User Journey Games: Automating User-Centric Analysis

Paul Kobialka[1*], S. Lizeth Tapia Tarifa[1], Gunnar R. Bergersen[1,2] and Einar Broch Johnsen[1]

[1]Department of Informatics, University of Oslo, PO Box 1080, Oslo, NO-0316, Norway.
[2] GrepS B.V., Utrecht, the Netherlands.

*Corresponding author(s). E-mail(s): paulkob@ifi.uio.no;
Contributing authors: sltarifa@ifi.uio.no; gunnab@ifi.uio.no; einarj@ifi.uio.no;

Abstract

The servitization of business is moving industry to business models driven by customer demand. Customer satisfaction is connected with financial rewards, forcing companies to invest in their users' experience. User journeys describe how users maneuver through a service. Today, user journeys are typically modeled graphically, and lack formalization and analysis support. This paper proposes a formalization of user journeys as weighted games between the user and the service provider and a systematic data-driven method to derive these user journey games from system logs, using process mining techniques. As the derived games may contain cycles, we define an algorithm to transform user journeys games with cycles into acyclic weighted games, which can be model checked using Uppaal Stratego to uncover potential challenges in a company's interactions with its users and derive company strategies to guide users through their journeys. Finally, we propose a user-journey sliding-window analysis to detect changes in the user journey over time by model checking a sequence of generated games. Our analysis pipeline has been evaluated on an industrial case study; it revealed design challenges within the studied service and could be used to derive actionable recommendations for improvement.

Keywords: User journeys, Data-driven model construction, Time-series analysis, Games, Model checking, UPPAAL.

1 Introduction

Consider a company, Amend ltd., that offers document reviewing services: Users submit documents and meta-data to receive a professional review. To ensure user satisfaction, the company commissions a report on the users' experience with the offered service. A team of analysts conducts interviews with selected users to manually create user journey maps that reveal unknown pain points in the service, which could cost the company users. When expanding and improving its services, Amend wants to integrate continuous user feedback, and wonders if the team could use system logs to avoid the cumbersome manual questionnaires and scale the feedback to all users. However, their current *performance dashboards* [23] only display recent server statistics without incorporating user-centric analysis. Methods and tools to analyze user experience based on such large-scale collected logs are currently lacking [28]. In this paper, we present a method to automatically analyze logs from the users' perspective, based on weighted automata [18].

The scenario described above stems from the *servitization of business* [50], a concept of creating added value to products by offering services.

Servitization of business is a major practice embraced by most (if not all) successful companies today. Companies are interested in the analysis of their services, which traditionally focus on the *managerial* perspective, where the service is analyzed with respect to the companies' view. Recent trends shift the focus from the company's to the end-users' view, where a positive experience and impression that a user has while engaging in the service, has shown to have a positive impact on the financial reward of a company [25]. Thus, companies aim to analyze and improve their services, based on their users' satisfaction.

User journeys (also called customer journeys) analyze services from the user perspective [42]: A user journey is inherently a goal-oriented process, because humans engage in a service with a goal in mind. The user moves through the journey by engaging in so-called touchpoints, which are either actions performed by the user or a communication event between the user and a service provider. We here assume that users only engage in one touchpoint of a service at a time.

Tools to analyze user journeys are currently lacking [28], which hinders their operational use. User-journey diagrams are usually generated by hand, and the user perspective is derived from interviews with experts and users, e.g. [27, 42]. This process has been highly successful, discovering points of failure in the studied services and, as a result, providing advice to companies on how to improve their services. However, this manual process is best suited for relatively small services and a restricted number of users, and a particular point in time. For services with thousands of users, journey diagrams need to be automatically generated and analyzed. In particular, in business processes that change often, the impact of these changes needs to be evaluated as quickly as possible. *Concept drift* detection quantifies changes in an underlying business process [16]. However, not all observed changes are easy to detect, since they might originate from events out of the company's reach, but are still perceived from the users' point of view. Thus, systematic analysis techniques are needed to evaluate user journeys, and detect and stop unwanted trends.

This paper formalizes user journeys as *weighted games* [17, 18] between users and a service provider, and proposed a method to derive such games from system logs. Our aim is to use these games to analyze services and suggest service improvements such that service providers always have a strategy to guide their users toward a desired goal. The aim of our work is to reduce the gap towards fulfilling the analysis needs of companies such as Amend, the company of the motivating scenario above. In short, our contributions are:

1. a formalization of user journeys as weighted games;
2. an analysis pipeline to automatically discover and model check weighted games from system logs;
3. a sliding-window analysis for user journeys that lifts the analysis of one weighted game per system log to a series of weighted games over windows of time in a system log.
4. an experimental investigation of the feasibility of our approach on two data sets from an industrial case study.

User journey games systematically capture the user perspective of services by means of so-called *gas*. The term is inspired by blockchain technology such as Ethereum, where gas refers to the cost necessary to perform a transaction on the network. In our work, the gas quantitatively reflects how moves in the user journey contribute to the users reaching their goal. Consequently, the moves in the derived games are weighted and accumulated into the gas of the journeys, which allows journeys to be analyzed and compared using model checkers such as Uppaal Stratego [22] or PRISM-games [19], and to give strategic recommendations to service providers.

This is an extended version of a paper that appeared at SEFM 2022 [35]. Compared to that paper, we have here expanded the discussion of user journey games, introduced a time-driven analysis method for user journeys (henceforth called a *sliding-window* analysis) which extends the analysis pipeline based on user journey games, and expanded our experimental evaluation to a significantly bigger data set. A sliding-window analysis uses a series of automatically generated user journey games, where each game is automatically derived from a time window in the system log. We lift the model checking analysis to the series of games to uncover trends and changes over time.

Outline. Section 2 discusses related work. Section 3 provides background on weighted games and the model checking suite Uppaal that we

State of the art	*Contribution*
User journey modeling [8, 15, 20, 26, 36, 41, 42]: Mostly manual, very limited tool support	Digital support through automated model generation and improvement recommendations
Data-driven process discovery [3, 10–13, 30, 46, 47]: Lack support for user perspective	Used to generate formal, user-centric model with multiple actors
Timed-arc Petri net analysis [14, 21]: Verification of medical processes	Building games from logs to model actual user behavior
Concept drift detection [5, 6, 16, 44]: Detect process changes at the event level	Quantifying changes in user journeys over time

Table 1: Contributions and related work.

use for analysis. The formal model for user journeys is introduced in Sections 4–6, model checked in Section 7, and extended to a sliding-window analysis in Section 8. Section 9 discusses the implementation, Section 10 evaluates our approach experimentally in terms of an industrial case study, Section 11 discusses our approach and Section 12 concludes the paper.

2 Related work

We discuss related work on the modeling of user journeys and on using data-driven techniques to discover user journeys, Table 1 summarizes related work and positions our contributions. We are not aware of prior work that uses automatic verification methods to analyze user journeys.

User journeys aim to improve service design by describing how users interact with services [24, 49]. Modeling notations for user journeys aim to support the so-called blueprinting [15], i.e., to create an anticipated model of a service. There are various notations to create diagrams for user journeys [8, 20, 26, 36, 41, 42]; these diagrams are mostly handmade and only limited digital support exists; for example, a semantic lifting into ontologies has been used to visualize fixed aspects of a model [36]: the data sent, the communication channels and devices used, etc. Berendes *et al.* propose in [8] the *high street journey modeling language* (HSJML) tailored to journeys in shopping streets. Razo-Zapata *et al.* propose the *VIVA* modeling language with focus on interactions [41]. In contrast, our work aims to use data-driven techniques [3] to automatically discover user journey diagrams and formal methods to automatically check properties of user journeys and derive recommendations for improving the service under analysis. The aim with our work is to automate the labour-intensive manual mapping that captures the user's interaction with a service, thereby enabling scalability of user-centric analysis of complex services with many users.

The *Customer Journey Modeling Language* (CJML) [27, 29] captures the end-users' point of view. CJML distinguishes planned and actual user journeys, which represent the journey as planned as part of the service design and as perceived by the user, respectively. Our work is part of a project [28] on tool support for data-driven user-journey modeling in CJML. Whereas previous work on CJML manually quantifies user experience collected through user feedback questionnaires, our work aims to capture the journeys as perceived by the user in a data-driven manner, based on system logs.

Data-driven techniques for process discovery allow us to discover user journeys. Harbich *et al.* [30] use mixtures of Markov models to derive user-journey maps. Bernard *et al.* [11, 13] study process mining [3] for user journeys, such as hierarchical clustering to explore large numbers of journeys [10] and process discovery techniques to generate user-journey maps at different levels of granularity [12]. Terragni and Hassani [46] apply process mining to user journey web logs to build process models, and improve the results by clustering journeys. This work has been integrated with a recommender system to suggest service actions that maximize key performance indicators [47], e.g., how often the product page is visited. David *et al.* present TAPPAAL [21], a tool for analyzing timed-arc Petri nets, realized through mappings to UPPAAL. Bertolini *et al.* used TAPAAL for the verification of medical processes [14]. They focus on the graphical notation language Little-JIL, leaving the human aspect for future work. In contrast, our work focuses on the user-centric perspective, using games to model actual user behavior [35]. In this paper, we propose and use a data-driven method to automatically construct formal models of user-centric journeys with multiple actors. Complementing the work presented

here, we have studied the scalability of the basic mining technique for user journey games [33] and the integration of the strategies derived from user journey games in an actor-based simulation framework for user journeys [34].

The above-mentioned work inherently assumes that the analyzed collection of journeys is generated from an unchanged process. Bose *et al.* define different types of process changes, so-called *concept drifts*, and propose their detection based on follows- and precedes-relations at the event level with hypothesis testing [16], recent developments on concept drift are surveyed by Sato *et al.* [44]. Banham *et al.* extend process models with periodically recorded numerical values to gain closer insights into exogenous influences on a process [5, 6]. In contrast, our proposed sliding-window analysis does not investigate changes at the event level, but quantifies changes in the user journey over time, abstracting from the business process, thereby enabling the service provider to quantify the impact of intented as well as unintended changes on the user journey.

3 Preliminaries

We briefly summarise the formal notations and tools that we build on for the proposed user journey pipeline to analyze a service.

