find partially matching traces, and derive metrics from these matches. Models such as Generalized Stochastic Petri-nets are not sufficient because these models summarize and generalize the execution of the original version [34], and finding adequate partial matches would be a challenge.

A node in the tree consists of an activity profile and a transition probability. An activity profile is composed of the activity name and a collection of metrics such as cost, duration, and waiting time. The metric collection contains raw data extracted from the event log a process. Transition probability of a node dictates whether the activity represented by the node can be followed by another activity. In addition, edges in the tree have a Bayesian transition probability. Given that a transition from a node is possible, the probability on a given outgoing edge indicates the odds of transitioning to the respective child node. These probabilities are based on the frequency of activities in traces extracted from the event log a process.

Recurring sequences of activities in the log are combined in the tree by adding metrics of the activity into the metric collection of the matching activity profile. For example, in Fig. 2 all activity sequences from Table 2 can be found as a path in the TST. Both of our traces start with activity a. Therefore, the

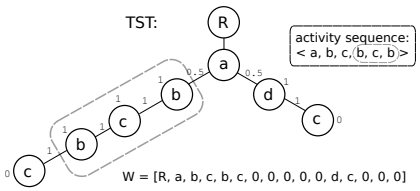


Figure 2: Transition Simulation Tree (TST), its prefix representation, and the longest prefix match.

Table 3: Auxiliary table with activity profiles in the TST of Fig. 2.

Activity	Cost	Duration	Wait.time
a	100	1 hr	0 min
	150	2 hr	0 min
b	50	2 hr	1 hr
	25	10 min	0 min
c	150	1 hr	0 min
	75	15 min	0 min
	100	1 hr	0 min
d	50	1 hr	0 min

activity profile of this node contains metrics from both traces, i.e., cost of 100 from case id 1, and 150 from case id 2. In the traces, activity a is always followed by b or d, and the proportion of traces that transition to b is the same as that of d. Therefore, the transition probability of node a is set to 1, and those of the corresponding edges are set to 0.5 each.

In addition to the TST, we store metrics in auxiliary tables as a fallback mechanism. For each activity, the tables store all observed metrics from the given log. Should the TST be insufficient for finding activity metrics, we can perform a look-up on these tables and sample the desired metric. For example, in Fig. 2 there are three distinct nodes for activity c which contain cost metric of 150, 75 and 100 respectively. In contrast, the auxiliary table contains all of these values for activity c regardless of the position of c in the traces.

Traces that have been seen during the construction of the TST can always be estimated by looking up the corresponding path in the tree. However, a process model that differs from the source of the log or that contains loops can produce an activity sequence that cannot be matched in the tree. In such cases, we can find the longest prefix match between the activity sequence and the TST. For example, the process model associated with Fig. 2 can produce an activity sequence $S = \langle a, b, c, b, c, b, c \rangle$. However, the TST only contains trace $S' = \langle a, b, c, b, c \rangle$. When a simulated process reaches a state of $S_{\text{new}} = \langle a, b, c, b, c, b \rangle$, the metrics and transition for the last task, i.e., b in S_{new}, cannot be found by parsing the TST. In this case, for S_{new} the longest prefix match would be $\langle b, c, b \rangle$. We take this part from the TST and sample the metrics and transition probability for activity b.

The longest matching prefix is found by searching for an exact rooted subtree match using the pre-order list representation of trees [37]. The pre-order list representation W is recursively defined as follows: $W = [l, 0]$ or $W = [l, W, 0]$ where l is a node. On pre-order traversal, node l is added to W until no child nodes exist. On every backtracking step to the previous level, 0 is added to W.

Algorithm 1 outlines how pre-order lists can be used to estimate metrics using the longest prefix match. Given the TST and an activity sequence as pre-order lists, the algorithm iteratively finds a match from the tail of the activity sequence and samples the metric. When a match can no longer be found, the

algorithm returns the last sampled metric. An example of longest prefix match is highlighted in Fig. 2. In addition to sampling a metric, we can also find the next activity to be executed by choosing a child of the last node in the longest match. The choice is made according to the edge probabilities in the TST.

Algorithm 1: Activity metric estimation with longest prefix match

```
   Input: W, W_l          // prefix representation of TST and an activity sequence
1  forall i in 2...len(W_l) do
2      W_p = W_l [len(W_l) − i :]  // tail of the activity sequence
3      match = a match of W_p in W using [37], selected at random
4      if match exists then
5          metric = samples from last event in match
6      else
7          return metric
8  return metric
```

The next activity and the metrics associated with an activity depend only on the the last node in the longest match. Also, the number of nodes and edges in the TST is finite. In this regard, the tree holds the Markov property, and can be seen as a type of Markov chain. The TST also bears resemblance with Frequency-enhanced Prime Event Structure (FPES) [38], where the event log is represented as graph of nodes composed of activity labels and frequencies, and edges contain occurrence probabilities. The TST does not encapsulate binary relations like FPES, but it contains activity metrics and allows partial matching. Partial matching for FPESs is discussed, e.g., in [39], but focusing on similarity in general. The partial matching we propose here is subject to other criteria, such as long connected chains. Finally, FPESs aggregate separate traces – which we refrain from in the TST, to capture performance metrics in maximally tight contexts.

3.2.2. Simulating New Versions

A simulator can generate traces of new process versions using the historical event log of the old version. The simulator consists of two main parts - the BPMS and the TST. Given the process model of the new version, the BPMS can infer which activities can be performed next from the current state. The BPMS uses the current activity sequence and these candidate activities to query the TST for the next activity to execute, and the estimates for the activity. In order to drive the execution using this mechanism, the process model is updated with conditions on every outgoing flow of a gateway such that setting a variable with the next activity will enable the appropriate outgoing flow.

Non-conforming logs can be found using the longest prefix match mechanism. In cases where multiple matches are found, one can be chosen based on the depth of the match in the TST, proximity to the end of a trace of the old version, or at random. The heuristics for breaking ties depends on the goal of the simulation. For instance, if we want to minimize the number of loops in simulated traces, a match can be chosen based on proximity to the end of a trace.

During simulation, there can be many activities that can be executed from the current state. We refer to these activities as candidate activities. New process versions may be structured in such a way that the candidate activities in the TST (the child nodes) and those in the process model of the new version are different. To drive the execution in such cases, the transition probabilities have to be adjusted to accommodate such differences. Algorithm 2 shows how we make such adjustment.

Algorithm 2: Transition probability adjustment

Input: C_{TST}, C_{new}, p `// sets of candidate activities, and probability allocation for new candidates`

1 $C_{\text{new}} = C_{\text{TST}} \backslash C_{\text{new}}$
2 $C_{\text{absent}} = C_{\text{new}} \backslash C_{\text{TST}}$
3 $C_{\text{common}} = C_{\text{new}} \cup C_{\text{TST}}$
4 $pr_{\text{d}} = \sum\{Pr(x) \mid x \in C_{\text{absent}}\}$
5 distribute pr_{d} to C_{common} in proportion to their existing probabilities
6 allocate p to C_{new} equally
7 normalize probabilities of C_{common} to scale of $[0, 1-p]$
8 return $C_{\text{common}} \cup C_{\text{new}}$

First we find the common set of candidates between the new process model and the TST. Transition probabilities for common candidates can be found in the TST. Some candidates may be present in the TST, but not in the new model. We extract probabilities of these candidates and distribute them to the common candidates in proportion to their existing probabilities. In every simulation run, we set aside a probability allocation p for candidates from the new model that are not in the common candidates set, and divide p equally among them. Then, we normalize the probabilities of the common set to the scale of $[0, 1-p]$. If the common candidate set is empty, then p is set to 1 and divided equally among the candidates from the new model. Transition is made to one of the candidate activities in the new model based on these probabilities.

For example, consider that we find the partial match shown in Fig. 2. The matching node has a transition probability of 1, and the candidate activity set $\{c\}$ with the probability of 1 on the corresponding edge. If the new process model has the candidate set of $\{c, e, f\}$ and we set $p = 0.2$, the probability of transitioning to c is 0.8, e is 0.1, and f is 0.1.

The above mechanisms are not sufficient to derive metrics for new activities introduced in the new process model. For those, estimates have to be provided by the user during the simulation setup. These estimates are stored in the auxiliary tables, and queried during simulation. If the tables do not contain information about a new activity that is encountered during simulation, the simulation run is aborted and the issue is recorded.

3.3. AB Testing

AB testing assesses the performance of the new version without making any assumptions. This is accomplished by exposing the new version to real

customers through the production environment. For risk management, the AB-BPM methodology advocates AB testing after simulation if possible. To further minimize the risk of exposing users to a sub-optimal version, our AB testing approach performs dynamic routing of user requests based on the observed performance of process versions.

3.3.1. Instance Routing – a Multi-Armed Bandit Problem

In order to integrate concepts of process execution with AB testing, we have to discuss how new instances are assigned to a specific version of the process. Therefore, we need an instance router that distributes requests to versions in such a way that any relevant PPI is maximized. The instance router also needs to deal effectively with the issue that processes can be long-running, and that PPI measurements can be delayed.

The PPI maximization can be mapped to the so-called *multi-armed bandit problem* [40, 41]. The multi-armed bandit problem models a hypothetical experiment where, given some slot machines with different payoff probability distributions, a gambler has to decide on which machines to play. The objective of the gambler is to maximize the total payoff during a sequence of plays. Since the gamblers are unaware of the payoff distribution, they can approach the plays with two strategies: *exploring* the payoffs by pulling different arms on the machines or *exploiting* the current knowledge by pulling arms that are known to give good payoffs. The exploration strategy builds knowledge about the payoffs, and the exploitation strategy accumulates the payoffs. Multi-armed bandit algorithms aim to find a balance between these strategies. If the performance is affected by some context, this can be seen as the so-called *contextual multi-armed bandit problem*, where the gambler sees context (typically represented as a multi-dimensional feature vector) associated with the current iteration before making the choice.

We model the routing algorithm as a multi-armed bandit problem by representing the process versions as the "arms", and the PPI as "payoffs/rewards". The objective of the instance router is to find a good trade-off between exploration and exploitation, possibly based on the context. To learn the performance of a version in exploration, it sends some of the process instantiation requests to either version. Based on the instance router's experience, it can exploit its knowledge to send more or even all request to the better-performing version. The reward for the routing algorithm can be designed around a PPI like user satisfaction. We discuss these topics in more detail below.

3.3.2. Instance Routing Algorithms & Selection

The multi-armed bandit problem has been explored in related literature. LinUCB [42] is a contextual multi-armed bandit algorithm that has been employed to serve news articles to users with the objective to maximize the total number of clicks. Thompson sampling [43] is one of the simplest approaches to address multi-armed bandits. It is based on a Bayesian approach where arms are chosen according to their probability of producing optimal rewards [44, 43].

Extensions of Thompson sampling can be used to solve the contextual multi-armed bandit problem with linear rewards [41]. In this paper, we chose three algorithms – see below – as candidates for process instance routing and investigate their effectiveness. We have selected these algorithms based on their demonstrated benefits and simplicity. Other algorithms, such as ϵ-greedy, ϵ-first, UCB, and EXP4 also address multi-armed bandit problems [40].

Since the goal for the routing algorithms is to maximize an aggregate value of the PPI, as preparatory work, we have experimented with different routing algorithms with different configurations to find the best performing algorithm. We have compared the Thompson sampling technique [44, 43], its extension for contextual bandits [41], LinUCB [42], and a baseline algorithm which uniformly distributes requests to process versions regardless of context and rewards.

We designed an AB testing experiment in which the routing algorithms dynamically allocate requests at run-time to two very simple business process versions, version A and version B. Based on the context (type of requests) these versions return a user satisfaction score between 1 and 5. In addition, the execution time of each process is set to 60 seconds. The key idea behind this design was to have two processes that behave deterministically so that the routing algorithms are not affected by the randomness of rewards and delay times. We devised six request types determining the context, which were embedded in the requests forwarded by the routing algorithm to the process versions. In our design, Version A produced a higher satisfaction score in five out of six cases. However, in one case version B outperformed version A by a large margin. With this setup, we tested the algorithms by sending 500 requests with embedded contextual information to at the rate of 1 request per second.

The experiment results showed that contextual bandits performed well, and that *LinUCB produced the highest cumulative satisfaction score throughout the experiments*. Therefore we have selected and adapted LinUCB for typical business process scenarios. Our architecture is flexible enough that it can be easily replaced by other algorithms.

3.3.3. Adapting the Routing Algorithm to Business Processes

As discussed above, we chose LinUCB as the starting point for our routing algorithm. However, we observed that the algorithm can be derailed by process-specific circumstances, such as the long delays before rewards. Long delays are inherent to long-running processes, and not considered in AB testing solutions for Web applications, where delays are measured in seconds or minutes. In contrast, the real-world data which we use in the evaluation has one process instance with an overall duration of more than 3 years.

This results in the following issue. Oftentimes overly long process completion times correlate with problematic process instances, leading to negative rewards. Thus, instances with short completion times can give an overly positive impression of a process version early on. If the algorithm receives too many positive rewards from one version during the early stages of the AB test, the algorithm is more likely to see that version as preferable. Such an early determination can

Algorithm 3: Instance Routing with LTAvgR

Input: $\alpha \in \mathbb{R}_+$, $\lambda \in \mathbb{R}$, $M \in \mathbb{N}$
// α is the LinUCB's tuning parameter; λ and M are experimentation decay and length
Output: arm id

for *t = 1, 2, 3, ..., T* **do** // t is the request count
  Observe features of all arms $a \in A_t : x_{t,a} \in \mathbb{R}^d$
  for $a \in A_t$ **do**
    if *a is new* **then**
      $A_a \leftarrow I_d$, $b_a \leftarrow 0_d$ // identity and zero matrices of dimension $d \times d$, resp.
    $\hat{\theta}_a \leftarrow A_a^{-1} b_a$
    $p_{t,a} \leftarrow \hat{\theta}_a^\intercal \, x_{t,a} + \alpha \sqrt{x_{t,a}^\intercal \, A_a^{-1} \, x_{t,a}}$
  arm $a_{linucb} = \underset{a \epsilon A_t}{\operatorname{argmax}} p_{t,a}$ with ties broken arbritarily
  if $t \leqslant M$ **then** // experimentation phase
    $pr_{exp} \leftarrow$ sample y from $Exp(\lambda)$ s.t. $x = t$
    Choose arm $a_t = a_{linucb}$ or $a_{alternate}$ with probability pr_{exp}
    Poll for reward r_t
    $A_{a_t} \leftarrow A_{a_t} + x_{t,a_t} \, x_{t,a_t}^t$
    $b_{a_t} \leftarrow b_{a_t} + r_t x_{t,a_t}$
  else
    Choose arm $a_t = a_{linucb}$

introduce a bias in the evaluation. Thus, we need to ensure that the algorithm gets enough samples from both versions.

We solve this issue by adopting the idea of a "warm-up" phase from Reinforcement Learning [45], during which we emphasize exploration over exploitation. We sample the probability of exploration by using an exponential decay function, acting as an injected perturbation that diminishes as the experiment proceeds – the sample determines whether the algorithm follows the instance routing algorithm's decision or picks a version at random. We consider the "warm-up" as the *experimentation phase:* after all instances started during the experimentation phase are completed, no more rewards are collected and the instance router stabilizes.

Finally, the original LinUCB algorithm makes its decision based on the summation of past rewards. We found out during the experiments that this can also deceive the algorithm. Therefore, we have modified the LinUCB algorithm to make its reward estimates on the basis of the average of past rewards rather than their sum. We term our adapted instance routing algorithm *Long-term average router (LTAvgR).* Algorithm 3 shows the pseudo code of LTAvgR.

3.3.4. Reward Design for Long Running Processes

Routing behaviour of LTAvgR is affected by the value of the reward, and also by factors associated with time. Delay between instantiating a process and receiving a reward, and the rate of receiving a reward vs. the rate of process instantiation influences the reward estimates at any given time.

If we use duration of a process instance as the PPI, we need to be careful in deriving rewards from the raw values. The raw value of duration is too fine-grained to be meaningful as a reward because of two reasons. First, even minor differences in duration can cause the instance router's decision to fluctuate. Second, it is implied that magnitude of difference in duration of two process executions is the same as the difference in their quality. The effect of rewards derived from small number of slow process instances can outweigh the effect of rewards derived from others. Also, we will observe that good rewards come early and bad rewards come late, which further exaggerates these problems if the rewards are not well designed.