A *transition system* [40] is a tuple $S = \langle \Gamma, A, E, s_0, T \rangle$ with a set Γ of states, a set A of actions (or labels), a transition relation $E \subseteq \Gamma \times A \times \Gamma$, an initial state $s_0 \in \Gamma$ and a set $T \subseteq \Gamma$ of final states. A *weighted transition system* [48] $\mathcal{S} = \langle S, w \rangle$ extends the transition system S with a weight function $w : E \to \mathbb{R}$ that assigns weights to transitions.

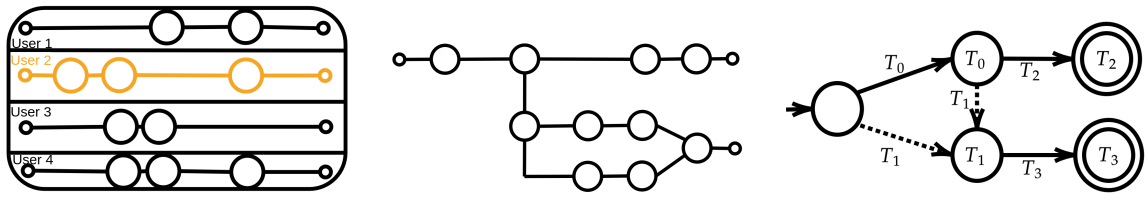
Weighted games [17] are obtained from weighted transition systems by partitioning the actions A into *controllable* actions A_c, and *uncontrollable* actions A_u, where only actions in A_c can be controlled by the analyzer, while actions in A_u are nondeterministically decided by an adversarial environment. When analyzing games, we look for a *strategy* that guarantees a desired outcome, i.e. winning the game by reaching a certain state. The strategy is given by a partial function $\Gamma \to Act_c \cup \{\lambda\}$ that decides on the action of the controller in a given state (here, λ denotes the "wait" action, letting the adversary move).

Uppaal Tiga [7] can be used to analyze reachability and safety properties for games expressed using (timed) transition systems, extending the model checker Uppaal [37]. Uppaal Tiga checks whether there is a strategy under which the behavior satisfies a control objective, denoted `control: P` for a property P. Property P is expressed in computational tree logic [4], an extension of propositional logic that is used to express properties along paths in a transition system. Recall that computational tree logic *state properties* ϕ can be decided in a single state; while *reachability properties* `E <>` ϕ express that the formula ϕ is satisfiable in some reachable state in a transition system; *safety properties* `E []` ϕ express that the formula ϕ is always satisfied in all the states of *some path* in a transition system and `A []` ϕ expresses that ϕ is always satisfied in all the states of *all paths* of a transition system. Similarly, *liveness properties* `A <>` ϕ express that the formula ϕ will eventually be satisfied in all the paths in a transition system and the formula ϕ `-->` ψ expresses that satisfying formula ϕ leads to satisfying formula ψ.

Uppaal Stratego [22] can be used to analyze and refine a strategy generated by Uppaal Tiga with respect to a quantitative attribute like weights. Uppaal Stratego is a statistical model checker [39]; it extends Uppaal for stochastic priced timed games and combines simulations with hypothesis testing until statistical evidence can be deduced.

4 From system logs to games

To capture the *user perspective* in games that model user journeys, user actions (representing communication initiated by the user) can be seen as controllable, and the service provider's actions as uncontrollable. However, from an analytical perspective, it is more interesting to treat user actions as uncontrollable and the service provider's actions as controllable. The service provider should have suitable reactions to all possible user interactions. Ideally, the service provider should not rely on the user to make the journey pleasant. Treating user actions as uncontrollable exposes the worst behavior of the service provider, and thereby strengthens the user-centric perspective promoted by journey diagrams. Games for user journeys are then defined as follows:



Log Directly Follows Graph Game

Figure 1: Creation of the Journey Model.

Definition 1 (User journey games) A *user journey game* is a weighted game $G = \langle \Gamma, A_c, A_u, E, s_0, T, T_s, w \rangle$, where

- Γ are states,
- A_c and A_u are disjoint sets of actions,
- $E \subseteq \Gamma \times A_c \cup A_u \times \Gamma$ are the transitions,
- $s_0 \in \Gamma$ is an initial state,
- $T \subseteq \Gamma$ are the final states,
- $T_s \subseteq T$ are the successful final states, and
- $w : E \to \mathbb{R}$ is the weight function.

In user journey games, the edges E model the touchpoints, A_c the actions initiated by the service provider, A_u the actions initiated by the user, and T_s the successful goal states.

The process of deriving such user journey games from system logs is illustrated in Figure 1. In a first step, we go from logs to a user-journey model, expressed as a directly follows graph (DFG), and in a second step, the DFG is extended to a game. The derivation of weights for the transitions is discussed in Section 5.

4.1 From system log to graph

We use a *directly follows graph* (DFG) as an underlying process model to capture the order of events in a system log; a DFG is well-suited as the process model provided that users only engage in one touchpoint at a time. DFGs are derived from system logs by means of *process discovery* [3]. A *system log* L is a multi-set of journeys. A journey $J = \langle a_0, \ldots, a_n \rangle$ is a finite and ordered sequence of events a_i from a universe $\mathscr{A}$.

We construct the DFG of a system log L as a transition system $S_L = \langle \Gamma, A, E, s_0, T \rangle$ where the states Γ capture the event universe, $\Gamma \subseteq \mathscr{A} \cup \{s_0\} \cup T$. Every sequence of events is altered to start in the start state s_0 and to end in a final state $t \in T$. Without loss of generality, we can assume that T only contains the states `finPos` and `finNeg`, marking the *successful* and *unsuccessful* completion of a journey, respectively (i.e., $T_s = \{\texttt{finPos}\}$ and $T \setminus T_s = \{\texttt{finNeg}\}$). The set of actions A is the union of the event universe and the final states, $A = \mathscr{A} \cup T$. The transition relation E includes a triple (a_i, a_{i+1}, a_{i+1}) if a_i is directly followed by a_{i+1} in some $J \in L$; we can traverse from state a_i to state a_{i+1} by performing the action a_{i+1}. Here reaching a state in S_L is interpreted as the corresponding event in L already having been performed. Note that events in logs represent user journey activities as states (e.g., as depicted in Figure 1), while the DFG and game represent activities as transitions, and *completed* activities as states. By construction, the DFG S_L obtained from log L can replay every observed journey in L. However, S_L may capture more journeys than those present in L; for example, S_L may contain transitions with loops.

There is a trade-off in the mining process between the precision and the generalization of the transition system with respect to the log [2, 3]: The precision can be increased by including a part of the event history in every state. A *h-sequence refinement* considers the last h events as one state. The size of the sequence refinement is captured in the superscript of the transition system; i.e., S_L^h denotes the transition system obtained from log L under an h-sequence refinement. We omit the superscript for history 1, thus S_L is a DFG. Assume a log $L_0 = \{J\}$ with one user journey $J = \langle a, b, a \rangle$, the 2-sequence states of J are $\{\langle a \rangle, \langle a, b \rangle, \langle b, a \rangle\}$.

The construction of the h-sequence refinement transition system S_L^h from refined states is similar to the construction of S_L above (which uses the original states). The activity of a transition is the

last event in the targeted state's history (so transitions keep their unique action). Observe how $S^2_{L_0}$ resolves the loop in S_{L_0} by including the histories. However, not all loops can be removed using the h-sequence refinement.

4.2 From graph to game

The transition system S^h_L is now transformed into a user journey game G^h_L. Observe that the transition system captures the temporal ordering of events but it does not directly differentiate the messages sent by the user to the service provider from those sent by the service provider to the user. For simplicity, let us assume that this information is either part of the events in the logs or known in advance from domain knowledge concerning the event universe. The mined transition system can then be extended into a game by annotating the actions that are *(un)controllable*.

5 Capturing user feedback in user journey games

We extend the games derived from system logs into weighted games by defining a *gas* function reflecting user feedback. The gas function will be automatically calculated and applied to the transitions of the game, depending on the traversal and entropy present in the system log. Informally, the gas function captures how much "steam" the consumer has left to continue the journey. With less steam, the user is more likely to abort the journey and with more steam, the user is more likely to complete the journey successfully. If the service provider attempts to provide the best possible service, its goal is to maximize gas in a journey. The adversarial user aims for the weaknesses in the journey and therefore minimizes the gas. Formally, the weight function $w : E \to \mathbb{R}$ maps the transitions E of a game to weights, represented as reals. Given a log L and its corresponding game, we compute the weight for every transition $e \in E$.

Since user journeys are inherently goal-oriented, we distinguish successful and unsuccessful journeys; the journeys that reach the goal are *successful* and the remaining journeys are *unsuccessful*. This is captured by a function majority : $E \times L \to \{-1, 1\}$ that maps every transition $e \in E$ to $\{-1, 1\}$, depending on whether the action in the transition appears in the majority of journeys in L that are unsuccessful or successful, respectively. Ties arbitrarily return -1 or 1.

Many actions might be part of both successful and unsuccessful journeys. For this reason, we use Shannon's notion of *entropy* [45]. Intuitively, if an action is always present in unsuccessful journeys and never in successful ones, there is certainty in this transition. The entropy is low, since we understand the context in which this transition occurs. In contrast, actions involved in both successful and unsuccessful journeys have high entropy. The entropy is calculated using

1. the number of occurrences of an event in the transitions of successful journeys within the system log L, denoted $\#^{pos}_L e$, and the number of transitions in unsuccessful ones, denoted $\#^{neg}_L e$; and
2. the total number of occurrences of the event in L, denoted $\#_L e$.

The entropy H of transition e given the system log L is now defined as

$$H(e, L) = -\frac{\#^{pos}_L e}{\#_L e} \cdot log_2(\frac{\#^{pos}_L e}{\#_L e}) - \frac{\#^{neg}_L e}{\#_L e} \cdot log_2(\frac{\#^{neg}_L e}{\#_L e}) \ .$$

The weight function w that computes the weights of the transitions can now be defined in terms of the entropy function, inspired by decision tree learning [43]. Given a system log L, the weight of a transition e is given by

$$w(e) = ((1 - H(e, L)) \cdot \text{majority}(e, L) - C) \cdot M \ .$$

The constant C represents an aversion bias and is learned from the training set. It is used to model a basic aversion against continuous interactions. The sign of a transition depends on its majority. If the transition is mostly traversed on successful journeys, it is positive. Otherwise, it is negative. The inverse entropy factor quantifies the uncertainty of transitions. The constant M scales the energy weight to integer sizes (our implementation currently requires integer values, see Section 9).

The gas quantitatively reflects the history of a journey, allowing us to not only compare the weights of transitions but also to compare (partial) journeys. The gas $\mathcal{G}$ of a journey $J = \langle a_0, \ldots, a_n \rangle$ with transitions $e_0, \ldots e_{n-1}$ is defined as the sum

of the weights along the traversed transitions:

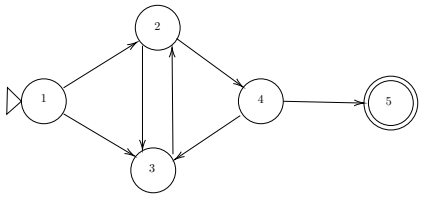
$$\mathcal{G}(J) \coloneqq \sum_{i=0}^{n-1} w(e_i) \ .$$

6 Finite unrolling of games

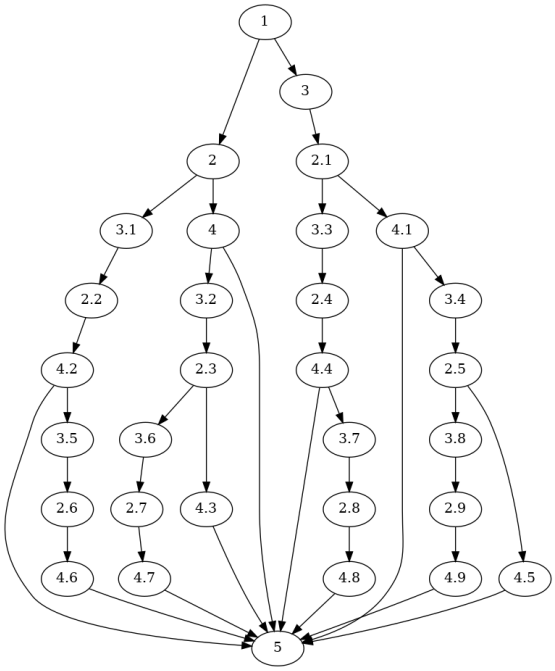
The generated weighted games may contain loops, which capture unrealistic journeys (since no user endures indefinitely in a service) and hinder model checking. Therefore, the weighted games with loops are transformed into acyclic weighted games using a breadth-first search loop unrolling strategy bounded in the number of iterations per loop. The transformation is implemented in an algorithm that preserves the original decision structure and adds no additional final states.

The algorithm for *k-bounded loop unrolling* (shown in Algorithm 1) returns an acyclic weighted game, where each loop is traversed at most k times. The unrolling algorithm utilizes a breadth-first search from the initial state s_0 in combination with loop counting to build an acyclic weighted game. In the algorithm, the state s denotes the current state that is being traversed. To traverse the paths in the weighted game, we use a queue Q to store the states that need to be traversed, a set C containing all the cycles in the graph (where each cycle is a sequence of states), and the function allSimplePaths(G, s, T) that returns all paths in the weighted game G from s to any final state $t \in T$. The extended graph is stored in the acyclic game G'. A state in a cycle can be traversed if it has been visited less than k times (see Lines 9–10). The function repetitions checks the number of traversals. If the counter for one cycle is k, the algorithm checks whether the cycle can be partially traversed (see Lines 11–16).

Partial traversals guarantee that we reach a final state without closing another loop. The partial traversal does not increase the count of another cycle to $k+1$ (Lines 14–16). Every state stores its history (a sequence of visited states), which can be retrieved using the function history. Line 14 increases the current history by including a (partial) path through the loop. This check iterates through all paths from the current state to any final state. If state t can be traversed, it is added to the acyclic game (Lines 17–20). A copy t' of t is added to the queue Q, the transition (s, t'), its weight and actor



(a) Cyclic Game



(b) Acyclic Game

Figure 2: Unrolling Example.

are added to G' using the function addTransition. If a final state is copied, the new set of final states is updated (i.e., $T' \leftarrow T' \cup \{t'\}$). The resulting weighted game can be reduced. All copies of states outside a cycle can be merged into the same state. This can either be done after unrolling the whole game or on the fly while unrolling.

6.1 Example

Figure 2 illustrates the unrolling algorithm (for simplicity, we ignore transition weights and do not distinguish controllable and uncontrollable actions in the example). Starting from the cyclic weighted game in Figure 2a, the algorithm with $k = 1$ generates the acyclic weighted game in Figure 2b. The input contains two loops: $C = \{\langle 2, 3\rangle, \langle 2, 4, 3\rangle\}$.

Algorithm 1 k-bounded loop unrolling

Input: Weighted Game $G = \langle \Gamma, A_c, A_u, E, s_0, T, T_s, w \rangle$, constant $k \in \mathbb{N}^+$
Output: Acyclic Weighted Game $G' = \langle \Gamma', A_c, A_u, E', s_0, T', T_s, w' \rangle$

Initialize $G' = \langle \emptyset, A_c, A_u, \emptyset, s_0, \emptyset, T_s, w \rangle$ and queue $Q = [s_0]$
$C \leftarrow \{c \mid c \text{ is simple cycle in } G\}$
while not EMPTY(Q) **do**
  state $s \leftarrow$ FIRST(Q)
  for $t \in \{t \mid (s,t) \in E\}$ **do**
    hist $\leftarrow$ PUSH(HISTORY(s), t)
    allSmaller $\leftarrow$ **True**
    canTraverse $\leftarrow$ **False**
    if REPETITIONS($c, hist$) $\geq k$ for all cycle $c \in C$ **then**
      allSmaller $\leftarrow$ **False**
    end if
    if !allSmaller **then**
      $P \leftarrow$ ALLSIMPLEPATHS(G, t, T)
      for path $p \in P$ **do** ▷ check whether cycle might be partially traversed
        $hist' \leftarrow$ MERGE($hist, p$)
        if REPETITIONS($c, hist'$) $\leq k$ for all cycle $c \in C$ **then**
          canTraverse $\leftarrow$ **True** ▷ cycle can be partially traversed
        end if
      end for
    end if
    if allSmaller $\vee$ canTraverse **then**
      state t' copy of t with history $hist$
      PUSH(Q, t')
      ADDTRANSITION($(s, t'), G'$) ▷ Copies weight to w' and actor to A'_c, A'_u
    end if
  end for
end while
return G'

Starting at state 1, we can traverse two neighbor states which both are part of the cycles. Thus, both transitions are inserted in G', and Q is updated to $\langle 2, 3 \rangle$. Continuing with state 2, all reachable transitions are again inserted as the corresponding cycles have not been fully traversed. Names of copies of the states that are already present once in the graph are incremented (the first occurrence of state 3 is called 3, the second 3.1, the third 3.2, etc.) The algorithm continues until the first loop $2, 3, 2$ is closed. In this case, it is not possible to traverse again to state 3 without closing the loop $\langle 2, 3 \rangle$. Only state 4 and its corresponding loop can be traversed (see Figure 2b, left branch). As result of the state reduction, all final states are merged into one (removing the copies originally introduced by the algorithm).

6.2 Properties

Algorithm 1 constructs an acyclic user journey game that preserves the decision structure of the initial weighted game. By construction, unrolled weighted games do not traverse cycles in the initial game more than k times. Loops can be traversed partially to ensure that every final state in the acyclic weighted game is also a final state in the initial weighted game. Only unreachable states are excluded in the acyclic game. No further final states or "dead ends" are introduced.

The following lemma expresses that Algorithm 1 constructs an acyclic user journey game that preserves the properties described above.

Lemma 1 *Let* $G = \langle \Gamma, A_c, A_u, E, s_0, T, T_s, w \rangle$ *be a user journey game,* $k \in \mathbb{N}^+$ *the loop unrolling*

constant, and G' the unrolled game returned by Algorithm 1 for inputs G and k. Then

1. *G' as acyclic.*
2. *No path in G' traverses a loop via original nodes in G more than k times .*
3. *Final states in G' are copies from states in T.*

Proof (sketch). Point 1 is ensured in the algorithm by only inserting edges to fresh copies of states and never to existing ones (Line 24).

Points 2 and 3 follow from the algorithm's criterion for inserting new states (Line 21): to add a state to the unrolled game, either no loop is traversed more than k times or there exists a path leading to a final state in T that does not close any loop for the $k+1$-th time. □

The algorithm also preserves the local decisions between controllable and uncontrollable actions, so the strategies found in the unrolled weighted game carry over to the original weighted game. Observe that a game and its corresponding unrolled game share final states only if the unrolled game is reduced by merging states that do not occur in loops. Otherwise, the unrolled game might contain various copies of the final states T.

7 Model checking user journeys

In this section we describe how to model check properties for user journeys and generate strategies to improve user journeys, using acyclic weighted games. The analysis of a weighted game gives formal insights into the performance of a service. We introduce generic properties that capture the user's point of view on a user journey. The analysis in this paper uses the Stratego extension for Uppaal [22], which supports non-deterministic priced games and stochastic model checking. Stratego allows to model check reachability properties within a finite number of steps, when following a strategy (therefore the need for acyclic games). Stratego constructs a strategy that satisfies a property P, so that the controller cannot be defeated by the non-deterministic environment. We detail some strategies and properties of interest for games derived from user journeys.

7.1 Guiding users to a target state

A company needs a suitable plan of (controllable) actions for all possible (uncontrollable) user actions when guiding users through a service. We define the following Uppaal Stratego strategy:

```
strategy goPos
  = control: A<> Journey.finPos .
```

Model checking this property returns true if and only if there exists a company-strategy `goPos` such that the positive target state `finPos`, indicating that the journey is successful, is eventually reached in all paths. The corresponding strategy (given as a pseudo-code) can be produced with the Uppaal Tiga command-line tool verifytga. If the verification fails, the company should be advised to simplify their service and offer more support to avoid unsuccessful user journeys.

7.2 Analyzing user feedback

We can use the gas function and a liveness property to analyze the desired accumulated feedback at the end of successful user journeys:

```
Journey.finPos --> gas > 0 under goPos .
```

This property checks that in general users have balancing experiences within their journeys, when the company follows the `goPos` strategy.

We can also check the feedback levels along the journey. The following property checks that a user never falls below a defined constant feedback `C`:

```
control: A[] gas > C under goPos .
```

Fluctuations in the feedback level of users can be revealed using simulations. Uppaal uses an implicit model for the passage of time to guarantee termination of statistical queries and simulations, using an upper time-bound `T`, as specified in [22]. The following query simulates `X` runs through the system using the `goPos` strategy, where each run has `T` as a time-bound:

```
simulate [t<=T; X]{Journey.finPos, gas}
  under goPos .
```

The time-bound is set to a value that guarantees all runs to reach a final state.