A good reward strategy should penalize poor performances, but the penalty should not be so high that it overwhelms the estimates derived from the majority of execution. The granularity of rewards should be chosen such that unnecessary fluctuation is avoided but no meaningful information is lost. This is particularly important in scenarios where the raw value of PPI (duration) for outliers can be magnitudes higher than the average.

To establish a reward function for scenarios where duration is an indicator of quality, we adopt the following strategy. Assume that we are AB-testing P^i, the original version, vs. P^j, the new version. We collect all instance durations reported in the production data from executions of P^i and compute $K \geqslant 1$ quantiles $q_1, \ldots, q_K$. We use these quantiles to partition the space of possible durations into a set of $K+2$ intervals $I = \{\iota_0, \iota_1, \ldots, \iota_K, \iota_{K+1}\}$ where $\iota_0 = [0, q_0)$, $\iota_{K+1} = [q_K, +\infty)$, and $\iota_k = [q_{k-1}, q_k)$ for $1 \leqslant k \leqslant K$. Those intervals split the range of possible durations for P^j as follows: ι_0 contains the values below the minimum, and ι_{K+1} accounts for any duration beyond the maximum recorded in the production data. The K intervals in-between are meant to classify the performance of P^j. This strategy is illustrated as a step chart in Fig. 3 – every step in the chart represents a quantile.

The idea is to assign a reward of 1 when the duration P^j achieves is lower than any registered duration of P^i, and decrease it by a step of $2/K$ as long as the measured performance falls into the following intervals. In order to counterbalance the disruptive effect of outliers which take have extremely long durations, we establish a penalty value ρ (with $\rho = 4$ in Fig. 3) and define that the reward linearly decreases from -1 to $(-1-\rho)$ along ι_K. Finally, any duration beyond the last quantile is assigned a reward of $(-1-\rho)$.

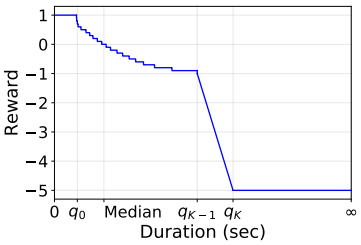


Figure 3: Reward strategy

Formally, let $\kappa_I(t) : \mathbb{R}^+ \to [0, \ldots, K+1]$ be a mapping function associating a process instance duration t to the respective interval $\iota_K \in I$ by the index k. That is, $\kappa_I(t) = 0$ if t is in the range of ι_0, $\kappa_I(t) = 1$ if t is in the range of ι_1, and so on. The reward function $r_I : \mathbb{R}^+ \to [-1-\rho, 1]$ is defined over the set of

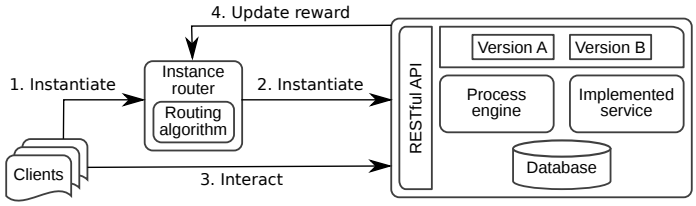


Figure 4: Application architecture

intervals I as follows:

$$r_I(t) = \begin{cases} 1 - \frac{\kappa_I(t)}{K} \cdot 2 & \text{if } \kappa_I(t) < K \\ -1 - \rho \cdot \frac{t - q_{K-1}}{q_K - q_{K-1}} & \text{if } \kappa_I(t) = K \\ -1 - \rho & \text{if } \kappa_I(t) > K \end{cases} \quad \text{with} \quad \kappa_I(t) = \sum_{k=0}^{K+1} k \cdot \chi_{\iota_k}(t)$$

where χ_{ι_k} is the characteristic function of interval ι_k. The underlying idea is to prefer the process demonstrating a shorter completion time to the slower ones while accounting for the outliers.

Fig. 3 shows a reward strategy when we set $K = 20$ and $\rho = 4$. With this configuration, our rewards are divided into 20 classes which incrementally differ from each other by a value of 0.5, except for the last interval where the rewards scale linearly. In this configuration, the slowest process instance is considered to be worse than the average by a factor of 5, but the fastest process is considered to be better than the average only by a factor of 1.

Many processes are long-running by design. To overcome the problem of time dependence of rewards for such processes, we can either use PPIs that do not use completion time (e.g., duration until some other activity is completed), or lead indicators [30].

3.4. AB Testing Framework, Architecture, and Implementation

Fig. 4 shows the architecture of our AB testing framework. We designed the architecture such that the two versions of the process model are deployed side by side in the same execution engine. The instance router distributes the instance creation requests as per its internal logic. All subsequent requests related to a given process instance are routed to the version chosen initially for that instance.

An alternative design would be to run two full copies of the entire application stack, one for each version, and using the instance router to distribute the requests across the two stacks. However, the multi-armed bandit algorithms can identify the superior version during the experimentation and alter the allocation of requests to different versions. When a version is clearly superior to the other, most of the requests are sent to the superior version. In such scenarios, the application stack that hosts the inferior version is underutilized. If we run both versions on the same stack, we can keep utilization of the system high, no matter which version is superior.

Given this design choice, the process definitions, implementation, PPI collection, and the shared process execution engine are wrapped by a web application.

Process execution data are stored in a shared database. Process instantiation, metrics, and other operations are exposed using RESTful APIs. Upon receiving a request, the instance router instantiates a particular version and receives an identifier. Identifiers of the process instances for which rewards have not been observed are stored in a queue. The instance router uses a polling mechanism in parallel to retrieve PPI metrics from the server and update the rewards.

We implemented the architecture prototypically in Java and Python, in part based on the Activiti BPMS. As outlined earlier, our framework is flexible in the choice of the instance routing algorithm: we implemented and tested LTAvgR, LinUCB, Thompson-sampling with and without linear rewards, and random uniform distribution. The experiments reported in the following section are run with the presented implementation with LTAvgR. We simulate the requests from users by replaying process logs.

4. Evaluation

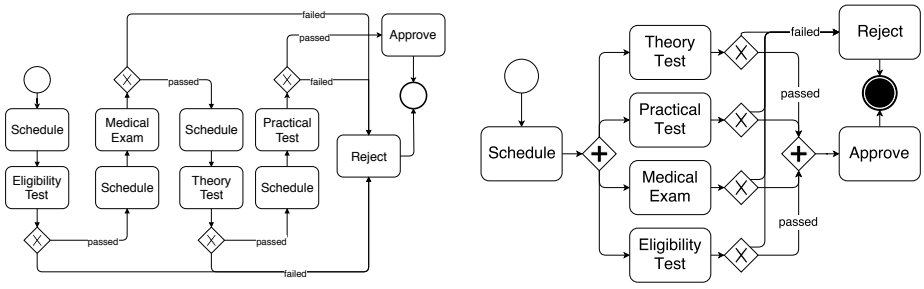
In this section, we present the methodology and the outcomes of our evaluation of the proposed approach. We assess the AB-BPM methodology in terms of simulating a new process version using the log of the original version, as well as AB tests using the LTAvgR algorithm and find the best performing version. We use three datasets: synthetic data (helicopter licensing process, *HL*), three versions of a real-world loan approval process from a Dutch bank (*Bank*), and five versions of a real-world building permit processes from five Dutch municipalities (*Permit*). On the third dataset, we only perform AB testing; on the other two we both simulate and perform AB tests.

4.1. AB-BPM Methodology on Synthetic Processes – Helicopter Licensing

In this section, we demonstrate our approach using two example process versions stemming from the domain of helicopter pilot licensing (HL). Version A of the process sequentially schedules the activities based on the cost of performing them. Based on the result (pass/fail), the process either schedules the next activity or terminates. In this version, we expect that successful candidates will pay more because of multiple scheduling costs. In contrast, version B of the process schedules all such activities at the beginning, thus reducing the scheduling costs. The processes are illustrated in Fig. 5. These processes have as a result the final status of the license: either *approved* or *rejected*.

As PPI we simulate the user satisfaction, here calculated as a combination of cost, completion time, and result of the process execution. Cost and processing time of tasks were derived using rates from Australian Civil Aviation Safety Authority (CASA) [2] and helicopter hiring rates from an Australian flight school.