7.3 Analyzing user journey trajectories

Reaching a final state in a journey with a positive feedback does not ensure a satisfying journey. The user might still visit every pitfall along the way. To provide a satisfying journey, a company is among others interested in minimizing the expected number of steps. A strategy minimizing the number of steps can be defined as follows:

```
strategy goPosFast = minE(steps) [t<=T] :
    <> Journey.finPos under goPos .
```

This strategy can additionally be used to examine the expected lower bound of gas within a journey and the expected maximum value of accumulated gas at the end of a journey (denoted by `finalGas`):

```
E[t<=T; X] (min: gas) under goPosFast ,
E[t<=T; X] (max: finalGas) under goPosFast .
```

These values are computed with a time-bound of `T` and over `X` runs. We denote the results of the previous queries for a specified model by `minGas` and `maxFinalGas`.

User journey games, generated from logs, can be very detailed and complex. Therefore, we consider how we can reduce complexity and simplify the understanding of results, for example, during a validation process, the results may need to be put into context by domain experts. We can reduce complexity by grouping various states into phases; e.g., a *sign-up* phase may consist of the states in a user journey game that captures sign-up events in a service. We lift the analysis described above to phases, e.g., to find out in which phase we encounter `minGas` or `maxFinalGas`. This lifting from states to phases can be encoded in the model such that every state lies in exactly one phase. Under a h-history refinement, the encoding should capture that a state belongs to a phase if the last state refined in the h-sequence belongs to that phase. The encoding of phases in states allows us to check if, e.g., the minimum gas `Min` occurs in a specific phase P by means of the following query:

```
control: A<> gas <= Min && phase == P
    under goPos .
```

The value `Min` used in this query can be calculated in advance, e.g., by means of the `minGas` query.

8 Sliding-window analysis for journeys

The analyses discussed in the previous sections inherently assume that the system logs are generated from an unchanged process; i.e., the data is recorded under equivalent system settings. In a fast-paced business setting, changes and updates are constantly developed, integrated, and evaluated. Recorded data may no longer be representative of the current state of the system. Nevertheless, practitioners are interested in the effectiveness and impact of their changes, motivating the need for time-driven user-journey analysis. The recorded data cannot be interpreted as one, coherent data set but must be analyzed in correspondence with their temporal information. Journey records before and after system-level changes are not recorded from the same process: issues observed in earlier journeys might be resolved at later stages and other issues only appear in later journeys. When analyzing massive system logs collected over a long time, the impact of architectural changes might be buried. Mixing journeys from different data-generation processes skews the evaluation of changes and newly introduced features in a system.

We now introduce steps to leverage the previous analysis method into a time-driven analysis and consider points in time over the time domain $\mathbb{R}^+$. Hence, we define time points for every user journey, e.g. the start of the journey, and split the log into sub-logs containing only journeys in the same interval, e.g. all journeys starting within 10 days. For every sub-log, an individual user journey game is generated and stored in a sequence of games. Model checking the sequence of games individually returns a time series over the single model checking results, revealing the impact of changes in the user journey over time. Given a log L, let $\mathcal{I} \subset \mathbb{R}^+$ be a finite sequence of time-points $\mathcal{I} = \langle \mathrm{tp}_1, \ldots, \mathrm{tp}_n \rangle$ with $\mathrm{tp}_i \in \mathbb{R}^+$; e.g., $\mathcal{I}_{\mathrm{days}}$ contains time-points for every 24 hours between the first and last day of the timestamps present in L (mapped into the domain of $\mathbb{R}^+$). Given a constant window size $\mu \in \mathbb{R}^+$, we let $W_i = [\mathrm{tp}_i, \mathrm{tp}_i + \mu]$ denote a window that represents a time interval spanning from time-point $\mathrm{tp}_i \in \mathcal{I}$ to $\mathrm{tp}_i + \mu$.

Given an event a in a journey J, the time function $\delta(J, a)$ denotes the timestamp of a in J

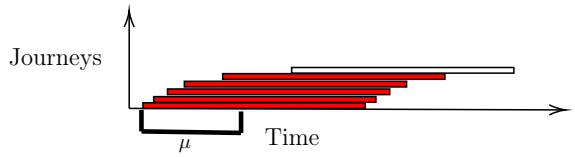


Figure 3: First window in sliding-window log.

(mapped into the domain of $\mathbb{R}^+$); e.g., the timestamp of the first or last event in a journey. The sliding-window analysis can be adapted to focus on different events by selecting other events as the second argument to δ; e.g., users finishing their journeys in the same window or users experiencing a particular event in the same window. We ignore the second argument to the function δ if we use the first event in a journey a_0 to select the timestamp of the initial event in a journey $J = \langle a_0, \ldots, a_n \rangle$. In the sequel, we focus on sliding-window analysis based on the initial event of the journeys, and therefore, we write $\delta(J)$ when the second argument is a_0.

The sub-log $L_i \subset L$, defined over the window W_i, includes all journeys in L that are contained in W_i, formally $L_i = \{J \in L \mid \delta(J) \in W_i\}$. Thus, a journey $J \in L$ is in L_i if $\delta(J)$ is in the window W_i. We define a *sliding-window* log $\mathbb{L}_\mu^{\mathcal{I}} = \langle L_1, \ldots, L_{|\mathcal{I}|} \rangle$ to be a sequence of sub-logs over L, where journeys are grouped based on the windows W_i ranging over time stamps $\mathcal{I}$ and window size μ.[1] Observe that sliding-window logs contain complete journeys; i.e. , journeys are not split by the window size if they are not completed within a window. Figure 3 shows (in red) the first sub-log L_1 of the sliding-window log W_1 grouped by journey start time (the timestamp of the first event), the top-most journey (in white) is not in L_1 as it starts outside the window size μ.

We analyze sliding-window logs to uncover changes over time from the user's perspective. For every log L_i in a sliding-window log $\mathbb{L}_\mu^{\mathcal{I}}$, we construct the corresponding user journey game; the resulting sequence of user journey games is denoted $\mathbb{G}^{\mathrm{h}}$, where h denotes the chosen h-sequence refinement. Each user journey game might contain loops and require unrolling to ensure that the analysis of queries terminates. Given an unrolling parameter k, we construct $\mathbb{G}^{\mathrm{h}}_{\mathrm{k}}$, which contains the sequence of k-bounded loop-unrolled user journey games from $\mathbb{G}^{\mathrm{h}}$.

The games of the sequence $\mathbb{G}^{\mathrm{h}}_{\mathrm{k}}$ are individually model checked, resulting in a time-series of strategies and statistics describing the user journeys. Queries might depend on each other; e.g., properties using the `goPos` strategy depend on its prior successful establishment. In case `goPos` cannot be established, all depending queries are mapped by default to a predefined constant. As multiple queries from a sequence of queries Q can be used on every user journey game $G^h_{L_i} \in \mathbb{G}^{\mathrm{h}}_{\mathrm{k}}$, the result is a series of vectors $Q_{\mathbb{G}^{\mathrm{h}}_{\mathrm{k}}}$ describing the analysis of the user journey games over time. Every query $q_i \in Q$ has a well-defined solution space S_i; e.g., `control` queries return strategies that map states to actions, expectation queries `minE(steps) [t<=T]` return values in $\mathbb{R}^+$ (expectation values with confidence intervals). Thus, each result vector $\nu \in Q_{\mathbb{G}^{\mathrm{h}}_{\mathrm{k}}}$ has $|Q|$ entries, one result for each query q_i. The space of ν is defined by the solution spaces S_i of the corresponding queries q_i, therefore $\nu : S_1 \times \cdots \times S_{|Q|}$. By comparing the analysis results in $Q_{\mathbb{G}^{\mathrm{h}}_{\mathrm{k}}}$, we can gain insights into the temporal changes occurring in the user journeys of a system.

9 Implementing the pipeline to analyze user journeys

This section describes the implementation of the analysis pipeline detailed in Sections 4–8. We focus on the implementation decisions made along the pipeline to facilitate the analysis. A source repository for our work on user journey games is available online [1].

The pipeline is implemented in Python. The input to the pipeline is a system log of a service provided by a company, and optionally a window size and time-points. The output is either a single Uppaal model or a sequence of Uppaal models (if the window size and time-points are given). Sequences of models are generated by repeatedly calling the single window construction for all sub-logs. The returned models can be model checked by either the proposed properties in Section 7 or by other custom-made properties using Uppaal Stratego.

[1] In the sequel, we assume that the time function selects the time of the first event of a journey and omit the time function δ in the sliding-window log notation $\mathbb{L}_\mu^{\mathcal{I}}$.

9.1 Pipeline implementation for single window logs

Here, we describe the implementation of the analysis pipeline for a single log. We mine the transition system from logs and then remove transitions that were rarely traversed, to simplify the graph and make it robust. Leemans *et al.* describe two ways to build a robust transition system [38]: One can (1) remove either transitions from the graph or (2) remove journeys from the log and rebuild the graph. For single window analysis, we use the approach (1) above, since removing journeys requires larger datasets. This modification ensures that the model only contains relevant journeys. Our implementation supports h-sequence refinements with transition removals.

We enrich the graph with knowledge indicating which actions are controllable and uncontrollable. Since companies want to understand why onboarded users reach their goal or quit in the middle of a journey, we add to the model two final states representing a positive endpoint, `finPos`, and a negative one, `finNeg`, respectively.

We generate a weighted transition system by computing a weight for each transition, as discussed in Section 5. The factor M scales the weights to integer sizes, required by Uppaal's model checker. However, given that we can simplify the transition system, the logs might contain journeys that are not re-playable in the graph. Computing the gas of such journeys corresponds to the *alignment* problem [31, 38]. The alignment procedure consists of either allowing additional steps in the log without counterparts in the model or allowing steps in the model without steps in the log. Since the simplification omits steps in the model, it was here sufficient to use the information given in the log, without inferring further model steps. Optimal alignments can also be used to compute the gas.

As a final step, we unroll the weighted game with cycles, as described in Section 6, to obtain an acyclic weighted game, which is the output of the transformation and the input to Uppaal for further analysis. Bounded constraints in the properties are introduced to the unrolled model to ensure termination.

9.2 Pipeline implementation for sliding-window logs

For the sliding-window analysis, we use time function $\delta(J)$, mapping journeys J to their start times, days between the first and the last journey as timepoints $\mathcal{I}_{\text{days}}$, and a fixed window size μ, resulting in a sliding-window log $\mathbb{L}_{\mu}^{\mathcal{I}_{\text{days}}}$, see Section 8. The number of users may vary highly between sub-logs L_i, and many journeys contain rarely traversed transitions. Thus, we use h-sequence refinements without removing transitions to guarantee connected models.

For every sub-log L_i in $\mathbb{L}_{\mu}^{\mathcal{I}_{\text{days}}}$ we generate a new user journey game with h-sequence refinement. Each of the resulting games is then k-times unrolled. The unrolled game $\mathbb{G}_{\text{k}}^{\text{h}}$ is then model checked with a sequence of Uppaal Stratego queries Q.

10 Experiments

In this section we evaluate the pipeline for user journey analysis from Section 9 experimentally. We aim to answer the following research questions:

- **RQ1:** Does the user journey game analysis reveal actionable insights for stakeholders?
- **RQ2:** Can user journey games be used over a time series of system logs to discover changes in the user journey?
- **RQ3:** How much does the experience of using the service differ from customer to customer?

The evaluation was done on an industrial case study from the company GrepS. We describe the context for the system logs provided by GrepS in Section 10.1, the experimental design and setup for our experiments in Section 10.2 and the results we obtained in Section 10.3. The results are discussed from the industrial perspective of GrepS in Section 10.4 and threats to validity are considered in Section 10.5.

10.1 Context

GrepS is a company that offers a research-based service [9] to analyze and measure programming skills for the Java programming language. Typical customers are organizations hiring or training software developers. The *users* of the service are

Timestamp	···	*Metadata*
5245944	···	Registered
5780525	···	Registered
6104714	···	Activated
6104714	···	Logged in: Web page

Figure 4: Extract of GrepS' system logs.

developers who receive a request from a *customer* organization to complete a skill analysis within a given time frame, typically 1–2 weeks.

The service consists of a sign-up phase followed by a phase in which users solve programming tasks in an authentic programming environment, including an instructional task and a practice task. The service then analyses the users' skills and asks them to share the skill report with the customer. In a *successful* use of the service, a user successfully completes three phases: (1) sign up, (2) solve all programming tasks, and (3) review and share the skill report. In an *unsuccessful* use of the service, the user permanently stops using the service or does not share the report with the customer.

GrepS provided two anonymized system logs for our experiments:

- $GL1$ is a small system log recording users' interactions with the system during 49 days (released in spring 2022, initially reported in [35]), and
- $GL2$ is a large system log spanning over more than two years of users' interactions with the system (released in December 2022).

$GL1$ is not contained in $GL2$, since they span different time frames. The logs were provided in the form of tabular data; only the fields *Timestamp*, which gives the order of events, and *Metadata*, containing meta-information on the kind of event, were used to generate the weighted games.

Ties between concurrently recorded events can be broken either arbitrarily, capturing different order of execution for these concurrent events in the model, or by inferring (implying) an order, e.g. the order events are stored in the event log file, thereby reducing the number of transitions in the model. In our experiments, we decided to break ties arbitrarily to capture different orders of execution and not assume additional expert knowledge for the model generation that could give insights about certain orders of events that are expected in the journey. An extract of the system log is shown in Figure 4. The full details of the data sets in the log are given in the accompanying artifact.[2]

10.2 Experimental Design and Setup

To answer our research questions, $GL1$ was used for the single-window analysis and $GL2$ for the sliding-window analysis, respectively.

RQ1. We analyzed the user journey game generated from $GL1$. The outcomes of the analysis are then discussed from the comapny's perspective in Section 10.4. The analyses of the user journey game include:

RQ1-A: Observations of the weighted game,
RQ1-B: Observations of the model checking of the properties, and
RQ1-C: Further recommendations for GrepS to improve their service, based on the analysis results.

The recommendations in RQ1-C form the basis for the discussion in Section 10.4.

RQ2. We analyzed the user journey games generated from $GL2$. We generated a sliding window log $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$ in which the window size for the sliding window analysis was set to 49 days, the length of the log $GL1$, and considered each day between the first and last journey as the time-points ($\mathcal{I}_{\text{days}}$). Comparing different window sizes reveals that the 2-sequence refinement is sufficient to reduce the number of loops while generating games with the least number of states, longer sequence refinements increase the number of states and did not contribute enough to the reduction of cycles. We unrolled the resulting games, returning the sequence of acyclic games $\mathbb{G}_1^2$, i.e. we construct games with a 2-sequence history and an unrolling parameter of 1.

We use all queries besides (2) and (3) from Figure 5 as query sequence Q. The vector-sequence $Q_{\mathbb{G}_1^2}$ contains the results from querying the game sequence $\mathbb{G}_1^2$, and contains $|\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}|$ vectors ν of size 8, whereby the first and fifth element in each ν contains the established strategies `goPos` and `goPosFast`. As the models are automatically

[2] An artifact for the implementation and evaluation of the single-window analysis pipeline, the sliding-window analysis pipeline, and the second system log is available: https://doi.org/10.5281/zenodo.10666884.

`strategy goPos = control: A<> Journey.finPos`	True
`Journey.finPos --> gas > 0 under goPos`	False
`control: A[] gas > -42 under goPos`	True
`E[t<=100; 500] (max: steps) under goPos`	27.5
`E[t<=100; 500] (min: gas) under goPos`	-26.3
`E[t<=100; 500] (max: finalGas) under goPos`	65
`strategy goPosFast = minE(steps) [t<=100] : <> Journey.finPos under goPos`	True
`E[t<=100; 500] (max: steps) under goPosFast`	20.9
`E[t<=100; 500] (min: gas) under goPosFast`	-18.9
`E[t<=100; 500] (max: finalGas) under goPosFast`	36

Figure 5: Analysis of the weighed game generated from $GL1$.

generated, they do not necessarily allow for a guaranteeing strategy `goPos`; for this reason, we choose default values not in the query's solution space: -1 for non-negative results, e.g. `maxFinalGas`, and 1 for non-positive results, e.g. `minGas`. The analysis consists of two parts:

- RQ2-A: Identify sub-logs for which no guaranteeing strategy `goPos` could be established, and
- RQ2-B: Identify general insights gained from model checking the game sequence $\mathbb{G}_1^2$.

RQ3. We analyzed the user journey games generated from $GL2$ by filtering per customer company the journeys contained in each sub-log in $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$; i.e., we grouped users applying to the same company and built sliding window logs for the three largest customers which commissioned 78% of the users in $GL2$. To compare the experiences of different customers with the overall experience, we repeat experiments conducted for RQ2 on the filtered event logs. We extend the analysis and investigate the service phases in which users experience their minimum of gas to uncover the customer-specific behavior of their users.

10.3 Results

RQ1. The observations in the generated user journey game (RQ1-A) for $GL1$ and results from simulations and model checking (RQ1-B) lead to actionable insights concluded in improvement recommendations for the service (RQ1-C).

RQ1-A. The generated cyclic user journey game for $GL1$, which still contains loops, is shown with events (or touchpoints) T and weighted transitions in Figure 6. We opt to remove transitions with less than four traversals to ignore rare transitions yet keep the graph connected. In the figure, the transition thickness indicates how often a transition was traversed and dashed lines represent uncontrollable transitions. Positive (negative) transitions are green (respectively, red).

The derived weights already allow us to make some interesting observations. The weighted game shows negative weights (about -1 to -2) through Phase 1 (T0–T5), up until the practice task has been completed (T12) in Phase 2 (T6–T20). After that, the weights are positive (about $+1$ to $+5$) and increase steadily for each new task. Phase 3 (T21–T26) also has positive weights through the user journey; here, a developer logs back into the web system after having completed all tasks (T19), waits for the report to be ready (T21), and finally approves the sharing of the report with GrepS' customer (T26).

Phase 1 shows two negative weights for some users that involve more touchpoints than what the planned journey entails: (1) T4 captures an error where a virtual computer does not spin up correctly thereby requiring the user to contact support; (2) there are cyclical negative weights between T6–T8 where a user starts receiving instructions for Phase 2, but stops and then returns to the system again at a later time. Phase 3 also has negative weights due to deviations from the planned journey, for example when the user does not login after the report is available (T24).

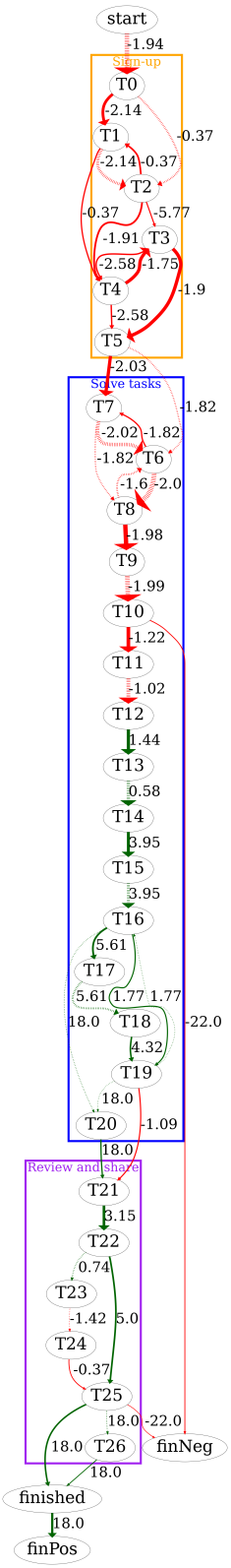


Figure 6: The weighted game built from $GL1$.

The figure also shows a strong negative weight (of -22) when a user does not submit the practice task in T11, resulting in a negative outcome, a transition to `finNeg`. Seen from a user perspective, Figure 7 shows the four touchpoints where most users stop using the service: 18% of all users quit after finishing the practice task (T10), which is twice that of users who stop after the first (T12, 9%) and second task (T14, 9%); 12% of the users do not want to share their report (T25). The blue line shows how many users remain using the service in percent after each of the four touchpoints.