[2] Civil Aviation (Fees) Regulations Amendment F2016L00400 https://www.legislation.gov.au/Details/F2016C00882, Accessed: 03-01-2018



(a) Version A

(b) Version B

Figure 5: Helicopter Licensing process versions

Success rates were derived using application statistics reported by CASA[3]. Note that these success rates are different from the ones used in the original version of this paper [5]. Table 4 shows the costs and processing times for both process versions. Table 5 shows how user satisfaction scores from 1 (lowest) to 5 (highest) are derived. The basic rationale is, the shorter and cheaper, the better. The score ranges from 1 to 3 if the outcome is negative, and from 3 to 5 if the outcome is positive.

Table 4: Cost model of the activities, HL

Activity	Cost	Min. processing time	Max. processing time	Success Rate
Schedule	25	1 day	1 day	N/A
Eligibility Test	190	1 day	3 days	99.7%
Medical Exam	75	1 day	3 days	99.7%
Theory Test	455	2 weeks	5 weeks	72.1%
Practical Test	1145	1 week	2 weeks	33.2%
Approve	100	Immediate	Immediate	N/A
Reject	0	Immediate	Immediate	N/A

4.1.1. Simulation

We have designed the simulation experiments such that the TST is constructed from the log of Version A. This log is obtained by executing version A in the BPMS with a workload of 1 request per second. For the execution, we sped up time such that each day corresponds to 1 second. We also introduce some non-determinism by sampling processing times for each activity using a probability distribution function. The TST constructed from this log is used to guide the simulation of version B. We set $p = 0.5$ as the probability allocation for new activities during the simulation.

[3]CASA Annual report 2016-2017 https://www.casa.gov.au/sites/g/files/net351/f/annual_report1617.pdf?v=1508473202, Accessed: 15-01-2018, Australia wide pass rates https://www.casa.gov.au/standard-page/australia-wide-pass-rates, Accessed: 15-01-2018

Fig. 6 shows the estimates of the cost and the duration for version B from simulation. We can observe that version B has good potential for reduction of mean cost and duration. However, the variance in measurements are high. Since the new version introduces a possibly large number of unseen traces, we should view these results with a degree of scepticism. The simulation was based on probabilities extracted from version A, where the activities are strictly ordered. Therefore, the simulation may not produce traces with proportions reflective of production. In particular, simulation of version B may produce more traces for rejected applications, which run with a shorter duration and lower cost. We can use this simulation as a sanity check for the new process version and conclude that the new version performs reasonably and provides some potential for improvement in cost and duration. Therefore, as per the AB-BPM methodology, we can perform AB tests.

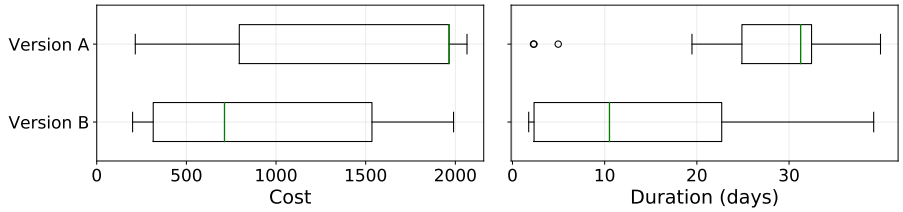


Figure 6: Comparison of simulated metrics of HL Version B with real data of Version A

4.1.2. AB Testing with LTAvgR

We have designed the AB testing experiments such that the instance router receives requests with embedded contexts at a rate of 1 request per second. The length of AB test is set to 1000 requests, where the experimentation or "warm-up" phase is set to $M = 500$ requests and decay to $\lambda = 150$. Figure 7 shows the cumulative request distribution throughout the AB test.

Despite the length of the experimentation phase and high value of λ, the instance router settles on a version after roughly 200 user requests. A post-hoc analysis shows that the median user satisfaction across all cases was the same

Table 5: User satisfaction model, HL

Outcome	Cost	Duration	Sat.
Approved	$[0, 1990]$	$\leqslant 5$ weeks	5
	$(1990, \infty]$	$\leqslant 5$ weeks	4
	$[0, \infty)$	> 5 weeks	3
Rejected	$[0, 1890]$	$\leqslant 5$ weeks	3
	$[0, 1890]$	> 5 weeks	2
	$(1890, \infty]$	$\leqslant 5$ weeks	2
	$(1890, \infty)$	> 5 weeks	1

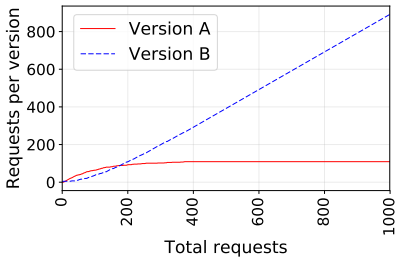


Figure 7: Requests routing in HL AB tests

Table 6: Analysis of HL Versions A and B by cases

Metric	Outcome	Version A	Version B	Overall
Samples(N)	All	101	899	1000
	Approved	25	214	239
	Rejected	76	685	761
Median user satisfaction	All	3	3	3
	Approved	4	5	5
	Rejected	2	3	3
Median cost	All	1965	1625	1700
	Approved	2065	1990	1990
	Rejected	1965	1360	1360
Median duration	All	32 d	28 d	29 d
	Approved	33 d	32 d	32 d
	Rejected	30 d	20 d	22 d

for both versions – but improved with version B if considering the approved and rejected cases in isolation. The distributions of user satisfaction scores also differed significantly (a Mann-Whitney test [46] resulted in U=25318, p-value $< 10^{-6}$ two-tailed, n_A=109, n_B=891). The median delay of the rewards was 36 seconds (corresponding to 36 days).

Table 6 shows the differences between the two versions. Version B is cheaper overall, but it is only faster when applications are rejected. Most of the rejections come at the late stage in version A – i.e. on the Practical Test. Since Practical Tests can be done earlier on version B, we expect such improvements in real execution. The instance router settles on Version B because it outperforms Version A on average.

Version B was cheaper overall, but not as much as that suggested by the simulation. Also, the median duration in AB tests was similar except for the rejected applications. This is the expected behavior, but the simulation suggested stronger improvements. This is attributed to the fact that the simulation is not aware of the decision logic internal to the process. The proportion of traces leading to rejection, especially from activities where success rate is good in reality, was higher in the simulation than in the AB test.

4.1.3. Discussion

The quality of estimates from simulation depends on how much the traces between process models can differ. Since the new version proposes an intuitive change, we used this simulation as a sanity check. In complicated process models, domain experts should be aware of how much the new version differs from the original before interpreting results of the simulation.

We used an evaluation based on synthetic processes in order to investigate the efficacy of our methodology. We observe that our simulation approach can be used as a sanity test for new process version. We also observe that our AB testing approach leads to a rapid identification of the more rewarding process version, which receives an increasing share of traffic. This observation is instrumental with respect to the requirements R1 - R3, which demand rapid validation, fair comparison, and rapid adjustment on the process level.

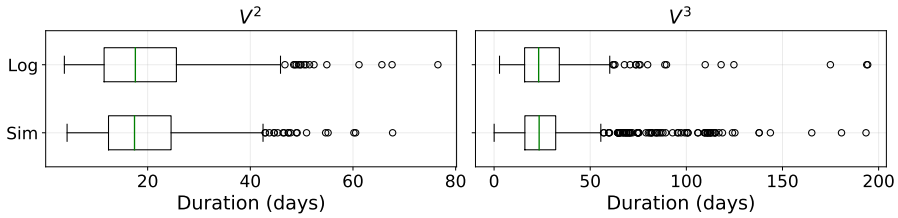


Figure 8: Simulation of Bank process versions using TSTs created from their event log

4.2. Simulation and AB Tests with Real-World Process Versions – Banking

In this section we evaluate our simulation and AB Testing technique using the loan application process models in banking (henceforth indicated as Bank) extracted from the event logs of BPI Challenge 2017.[4] We use three subsets of logs and treat them as versions. Version V^1 is extracted from process variants 1, 2, 3, and 9, as outlined in the awarded winning academic report from BPIC 2017 [47]. V^2 is a version that is made up of traces of variant 6. V^2 and V^1 use the same set of activities. Finally, V^3 is a version that is made up of traces that have a fraud detection step. This version contains some activities that do not exist in V^1 or V^2; V^3 has additional activities with the labels "W_Assess potential fraud", "O_Sent (Online only)", "W_Handle leads" and "W_Personal loan collection".

One key observation from the log is that the activities can be suspended, withdrawn, or aborted. During the construction of the TST, we summarize these state changes by aggregating all idle times as wait time and all active times as execution time, and then order the activities by their start time. The process models were discovered using Inductive Miner [36, Ch. 7] with the default noise filter of 0.2.