RQ1-B. The accumulated feedback along the paths of the journey supports the observations on unsuccessful journeys (RQ1-A). Figure 8 shows 10 simulations with the `goPos` and `goPosFast` strategies; the lines show the amount of gas (accumulated feedback) along the journey. We here used $k = 1$ for the unrolling. For all simulations, the gas has an initial dip with a steep increase afterwards. The results in Figure 5 summarize the model checking and support the observations for RQ1-A. Observe that the `goPos` strategy cannot prevent the gas from falling below 0; in fact, it can fall as low as -42 along the journey with an expected minimum of -26.3.

Depending on the application context, multiple factors can contribute to an optimized journey. The strategy `goPosFast` was introduced in Section 7 as a refinement of `goPos`. It searches for an optimal strategy towards a successful final state, while minimizing the expected number of steps. The lower part of Figure 5 evaluates the queries under `goPosFast`. The simulations of the refined strategy, in Figure 8, show a smaller dip than with the `goPos` strategy. It improves the expected minimum feedback by 7.4 units and reduces the expected length of the journey by 6.6 steps. The expected maximum final feedback is also reduced from 65 to 36.

RQ1-C. From the company's perspective, several key takeaways have been identified from the weighted game, the simulations, and the model checking of properties:

- The instructional task and practice tasks during Phase 2 should be integrated into a single task that is more motivating for the user to complete.

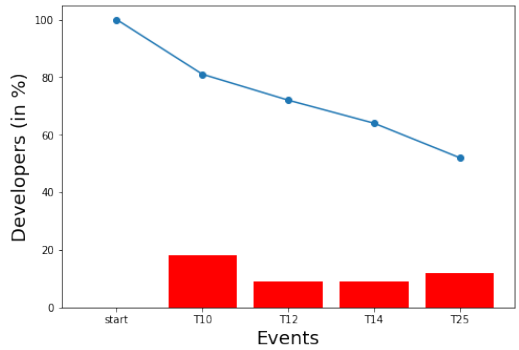


Figure 7: Events in unsuccessful journeys of $GL1$.

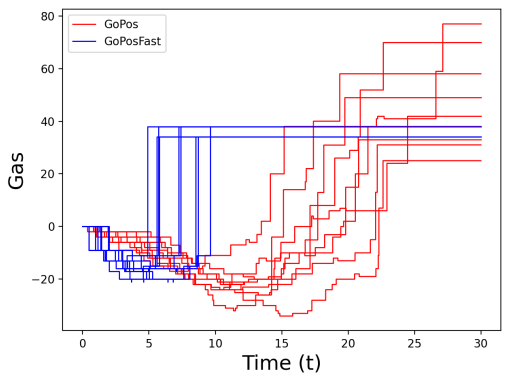


Figure 8: UPPAAL simulations.

- Users that disconnect from the service for several days after having progressed to the instructional, practice, or first task should be prompted to continue by, e.g., automatically sending a motivational email.
- The sign-up process should be simplified if possible.

RQ2. We generated a sliding window analysis for $GL2$. We answer the second research question in two steps: RQ2-A analyzes the automatic construction of the time series which requires guaranteeing strategies, and RQ2-B groups the resulting time series into regions of interest.

RQ2-A. For a positive user experience, the company ought to guide the users to a positive endpoint, supporting them in achieving their journey's goal. We test if a guaranteeing strategy exists in every sub-log, with a focus on the sub-logs in which GrepS cannot guide users to a positive outcome, i.e. `goPos` does not exist.

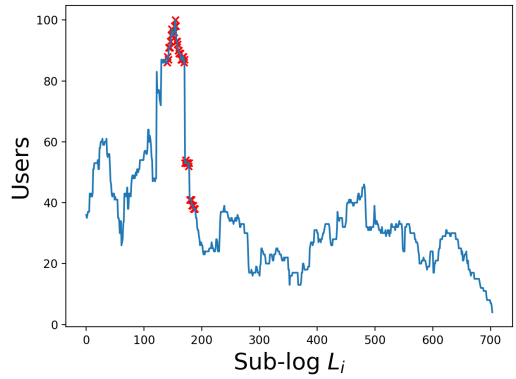


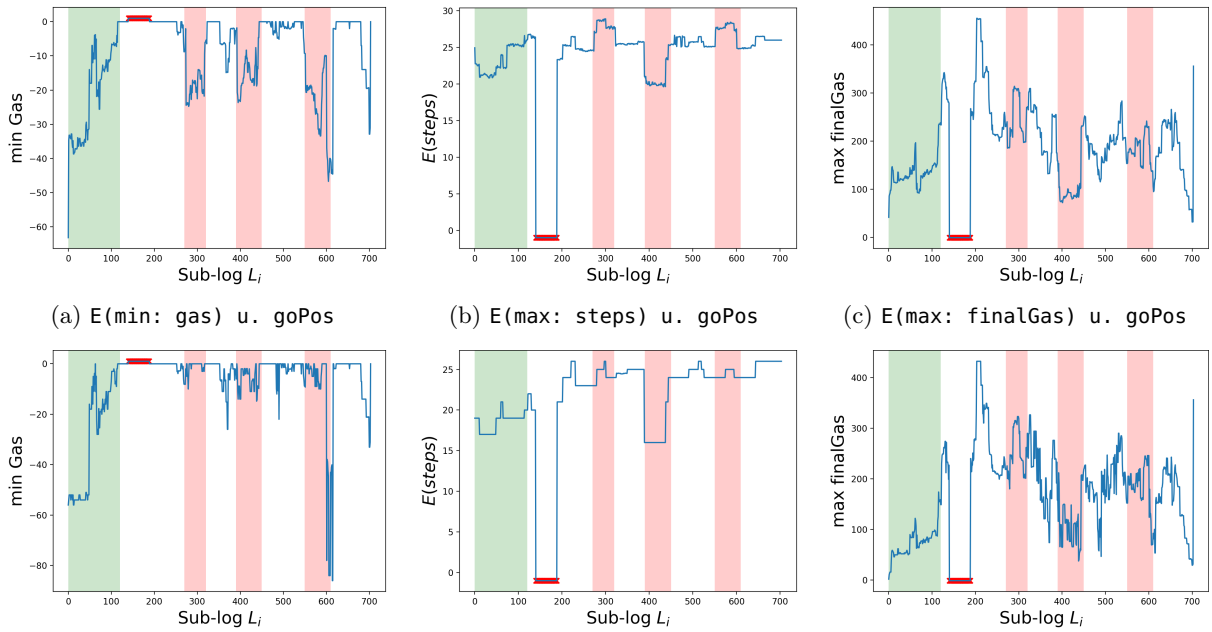
Figure 9: Development of users and `goPos` in sliding-window log $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$.

Figure 9 shows the number of users in each sub-log L_i for the sliding-window log $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$. Here, sub-log L_i starts i days after the first journey in $GL2$. The sub-logs in which `goPos` cannot be established are marked with red crosses. GrepS can establish a guaranteeing strategy for all sub-logs except for sub-logs 140–189, which coincides with the observed global peak of users. Further analysis of the game sequence reveals that the previously discussed technical error re-occurs after the completion of the first task (see RQ1-B). This behavior is also observable at a later time in which it does not hinder the construction of `goPos`.

Requiring a guaranteeing strategy reveals flaws in the user journey: encountering technical errors prevents users from progressing and favors unsuccessful journeys.

RQ2-B. Sub-logs that allow for a guaranteeing strategy `goPos` are analyzed on their expected number of steps throughout the journey, the minimum gas within the execution, `minGas`, and their maximum final gas, `maxFinalGas`; each under `goPos` and the refined strategy `goPosFast` (minimizing the number of required steps). The model checking results for $Q_{\mathbb{G}_1^2}$ are presented in Fig. 10 (results from strategies are not captured). Sub-logs without a guaranteeing strategy are marked with red crosses. Results under `goPos` are depicted in the first row (see Figs. 10a–10c), and results under `goPosFast` are depicted in the second row (see Figs. 10d–10f).

The average number of steps under `goPos` simulations is around 25, and under `goPosFast` it is reduced by around 5 steps. These results



(a) `E(min: gas) u. goPos` (b) `E(max: steps) u. goPos` (c) `E(max: finalGas) u. goPos`

(d) `E(min: gas) u. goPosFast` (e) `E(max: steps) u. goPosFast` (f) `E(max: finalGas) u. goPosFast`

Figure 10: Model checking results $Q_{\mathbb{G}_1^2}$, improvements are highlighted with a green background span and declines with a red background span.

align with what we previously observed while analyzing $GL1$, see Fig. 5. Low gas in the sliding-window log is captured by `minGas` and high gas by `maxFinalGas`. With `minGas` ≥ 0, the gas never drops below 0, thus all negative transitions are timely balanced by positive ones. Significant intervals in `minGas` are revealed through manual analysis and marked in Fig. 10, improvements are highlighted with a green background span and declines with a red background span:

1. Sub-logs after the first 120 days show improvements under both strategies `minGas` and `maxFinalGas`, respectively.
2. Sub-logs 270–320, 390–450, and 550–610 display that `minGas` suddenly drops under `goPos` and `goPosFast`.
3. The expected number of steps steadily increases in the first half of the recorded logs, and then cycles around 25 steps, with an exception in the later drops. The drops in `minGas` align with changes in the expected number of steps in the journey.

In between these drops, `maxFinalGas` peaks at different heights ranging from 200 to 400. Besides these drops, GrepS is able to sustain the user journeys on a constant level: `minGas` is often bounded by 0 and `maxFinalGas` fluctuates around 200.

RQ3. From the 517 users in $GL2$, 78% are commissioned by three different customers: customers c_1, c_2 and c_3 commissioned 260, 96, and 46 users, respectively, corresponding to 50%, 19% and 9% of the total users. Figure 11 shows the distribution of users belonging to customers c_1, c_2, and c_3 in relation to all users per sub-log. Observe the drift in users: c_2 commissions the largest share of users initially, but stops after 200 days. From there on, c_1 commissions constantly 80% of the users. We constructed three filtered sliding-window logs by only considering users for the same customer: $\mathbb{L}_{c_i}$ filters sub-logs from $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$ to users from customer $c_i \in \{c_1, c_2, c_3\}$ only.

Figure 12 compares the distribution of successful users per sub-log for the three customers with respect to the general performance of all users, showing that the users of customer c_1 consistently outperform those of customer c_2.