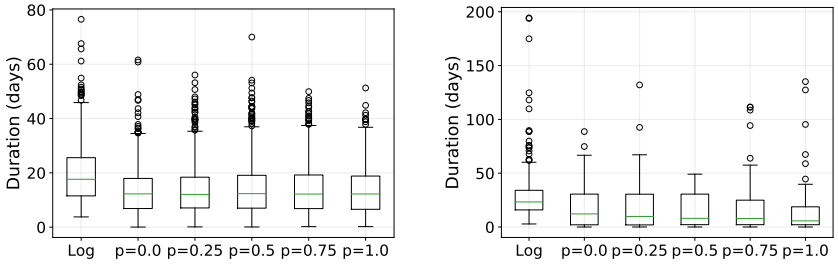
We start with the assumption that V^1 is deployed in production and that the historical event log of V^1 is available. The hypothesis is that the new versions, V^2 and V^3, are improvements over V^1. The following simulations and AB tests compare V^1 with V^2 and V^3 respectively.

4.2.1. Simulation

First, we perform a sanity test on the simulation technique itself: using its own historical event log as a basis to construct the TST, can the simulation produce traces with similar duration estimates as that of the log? Figure 8 shows the results from simulation of 1,000 traces. We can observe that the duration of simulated traces are similar to those from the historical event log.

Second, we simulate versions V^2 and V^3 using the TST derived from the event log of version V^1. As outlined in Section 3.2.2 , p is defined as the probability allocated to candidate activities that are present in the process model,

[4]BPI Challenge 2017, including logs, reports, and process models: https://www.win.tue.nl/bpi/doku.php?id=2017:challenge, Accessed: 15-01-2018



(a) V^2 (b) V^3

Figure 9: Simulation of Bank process versions V^2 and V^3, both with TST from V^1

but not in the TST. In this experiment, we vary p as $p \in \{0, 0.25, 0.5, 0.75, 1\}$. If multiple partial matches are found in the TST, we select one at random. Duration estimates for new activities in V^3 were extracted from the event log of V^3. Figure 9 shows the differences in duration of the 1,000 simulated traces for each value of p and the traces from the event log of the respective process.

We observe that this simulation produces optimistic results – the duration of simulated process instances are estimated to be less than the actual duration. The simulation does not necessarily produce the same distribution and outliers seen in the log. This can be attributed to the differences in duration of the new versions in production, which cannot be estimated using knowledge of the original process. For example, the median execution and waiting times for "W_Call after offers" activity is similar for all of versions. However, the minimum and maximums differ greatly. Minimum and maximum execution times in V^1 are ∼0 and ∼26 days respectively. However, for V^3, they are ∼16 and ∼80 days respectively. Another reason for the differences stems from the fact that these versions run in different contexts – our versions are in fact variants of the same process. Instances that need fraud detection, e.g., are served by a different variant than other applications.

Simulation results, though fuzzy, show that the performance of V^2 and V^3 can be better than that of V^1. In the following AB testing experiments, we investigate whether these versions are faster in practice and whether the routing algorithm can correctly identify the version that is faster.

4.2.2. AB Test

In this section, we compare duration of process versions V^2 and V^3 with V^1 through AB tests. Since the implementation details were not available, we have emulated the execution of process versions on the BPMS using the TST constructed from their own event logs. In Section 4.2.1, we have shown that such simulation produces results similar to that of the real execution. To ensure that the polling mechanism of the instance router and the reward delays are similar to that of a production environment, the PPI (duration) is made available only after the duration of a simulated process instance has elapsed. This allows us

to perform AB tests as if the implementation of the process were available.

As before, we sped up time such that 1 day in real time is equivalent to 1 second in the emulation. User requests are sent to the instance router at a rate of 1 request per second. Also as before, the length of AB tests is set to 1000 requests, the warm-up phase to $M = 500$, and the exponential decay parameter is $\lambda = 150$. We use quantile based reward strategy described in Section 3.3.4 with $K = 20$ and $\rho = 4$. The quantiles are derived using the historical data of V^1.

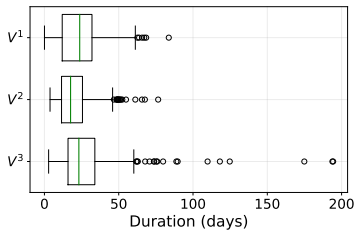


Figure 10: Duration of Bank process versions V^1, V^2, and V^3.

Figures 11 and 12 show request distribution, and the PPI observed by the instance router over time during the AB tests. Figure 10 shows a box plot of the duration of each process version extracted from the event logs of BPIC2017. We can observe that V^1 is slower on average than V^2 and V^3, though only marginally for V^3. In our AB tests, the routing algorithm is incentivised to send more requests to the version that is faster. These results show that the routing algorithm correctly identifies and converges to the best performing version.

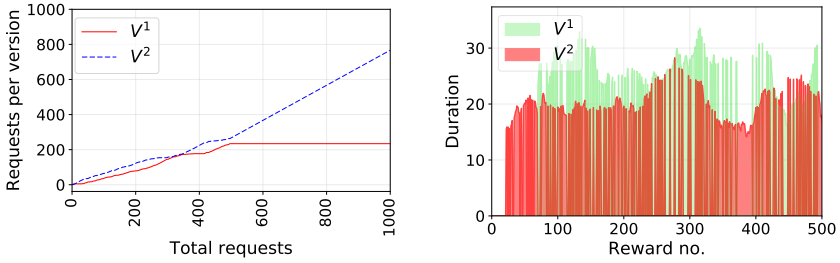


Figure 11: Request distribution and duration (smoothed) during warmup, Bank V^1 vs. V^2

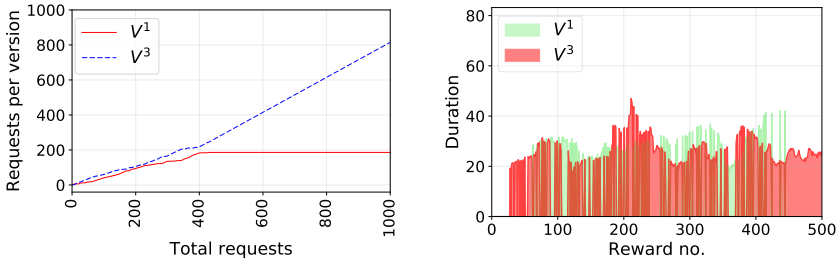


Figure 12: Request distribution and duration (smoothed) during warmup, Bank V^1 vs. V^3

4.3. Competitive AB Tests with Real-World Data – Building Permits

To further assess the applicability of our AB testing approach over real-world data, we have analysed the data stemming from the five logs in the BPI Challenge 2015,[5] herein identified as $L^1, \ldots, L^5$. Those logs contain the execution data of the building permit issuing processes in five Dutch municipalities. The processes behind each log are based on the same legislation but reportedly contain variations, which allow us to consider them as different versions of the same main process.

The complexity of these process versions and the lack of contextual information impedes us from creating accurate models for trace simulation. In the following experiment, we simulate the situation where one version is in use, when a new version is suggested and AB-tested in competition with the previous one. Better performance here is equated to shorter time to complete a process instance. Subsequently, the version that won the first round competes against the next version, and so on, until all versions have competed.

4.3.1. Execution Time Simulation

Based on the insights from [48], we filtered the events to retain only those activity instances that belong to a so-called "phase", namely constituting the core part of the process. We created TSTs for five process versions $P^1, \ldots, P^5$ from $L^1, \ldots, L^5$, respectively.

The events in the logs signal the completion of an activity, and bear eight timestamp fields. However, most of those attributes were missing or unreliable. Therefore, we followed the approach of [48], and used solely the completion *time:timestamp* attribute for each event. We computed the duration of every activity as the difference between the timestamp of its completion and the preceding completion timestamp. We thus summarized the execution time and the waiting time in a single metric, which we refer to as *duration* from here on.

We simulated the execution of the processes by sampling activity durations of conforming logs from the TST. For fairness, in this experiment we simulate only the logs that did *not* stem from the original processes. Say, we are AB-testing P^i vs. P^j; then we use the logs from $\mathcal{L}_{\text{test}} = \{L^1, \ldots, L^5\} \backslash \{L^i, L^j\}$ with $1 \leqslant i, j \leqslant 5$ and $i \neq j$. However, we want to test how the event traces from $\mathcal{L}_{\text{test}}$ behave on P^i and P^j in terms of timing. Therefore, for every conforming trace, we found a match in the TST, estimated the overall duration, and returned the duration information only after the estimated time has elapsed. We discarded all non-conforming traces, and did not perform partial matching on the TST.

4.3.2. Reward Strategy

The filtered BPIC 2015 dataset contains numerous outliers: while the median duration for processes are 39-46 days, outliers take up to 1158 days, i.e., 3 years and 63 days. In the following experiments, we use duration as the performance

[5] BPI Challenge 2015, including logs, reports, and process models: https://www.win.tue.nl/bpi/doku.php?id=2015:challenge, Accessed: 20-03-2017

Algorithm 4: Strategy for the selection of the best performing version among $\{P^1, P^2, P^3, P^4, P^5\}$.

$\mathcal{P}_{\text{test}} \leftarrow \{P^1, P^2, P^3, P^4, P^5\}$
$P^i \leftarrow$ original process version from $\mathcal{P}_{\text{test}}$
$\mathcal{P}_{\text{test}} \leftarrow \mathcal{P}_{\text{test}} \setminus \{P^i\}$
repeat
  $P^j \leftarrow$ alternative process version from $\mathcal{P}_{\text{test}}$
  $\mathcal{P}_{\text{test}} \leftarrow \mathcal{P}_{\text{test}} \setminus \{P^j\}$
  $\mathcal{L}_{\text{test}} \leftarrow \{L^1, \ldots, L^5\} \setminus \{L^i, L^j\}$
  $P^i \leftarrow$ best version between P^i and P^j as per the AB test over $\mathcal{L}_{\text{test}}$
until $\mathcal{P}_{test} \neq \varnothing$
return P^i

Table 7: Permit traces

Log	# of Traces
L^1	1199
L^2	830
L^3	1409
L^4	1051
L^5	1155

Table 8: Ratio of conforming traces in Permit

Version	L^1	L^2	L^3	L^4	L^5
P^1	1	0.928	0.949	0.974	0.928
P^2	0.913	1	0.928	0.982	0.938
P^3	0.901	0.812	1	0.975	0.886
P^4	0.873	0.731	0.913	1	0.829
P^5	0.897	0.929	0.944	0.979	1

metric and give higher rewards to faster process instances. We use the quantile-based reward strategy described in Section 3.3.4 with $K = 20$ and $\rho = 4$.

4.3.3. Competition: Selecting the Best Version

To simulate the situation where an organization gradually designs new versions of a process model, we run a competition between the five discovered process models. This competition is conducted as a set of pair-wise comparisons between versions, following the schema outlined in Algorithm 4. The idea is to initially consider an original version of the process, P^i, and a new version, P^j. To determine if P^j achieves an actual improvement over P^i while limiting bias as discussed above, the execution of the processes is simulated by replaying the traces in the logs from which P^i and P^j were *not* derived. For instance, P^1 and P^2 are evaluated on the basis of the traces in L^3, L^4, and L^5. If, at the end of a competition round, P^j demonstrated an improvement over P^i, then P^i is replaced with P^j. Otherwise, P^i is maintained. At that stage, another process version is compared to the winning version. The selection procedure continues until all process versions have competed. During the competition, traces which could not be replayed on the selected process version, i.e., were unfit, were discarded and their performance was not considered. The number of compliant traces still represents the vast majority, because the ratio of conforming traces of all logs over models remained around 0.9, and always above 0.7 as shown in Table 8. Also, the total number of traces per log is shown in Table 7.

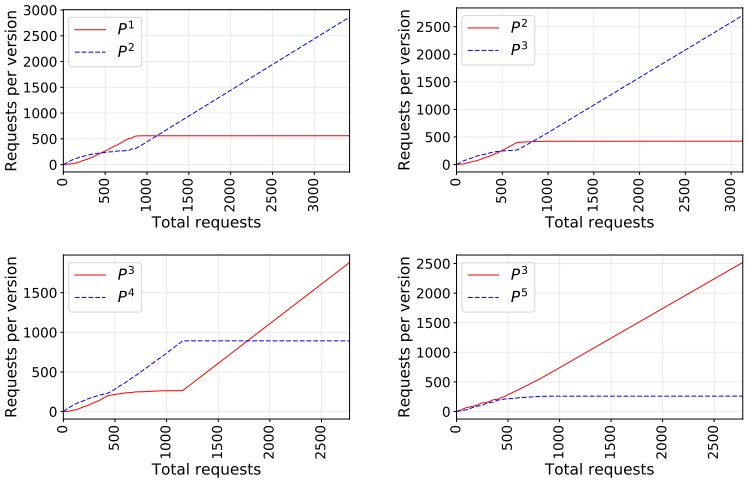


Figure 13: Request distribution for Permit over time

Table 9: Pair-wise performance comparison of Permit versions after the AB tests. Values for the faster version are shown in bold.

Metric	Round 1		Round 2		Round 3		Round 4	
	P^1	P^2	P^2	P^3	P^3	P^4	P^3	P^5
No. of requests	559	**440**	423	**575**	**263**	729	**735**	261
Median duration	33.8	**29.8**	28.8	**27**	**21**	21.85	**22.9**	27.9
Mean duration	55.3	**52.1**	51.8	**35.8**	**29.3**	49.9	**36.6**	38.3

4.3.4. Analysis

Without loss of generality, we began the selection considering P^1 as the process currently running on the production system, and progressively entering P^2, P^3, P^4, and P^5 into the competition as described above. Once more, we sped up the execution time such that one day in the trace was equated to one second in the experiments. We set experimentation phase length to $M = 1000$, and decay parameter $\lambda = 100$.

The sequence of tests was: (1) P^1 vs. P^2, P^2 wins. (2) P^2 vs. P^3, P^3 wins. (3) P^3 vs. P^4, P^3 wins. (4) P^3 vs. P^5, P^3 wins. We can observe that P^3 was the best-performing version. In all tests, the instance router chose the version with lower mean and median execution time.

Figure 13 shows the request distribution throughout the pair-wise tests. The experimentation phase ends roughly after 1000 requests in all cases. We can observe that occasionally the instance router decided to pick another version some time during the post-experimentation phase. In some cases, the decision made during the post-experimentation phase contradicted the decision during the experimentation phase. In these scenarios, the instance router was able to make the better decision only after all the delayed rewards were received.

In Table 9, we show the request distribution during the experimentation

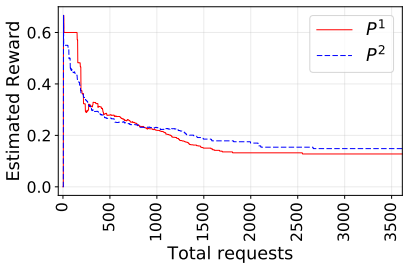


Figure 14: Estimated rewards during the Permit experiment P^1 vs. P^2

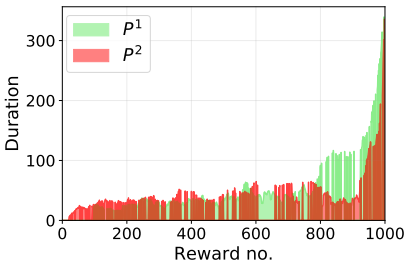


Figure 15: PPI (duration) during the warm-up phase of Permit, smoothed.

phase, and the performance metrics calculated using execution times of processes instantiated during this phase. Considering the median and mean times in this table confirms that the instance router using the *LTAvgR* algorithm made *the right decision in all cases.*

For an in-depth view of the reward estimates (the average reward observed by *LTAvgR*) and execution times, we depict in Figs. 14 and 15 how their values changed during the experiment P^1 vs. P^2. The effect that fast completion leads to positive rewards is clearly visible in Fig. 14: shortly after the start of the experiment, the reward estimates for both versions jump to more than 0.6. After some fluctuation, P^1 is preferred approximately from request 280 to request 811. This is also visible in Fig. 13, where we can observe that the change in maximum of the two reward estimates leads to change in the request distribution strategy.

Figure 15 shows the PPIs observed by the instance router in order. Better PPIs, which lead to better rewards, are received early. However, worse PPIs tend to accumulate near the end of the warm-up phase. At request 811, the two estimated rewards are very close to each other – see Fig. 14. At this point, P^2 collects actual rewards from longer durations than P^1 – see Fig. 15. These longer durations result in negative rewards, which cause the reward estimate of P^2 to fall below that of P^1. This development leads to the change in the decision of *LTAvgR*.