The extended technical error (see RQ2-A) only occurs for customer c_1. Although customer c_3

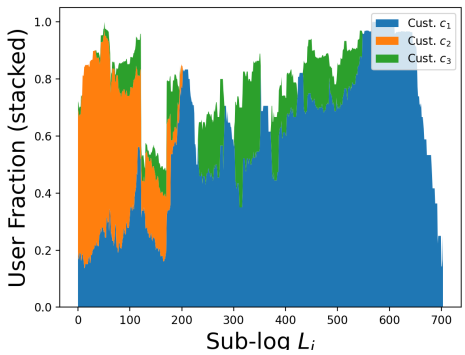


Figure 11: Distribution of users by customer per sub-log in $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$.

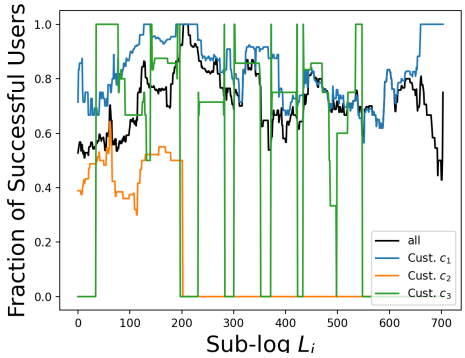


Figure 12: Distribution of successful users by customer in $\mathbb{L}_{49}^{\mathcal{I}_{\text{days}}}$.

strongly varies in the number of commissioned users, their average number of steps and `minGas` are very stable, `maxFinalGas`, however, has a very high variance. Customer c_3 shows two drops in `minGas`: one occurs slightly before the drop around sub-log 300 and the other does not align with previous observations. These two drops in performance should be used to improve the service offered to customer c_3 and its users.

In RQ1 we observed that the phase in which low gas is experienced coincides with the most demanding part of the user journey. We now shift the analysis to the three phases of the user journey: (1) sign-up, (2) task-solving, and (3) review. We lift the query from states to phases by encoding phases in the Uppaal model, such that minimum `gas` is mapped to a phase, and differentiate the severity of low gas; e.g., a `minGas` of 0 can be expected in the beginning of the journey, whereas a -30 low in the review phase depicts a serious problem at a late stage of the journey.

The low gas for customer c_1 appears mainly in the sign-up phase, customer-specific drops however move low gas into the task-solving or review phase. The customer analysis reveals that the `maxFinalGas` peak for customer c_2 appears after an improvement in `minGas` and `maxFinalGas` in sub-logs 0–120, which is also accompanied by stability in `minGas` afterwards, after which customer c_2 stops to commission users. The low gas occurs in the task-solving phase, favoring these sub-logs for further analysis.

10.4 Evaluation

From the perspective of a company considering automated analyses of user journeys, as proposed in this paper, it is essential to assess whether the proposed method yields actionable insights. The results reported for RQ1 in Section 10.3 demonstrated that user journey games derived from system logs can discover weaknesses in designed user journeys, and be used to improve and optimize these journeys . The company needs to implement additional actions in their service, which will improve user satisfaction and reduce costs in terms of resources. Further, the sliding window analyses for RQ2 and RQ3 detected major technical challenges with the service at certain points in time and showed that the user experience differs considerably for the users from three major customers. These results are evaluated by the third author, a long-term GrepS employee with experience in user and customer relations.

RQ1. The weighted game detects challenges early in Phase 2; in fact, this is reassuring for our analysis, as prior work at GrepS has reported that the users struggle more during the first three tasks [9]. However, a question that arises from our analysis of the derived user journey game is whether good user support during deviations from the planned journey may result in better overall satisfaction than if the planned journey had no deviations. It seems plausible that unplanned journeys that involve technical problems result in less motivated users who are less likely to successfully complete the journey. However, interactions with support

may also result in additional service to the user that yield positive weights in the overall game.

RQ2. We identified four separate periods to be of primary interest, highlighted spans in Figs. 10a–10f: one green (favorable) period, and three red (negative) periods. The periods were detected purely based on logs, no supplementary information was available. We paraphrase the feedback we got from the company on these four periods next. The initial favorable period occurred when three full-time equivalent developers were in an intensive development phase. The company was running out of funding at the time and focused all its effort on testing out a business-to-customer (B2C) version to secure new funding by building an extension to its existing business-to-business (B2B) solution. The favorable period lasted until about one month after the money ran out and all employees were laid off. The three subsequent negative periods all occurred during company-critical events where new funding was attempted but failed. The first and second periods coincide with a point in time when the two founders found a new (main) employer; the third period coincides when the point in time when the company's IP was sold to the present owner. Overall, our method thus seems to be able to identify both gradual technical system improvements as well as hardships that a company might face, e.g., as a result of losing key personnel.

RQ3. It has become clear during this evaluation is that the possibility of segmenting user journeys into sub-groups is needed for companies to make meaningful and timely changes to the software and the overall service. It seems reasonably clear that the three companies' data-generating mechanism (via users) is different, probably due to differences in the customer hiring process. By combining gas analysis (RQ2) with customer-specific analysis (RQ3), the recommendation for customer c_1 would be to investigate potential problems with the (first) sign-up phase closer. In contrast, customer c_2 has more issues during the (second) solve task phase. Differences between customer companies in the developer's capabilities and motivation to complete each skill analysis may also explain these differences in recommended actions.

10.5 Threats to Validity

We first consider threats to validity for each research question.

RQ1. In the construction of the user journey game, ties between concurrently recorded events are broken arbitrarily, leading to the automated discovery of different process models. In our experiments, the results and insights obtained under various generated models are comparable, including the ones where we fixed the order of the observed concurrent events.

RQ2. In the construction of the sliding-window sub-logs, the choice of window size μ is a trade-off between accuracy and generalization. The results are more accurate with a smaller window size, reflecting immediate temporal changes in the underlying windows. We observe that the start and end of the technical error are more detailed in $\mathbb{L}_{25}^{\mathcal{I}_{\text{days}}}$ than in $\mathbb{L}_{200}^{\mathcal{I}_{\text{days}}}$. However, structural changes could be hidden away with small window sizes. We could not observe structural changes with $\mathbb{L}_{25}^{\mathcal{I}_{\text{days}}}$, as the results show too many spurious changes. With larger window sizes, e.g. $\mathbb{L}_{100}^{\mathcal{I}_{\text{days}}}$ and $\mathbb{L}_{200}^{\mathcal{I}_{\text{days}}}$ are the 700 sub-logs divided into three notable ranges, 200–250, 250–450, and 450–700, each with steady results. The varying results observed in $\mathbb{L}_{25}^{\mathcal{I}_{\text{days}}}$ are "smoothed". Detailed results for mentioned window sizes are found in the repository [1]. Similarly, the window size also affects the sliding-window analysis performed for RQ3.

RQ3. After filtering the sliding window sub-logs by company c_1, c_2, and c_3, the resulting sub-logs are significantly smaller than the original log. The behavior of a few users thus has a large impact on the general analysis of the respective company.

We now consider threats to the validity of our experiment design. While $GL1$ and $GL2$ capture two distinct time spans of different lengths, i.e. 49 days and two years, the system logs are collected from the same company. Thus, our evaluation needs to be seen in the context of user journeys in digital services. Furthermore, our insights were evaluated from the third author's perspective, and not empirically. Therefore, our evaluation depends on his own experience within the company GrepS.

11 Discussion

In this section, we discuss two perspectives on the work reported in this paper. First, its possible implications on industrial practice and how practitioners work with user journeys. Second, its possible implications on theory development and how researchers work with formal methods. In summary, we believe that automated analyses techniques open for novel and less labor-intensive ways of working with user journeys. We further believe that data-driven model construction combined with automated analysis techniques, as investigated in our work, open up novel ways of working with formal methods and novel application domains for the techniques of our community.

11.1 Implications for industrial practice

User journeys are an established method to gain insights into the actual experiences of users. Until now, user journeys have largely been built by hand, collecting user feedback by means of, e.g., questionnaires to capture the user experience with the journeys. Existing tools (e.g., [36, 42]) offer limited support for automation that makes their application in larger settings very challenging and hinders the establishment of user-journey analysis as part of service development practice, as illustrated in our motivating scenario (see Sect. 1).

Methods that can automate both the construction and analysis of user journeys, such as the method proposed in this paper, open for a continuous assessment of the users' experience with a user journey as a service evolves. Our method uses interactions recorded in logs to construct the user journey game, which is then model checked for user-centric properties (see Sect. 7), automating major parts of the user-journey analysis, thereby accelerating the user-journey construction and analysis. For example, one could see this form of user-journey analysis integrated in visual dashboards that display the runtime performance of a service.

11.2 Implications for formal methods

Formal methods offer many advanced techniques for systematic model exploration and analysis. Usually, models are created by hand and checked against a desired property. In our work, we propose a procedure to automatically construct models from data sets. This is especially interesting for data sets that evolve over time, such as system logs, because it enables reuse of an analysis technique over a stream of models. Specifically, we have introduced a method for constructing automata from system logs by means of process mining techniques, enabling model checking to be performed without manual model generation. By considering the analysis of the stream of generated models, this approach opens for analyzing changes over time.

Data-driven model checking opens many interesting problems, which are not well understood in formal methods. The detection of changes in processes, which has been central in our analyses of different windows, is called *concept drift detection* in process mining. The presented method detects changes from the user's perspective, abstracting from system changes that do not impact the user journey. Bose *et al.* discuss four different types of concept drift [16]:

- *sudden*: a process is substituted by another process,
- *gradual*: a new process supersedes the old one and for a period of time both processes are observable,
- *recurring*: processes reappear periodically,
- *incremental*: the process change is not instantaneous but gradual over time.

It would be interesting to understand how such notions of concept drift in the underlying models generally affect the results of formal analysis techniques. In the use case considered in this paper, we observed a gradual drift, witnessed by three low gas periods in the sliding-window analysis. With a larger system log, one could possibly detect recurring changes, e.g., the re-establishment of previous journey properties. Observe that to detect incremental concept drifts, one would need analysis techniques that relate multiple models.

12 Conclusions and future work

This paper introduces a novel analysis pipeline for user journeys, based on the data-driven generation of formal models. Our generated models are weighted games, where the weights reflect user experience. The model construction is not subject to human inference but is built from system logs. Both single games, derived from a log, and sequences of games, derived from a sliding window view of a log, are considered. The paper proposes a method to automatically analyze derived models to gain insights into the user journeys of a service, by means of Uppaal Stratego queries using the Uppaal model checker. To the best of our knowledge, this is the first automatic analysis pipeline using formal methods in the context of service science and user journeys.