5. Discussion

In the following, we discuss relative strengths and limitations of the techniques for simulation and AB testing proposed in this paper, as well as the evaluation. We also highlight opportunities for future work.

5.1. Trace Simulation

Our simulation technique provides a high level idea of what might have happened if the new version was executed instead of the old version under the same circumstances with the same resources. If new versions are created in

response to expected changes in circumstances, we need to validate that these new versions have a positive impact in the changed circumstances. In such cases, our simulation technique is not appropriate. Our simulation approach only addresses model-level changes in the new version. It cannot provide good insights if the decision logic and activity implementations in the new version are changed. Simulations can be successfully carried out using the partial matching mechanism if the estimates for new activities are provided. However, the results will be accurate only if there is a high degree of similarity between the types of traces generated by two versions. Our approach does not execute activities, so the actual differences in traces may only be observed after deployment. A simulation approach that is aware of the data, the decision logic, and the types of changes would be required for better results. Understanding these factors and automatically assessing the quality of results from simulation are out of the scope of this work.

Our simulation runs in the BPMS where the process engine knows what activities can be executed next, but does not know which one exactly and for low long. To guide the simulation, we need to know which activity can be executed next, and which path in the BPMN model should be taken to execute that activity. Candidate activities can be found by analysing the structure of the process model, and they can be used to query the TST. However, to execute the next activity suggested by the TST, the relevant outgoing flows of gateways in the BPMN model must be activated. We accomplish this by setting up the outgoing flows with condition expressions that activate the relevant outgoing flow when a custom variable is set to the name of the next activity. Deriving these condition expressions is non-trivial in complicated processes with nested loops and gateways. In our experiments, these conditions have been set manually. Further research is required on automatically generating such conditions.

5.2. AB Testing

The design of our routing algorithm, *LTAvgR*, was informed by practical observations of the limitations when applying existing algorithms in the process execution context. As we have demonstrated, our approach addresses the key requirements R1-R3. Our evaluation focused on the practical use of AB-BPM and *LTAvgR*; theoretical analyses of the routing algorithm were out of the scope. The in-depth analysis above showed how *business-relevant PPI observations have a direct influence* on the routing decisions taken by *LTAvgR*.

We have used a multi-armed bandit algorithm with rewards derived from a single PPI. In practice, however, multiple PPIs may need to be considered. Furthermore, optimizing routing for one PPI can negatively affect other PPIs. One key challenge in using multiple PPIs is that the reward delay for each PPI can be different. Dealing with such scenarios may require improved collection and reward update mechanisms, which we plan to explore in future work.

One limitation of our evaluation of AB-BPM so far is that they are based on isolated environments with no real user interactions. Factors like effects from the novelty of a process version were not considered. For example, in changing the user interfaces and forms, we may observe that users behave differently

when exposed to a new version. This issue may raise the question: where does the change in version still pertain to the process, and where is it about user interfaces? As in the case study using real-world building permit logs, we expect to find some patterns unique to business processes when these factors are accounted for. We believe that observations from real-world systems can guide us towards designing a better instance routing algorithm, and identifying best practices for performing AB tests on process versions.

In our case studies, the only PPI available to us was instance duration; average duration was assumed to be a good indicator of quality. If another PPI (e.g. profit margin) is available at any time during execution, the instance router can use that instead as a reward. Such PPIs may show multi-modal distribution that cannot be summarized using the mean, as we have done in the LTAvgR algorithm. We plan to investigate real-world cases with such performance characteristics and devise solutions for them in future work.

5.3. Approach Comparison

In addition to the requirements outlined earlier, the three most relevant dimensions for comparing the approaches are time, risk, and accuracy. Time refers to the investment of time, but also the delay until the result is obtained. Risk refers to exposing the user to a sub-optimal (potentially bad) version of a process. Accuracy refers to the quality of the PPI results obtained. A qualitative comparison of the two proposed approaches and the traditional evolution of process versions through iterations of the lifecycle is shown in Fig. 16.

We argue that simulation poses no risk (no exposure of users) and takes very little time (relative to the duration of long-running processes). However, its accuracy is variable, depending on the magnitude of differences between the versions, and at this point not overly reliable. As such, we argued throughout the paper that it should be employed primarily as a sanity check, and its results need to be interpreted in the light of this low reliability. Traditional process evolution in contrast is slow, risky (because all users are temporarily exposed to a new version), and not perfectly accurate (due to the time bias it introduces, cf. Section 2.1). AB testing on the other hand, achieves high accuracy, but also takes time and incurs some risk. This risk is, to a certain degree, mitigated by using a dynamic instance routing algorithm, which switches to the better version as soon as the observations allow for it and the experimentation phases out.

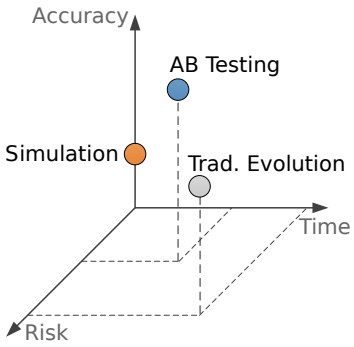


Figure 16: Qualitative comparison of process improvement approaches (positions on all three axes to be understood as indicative only).

Though risk is managed in two incremental steps, there is a sizeable increase of risk of exposure when switching from simulation to the experimentation phase of AB tests. In future work, we plan to explore a middle ground between

simulation and AB testing with the aim of reducing this risk while obtaining accurate performance estimates.

6. Conclusion

Business process improvement ideas do not necessarily manifest in actual improvements. In this paper we proposed the AB-BPM methodology and framework which can rapidly validate process improvement efforts, ensure fair comparison, and make process level adjustments in production environments. The methodology is supported by the framework, which includes our simulation and AB testing techniques, and the *LTAvgR* algorithm.

Our simulation technique extracts metrics and decision probabilities from the event log of an old process version and uses them to generate traces of a new version. Our AB testing approach uses an instance router that dynamically selects process versions based on their historical performance in terms of the chosen PPI. To this end, we proposed the *LTAvgR* algorithm and a reward design that can cater for the specifics of business process execution.

We evaluated our approach and methodology exhaustively, using synthetic and real-world data. Further, we analysed the framework's performance on real-world process versions by performing simulation and pair-wise AB tests on them. The evaluation results showed that our simulation gives fuzzy but informative results if a new version is simulated. It also showed that the simulation yielded accurate results if a process model is simulated using its own historical data – thus fulfilling a base assumption. Finally, our instance router with the *LTAvgR* algorithm dynamically adjusted request distribution to favour the better performing versions.

In future work, we aim to integrate our framework with approaches to balance multiple PPIs. We plan to consider user interaction, and run industrial case studies where we apply our instance router to actual production systems. In addition, we plan to explore solutions that provide a middle ground between simulation and AB tests.

Acknowledgements. The work of Claudio Di Ciccio has received funding from the EU H2020 programme under the MSCA-RISE agreement 645751 (RISE_BPM). We thank CASA for providing the information used in Section 4.1.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Springer, 2013. doi:10.1007/978-3-642-33143-5.

[2] C. W. Holland, 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.

[3] R. Kohavi, R. Longbotham, D. Sommerfield, R. M. Henne, Controlled experiments on the web: survey and practical guide, Data Min. Knowl. Discov. 18 (1) (2009) 140–181. doi:10.1007/s10618-008-0114-1.

[4] T. Crook, B. Frasca, R. Kohavi, R. Longbotham, Seven pitfalls to avoid when running controlled experiments on the web, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 1105–1114. doi:10.1145/1557019.1557139.

[5] S. Satyal, I. Weber, H. Paik, C. Di Ciccio, J. Mendling, AB-BPM: performance-driven instance routing for business process improvement, in: BPM, 2017, pp. 113–129. doi:10.1007/978-3-319-65000-5_7.

[6] H. A. Reijers, S. L. Mansar, Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics, Omega 33 (4) (2005) 283–306.

[7] N. Melão, M. Pidd, A conceptual framework for understanding business processes and business process modelling, Inf. Syst. J. 10 (2) (2000) 105–130. doi:10.1046/j.1365-2575.2000.00075.x.
URL https://doi.org/10.1046/j.1365-2575.2000.00075.x

[8] M. Hammer, J. Champy, Reengineering the Corporation : A Manifesto for Business Revolution, HarperCollins, 1993.

[9] T. H. Davenport, Process innovation: reengineering work through information technology, Harvard Business Press, 1993.

[10] V. Grover, S. R. Jeong, W. J. Kettinger, J. T. Teng, The implementation of business process reengineering, Journal of Management Information Systems 12 (1) (1995) 109–144.

[11] W. J. Kettinger, J. T. C. Teng, S. Guha, Business process change: A study of methodologies, techniques, and tools, MIS Quarterly 21 (1) (1997) 55–98.

[12] S. Poelmans, H. A. Reijers, J. Recker, Investigating the success of operational business process management systems, Inf. Tech. Mgmt. 14 (4) (2013) 295–314. doi:10.1007/s10799-013-0167-8.

[13] S. Alter, Work system theory: Overview of core concepts, extensions, and challenges for the future, J. AIS 14 (2) (2013) 1.

[14] A. Delgado, B. Weber, F. Ruiz, I. G. R. de Guzmán, M. Piattini, An integrated approach based on execution measures for the continuous improvement of business processes realized by services, Information & Software Technology 56 (2) (2014) 134–162. doi:10.1016/j.infsof.2013.08.003.
URL http://dx.doi.org/10.1016/j.infsof.2013.08.003

[15] D. A. Garvin, Managing quality: The strategic and competitive edge, Simon and Schuster, 1988.

[16] W. Jiang, T. Au, K.-L. Tsui, A statistical process control approach to business activity monitoring, IIE Transactions 39 (3) (2007) 235–249. arXiv:http://dx.doi.org/10.1080/07408170600743912, doi:10.1080/07408170600743912.

[17] T. Ōno, Toyota production system: beyond large-scale production, Productivity press, 1988.

[18] R. Andersson, H. Eriksson, H. Torstensson, Similarities and differences between tqm, six sigma and lean, The TQM magazine 18 (3) (2006) 282–296.

[19] F. Gregory, Cause, effect, efficiency and soft systems models, J. Operational Research Society (1993) 333–344.

[20] J. Worley, T. Doolen, The role of communication and management support in a lean manufacturing implementation, Management Decision 44 (2) (2006) 228–245.

[21] L. Bass, I. Weber, L. Zhu, DevOps - A Software Architect's Perspective, SEI series in software engineering, Addison-Wesley, 2015.

[22] R. Kohavi, T. Crook, R. Longbotham, B. Frasca, R. Henne, J. L. Ferres, T. Melamed, Online experimentation at Microsoft, in: Workshop on Data Mining Case Studies, 2009.

[23] M. Bozkurt, M. Harman, Y. Hassoun, Testing and verification in service-oriented architecture: a survey, Software Testing, Verification and Reliability 23 (4) (2013) 261–313. doi:10.1002/stvr.1470.
URL http://dx.doi.org/10.1002/stvr.1470

[24] D. Qiu, B. Li, S. Ji, H. Leung, Regression testing of web service: A systematic mapping study, ACM Comput. Surv. 47 (2) (2014) 21:1–21:46. doi:10.1145/2631685.
URL http://doi.acm.org/10.1145/2631685

[25] A. Burattin, PLG2: multiperspective processes randomization and simulation for online and offline settings, CoRR abs/1506.08415.
URL http://arxiv.org/abs/1506.08415

[26] M. Weidlich, H. Ziekow, A. Gal, J. Mendling, M. Weske, Optimizing event pattern matching using business process models, IEEE Trans. Knowl. Data Eng. 26 (11) (2014) 2759–2773. doi:10.1109/TKDE.2014.2302306.

[27] M. Weidlich, H. Ziekow, J. Mendling, O. Günther, M. Weske, N. Desai, Event-based monitoring of process execution violations, in: BPM, 2011, pp. 182–198. doi:10.1007/978-3-642-23059-2_16.

[28] C. Cabanillas, C. Di Ciccio, J. Mendling, A. Baumgrass, Predictive task monitoring for business processes, in: BPM, 2014, pp. 424–432. doi:10.1007/978-3-319-10172-9_31.

[29] D. Breuker, M. Matzner, P. Delfmann, J. Becker, Comprehensible predictive models for business processes., MIS Quarterly 40 (4).

[30] A. del-Río-Ortega, F. García, M. Resinas, E. Weber, F. Ruiz, A. R. Cortés, Enriching decision making with data-based thresholds of process-related kpis, in: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, Vol. 10253 of Lecture Notes in Computer Science, Springer, 2017, pp. 193–209. doi:10.1007/978-3-319-59536-8_13.
URL https://doi.org/10.1007/978-3-319-59536-8_13

[31] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, G. Tamburrelli, Dynamic qos management and optimization in service-based systems, IEEE Trans. Software Eng. 37 (3) (2011) 387–409. doi:10.1109/TSE.2010.92.
URL https://doi.org/10.1109/TSE.2010.92

[32] W. M. P. van der Aalst, Business process simulation survival guide, in: J. vom Brocke, M. Rosemann (Eds.), Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed., International Handbooks on Information Systems, Springer, 2015, pp. 337–370. doi:10.1007/978-3-642-45100-3_15.
URL https://doi.org/10.1007/978-3-642-45100-3

[33] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, C. J. Fidge, Workflow simulation for operational decision support, Data Knowl. Eng. 68 (9) (2009) 834–850. doi:10.1016/j.datak.2009.02.014.
URL http://dx.doi.org/10.1016/j.datak.2009.02.014

[34] A. Rogge-Solti, M. Weske, Prediction of business process durations using non-markovian stochastic petri nets, Inf. Syst. 54 (2015) 1–14. doi:10.1016/j.is.2015.04.004.

[35] I. Weber, J. Mendling, A vision of experimental process improvement, in: SIMPDA'15: International Symposium on Data-driven Process Discovery and Analysis, Vienna, Austria, 2015, pp. 127–130.

[36] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016. doi:10.1007/978-3-662-49851-4.

[37] F. Luccio, A. Mesa Enriquez, P. Olivares Rieumont, L. Pagli, Exact rooted subtree matching in sublinear time, Tech. Rep. TR-01-14, Università di Pisa (2001).

[38] N. R. T. P. van Beest, M. Dumas, L. García-Bañuelos, M. L. Rosa, Log delta analysis: Interpretable differencing of business process event logs, in: H. R. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), Business Process Management - 13th International Conference, BPM 2015,

Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, Vol. 9253 of Lecture Notes in Computer Science, Springer, 2015, pp. 386–405. doi:10.1007/978-3-319-23063-4_26.
URL https://doi.org/10.1007/978-3-319-23063-4_26

[39] N. van Beest, I. Weber, Behavioural classification for business process execution at runtime, in: PRAISE'16: International Workshop on Runtime Analysis of Process-Aware Information Systems at BPM'16, 2016.

[40] G. Burtini, J. Loeppky, R. Lawrence, A survey of online experiment design with the stochastic multi-armed bandit, CoRR abs/1510.00757.

[41] S. Agrawal, N. Goyal, Thompson sampling for contextual bandits with linear payoffs, in: Intl. Conf. Machine Learning, ICML, 2013, pp. 127–135.

[42] L. Li, W. Chu, J. Langford, R. E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Intl. Conf. World Wide Web, 2010, pp. 661–670. doi:10.1145/1772690.1772758.

[43] W. R. Thompson, On the likelihood that one unknown probability exceeds another in view of the evidence of two samples, Biometrika 25 (3/4) (1933) 285–294.

[44] O. Chapelle, L. Li, An empirical evaluation of thompson sampling, in: Neural Information Processing Systems (NIPS), 2011, pp. 2249–2257.

[45] R. S. Sutton, A. G. Barto, Introduction to Reinforcement Learning, 1st Edition, MIT Press, Cambridge, MA, USA, 1998.

[46] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, The annals of mathematical statistics (1947) 50–60.

[47] A. Rodrigues, C. Almeida, D. Saraiva, F. Moreira, G. Spyrides, G. Varela, G. Krieger, I. Peres, L. Dantas, M. Lana, O. Alves, R. Frana, R. Neira, S. Gonzalez, W. Fernandes, S. Barbosa, M. Poggi, , H. Lopes, Stairway to value: Mining the loan application process, BPI Challenge Report (2017).

[48] I. Teinemaa, A. Leontjeva, K.-O. Masing, BPIC 2015: Diagnostics of building permit application process in dutch municipalities, BPI Challenge Report (2015).