The proposed analysis pipeline was evaluated on an industrial case study and revealed challenges to the planned user journey of the service provider. The analysis of the derived game demonstrated that users' experiences fall in their accumulated feedback during the initial phases of the service. Our recommendations were reviewed and approved by an expert on user feedback in the company. We further performed a sliding-window analysis on a system log spanning two years, which suggested that a number of changes had occurred in the user journey. The company expert reviewed the detected changes and found that they aligned with key moments in the company's history. We finally showed that our analysis method can analyze user journeys for groups of users, filtering the log that generates the weighted games, to improve interaction with specific customer groups.

The work presented here opens many interesting possibilities for further work, both in formal methods and in service science. Our work so far has assumed that users and service providers have perfect knowledge of each other's possible actions. On the formal methods side, we therefore plan to study imperfect information games for user journeys with incomplete knowledge about user actions in the setting of, e.g., PRISM-games [19]. Furthermore, our current work is restricted by a fixed bound on loop unrolling; it would be interesting to directly analyze cyclic models. On the service science side, we plan to integrate our work with existing modeling languages for user journeys, such as CJML [27, 29], to automate the analysis of user-journey models that are manually reviewed today, and to provide feedback from our analysis in the visual language of these models. Finally, the analysis of sequences of formal models for concept drift [44] seems highly interesting to us; a starting point here could be methods for change point detection in a time-series by means of cost functions [32].

Acknowledgments. The authors thank Ruben Ernst (CTO of GrepS) for sharing data and Steinar Haugen (eXOReaction) for feedback on the use case evaluation. This work is part of the *Smart Journey Mining* project, funded by the Research Council of Norway (grant no. 312198).

References

[1] User Journey Games Repository. https://github.com/smartjourneymining/User-Journey-Games/releases/tag/SoSym2023

[2] van der Aalst, W.M., Rubin, V., Verbeek, E., van Dongen, B. F., Kindler, E., Günther, C. W.: Process mining: A two-step approach to balance between underfitting and overfitting. Software and Systems Modeling **9**(1): 87–111 (2010). https://doi.org/10.1007/s10270-008-0106-z

[3] W. M. P. van der Aalst: Process Mining - Data Science in Action. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4

[4] Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)

[5] Banham, A., Leemans, S. J., Wynn, M. T., Andrews, R.: xPM: a framework for process mining with exogenous data. In: Process Mining Workshops: ICPM 2021, pp. 85–97. Springer (2022)

[6] Banham, A., Leemans, S. J., Wynn, M. T., Andrews, R., Laupland, K. B., Shinners, L.: xPM: Enhancing exogenous data visibility. Artificial intelligence in medicine, **133**: 102409 (2022). https://doi.org/10.1016/j.artmed.2022.102409

[7] Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Proc. 19th Intl. Conf. on Computer Aided Verification (CAV 2007)). Lecture Notes in Computer Science 4590, pp. 121–125. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_14

[8] Berendes, C. I., Bartelheimer, C., Betzing, J. H., Beverungen, D.: Data-driven customer journey mapping in local high streets: A domain-specific modeling language. In: Proc. Intl. Conf. on Information Systems ICIS 2018. Association for Information Systems (2018)

[9] Bergersen, G. R., Sjøberg, D. I. K., Dybå, T.: Construction and validation of an instrument for measuring programming skill. IEEE Transactions on Software Engineering **40**(12): 1163–1184 (2014) https://doi.org/10.1109/TSE.2014.2348997

[10] Bernard, G., Andritsos, P.: CJM-ex: Goal-oriented exploration of customer journey maps using event logs and data analytics. In: Proc. BPM Demo Track and BPM Dissertation Award CEUR Workshop Proceedings 1920. CEUR-WS.org (2017)

[11] Bernard, G., Andritsos, P.: A process mining based model for customer journey mapping. In: Proc. CAiSE Forum 2017 CEUR Workshop Proceedings 1848, pp. 49–56. CEUR-WS.org (2017)

[12] Bernard, G., Andritsos, P.: CJM-ab: Abstracting customer journey maps using process mining. In: Information Systems in the Big Data Era - Proc. CAiSE Forum 2018. Lecture Notes in Business Information Processing 317, pp. 49–56. Springer (2018). https://doi.org/10.1007/978-3-319-92901-9_5

[13] Bernard, G., Andritsos, P.: Contextual and behavioral customer journey discovery using a genetic approach. In: Proc. 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019). Lecture Notes in Computer Science 11695, pp. 251–266. Springer (2019). https://doi.org/10.1007/978-3-030-28730-6_16

[14] Bertolini, C., Liu, Z., Srba, J.: Verification of timed healthcare workflows using component timed-arc petri nets. In: Foundations of Health Information Engineering and Systems: (FHIES 2012). Lecture Notes in Computer Science 7789, pp. 19–36. Springer (2013). https://doi.org/10.1007/978-3-642-39088-3_2

[15] Bitner, M. J., Ostrom, A. L., Morgan, F. N.: Service blueprinting: A practical technique for service innovation. California Management Review **50**(3): 66–94 (2008). https://doi.org/10.2307/41166446

[16] Bose, R.J.C., van der Aalst, W.M., Žliobaitė, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Networks Learn. Syst. **25**(1): 154–171 (2013). https://doi.org/10.1109/TNNLS.2013.2278313

[17] Bouyer, P., Cassez, F., Fleury, E., Larsen, K. G.: Optimal strategies in priced timed game automata. In: Proc. 24th Intl. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2004). Lecture Notes in Computer Science 3328, pp. 148–160. Springer (2004). https://doi.org/10.1007/978-3-540-30538-5_13

[18] Bouyer, P., Fahrenberg, U., Larsen, K. G., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Commun. ACM **54**(9): 78–87 (2011). https://doi.org/10.1145/1995376.1995396

[19] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A Model Checker for Stochastic Multi-Player Games. In: Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science 7795, pp. 185–191. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_13

[20] Crosier, A., Handford, A.: Customer journey mapping as an advocacy tool for disabled people: A case study. Social Marketing Quarterly **18**(1): 67–76 (2012). https://doi.org/10.1177/1524500411435483

[21] David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc Petri nets. In: Proc. 18th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012). Lecture Notes in Computer Science 7214, pp. 492–497. Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_36

[22] David, A., Jensen, P. G., Larsen, K. G., Mikučionis, M., Taankvist, J. H.: Uppaal Stratego. In: Proc. 21st Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015). Lecture Notes in Computer Science 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16

[23] Eckerson, W.W.: Performance dashboards: measuring, monitoring, and managing your business. John Wiley & Sons, 2010

[24] Følstad, A., Kvale, K.: Customer journeys: A systematic literature review. Journal of Service Theory and Practice **28**(2): 196–227 (2018). https://doi.org/10.1108/JSTP-11-2014-0261

[25] Fornell, C., Mithas, S., Morgeson, F. V., Krishnan, M.: Customer satisfaction and stock prices: High returns, low risk. Journal of Marketing **70**(1): 3–14 (2006). https://doi.org/10.1509/jmkg.70.1.003.qxd

[26] Halvorsrud, R., Haugstveit, I. M., Pultier, A.: Evaluation of a modelling language for customer journeys. In: Proc. Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2016), pp. 40–48. IEEE Computer Society (2016). https://doi.org/10.1109/VLHCC.2016.7739662

[27] Halvorsrud, R., Kvale, K., Følstad, A.: Improving service quality through customer journey analysis. Journal of Service Theory and Practice **26**(6): 840–867 (2016). https://doi.org/10.1108/JSTP-05-2015-0111

[28] Halvorsrud, R., Mannhardt, F., Johnsen, E. B., Tapia Tarifa, S.L.: Smart journey mining for improved service quality. In: Proc. IEEE International Conference on Services Computing (SCC 2021), pp. 367–369. IEEE (2021). https://doi.org/10.1109/SCC53864.2021.00051

[29] Halvorsrud, R., Sanchez, O.R., Boletsis, C., Skjuve, M.: Involving users in the development of a modeling language for customer journeys. Software and Systems Modeling, pp. 1–30. Springer (2023). https://doi.org/10.1007/S10270-023-01081-W

[30] Harbich, M., Bernard, G., Berkes, P., Garbinato, B., Andritsos, P.: Discovering customer journey maps using a mixture of markov models. In: Proc. 7th Intl. Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2017). CEUR Workshop Proceedings 2016, pp. 3–7. CEUR-WS.org (2017), http://ceur-ws.org/Vol-2016/paper1.pdf

[31] Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.: Trace alignment in process mining: Opportunities for process diagnostics. In: Proc. 8th Intl. Conf. on Business Process Management (BPM 2010). Lecture Notes in Computer Science 6336, pp. 227–242. Springer (2010). https://doi.org/10.1007/978-3-642-15618-2_17

[32] Killick, R., Fearnhead, P., Eckley, I.A.: Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association **107**(500): 1590–1598 (2012).

[33] Kobialka, P., Mannhardt, F., Tapia Tarifa, S. L., Johnsen, E. B.: Building user journey games from multi-party event logs. In: Proc. 3rd Intl. Workshop on Event Data and Behavioral Analytics (EdbA 2022). Lecture Notes in Business Information Processing 468, pp. 71–83. Springer (2022). https://doi.org/10.1007/978-3-031-27815-0_6

[34] Kobialka, P., Schlatte, R., Bergersen, G. R., Tapia Tarifa, S. L., Johnsen, E. B.: Simulating user journeys with active objects. In: Active Object Languages: Current Research Trends. Lecture Notes in Computer Science 14360. Springer (2023), to appear

[35] Kobialka, P., Tapia Tarifa, S.L., Bergersen, G.R., Johnsen, E.B.: Weighted games for user journeys. In: Proc. 20th Intl. Conf. Software Engineering and Formal Methods (SEFM 2022). Lecture Notes in Computer Science 13550, pp. 253–270. Springer (2022). https://doi.org/https://doi.org/10.1007/978-3-031-17108-6_16

[36] Lammel, B., Korkut, S., Hinkelmann, K.: Customer experience modelling and analysis framework a semantic lifting approach for analyzing customer experience. In: Proc. 6th Intl. Conf. on Innovation and Entrepreneurship (IE 2016). GSTF (2016)

[37] Larsen, K. G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2): 134–152 (1997). https://doi.org/10.1007/s100090050010

[38] Leemans, S. J. J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: Intl. Conf. Process Mining (ICPM 2019). pp. 25–32. IEEE (2019). https://doi.org/10.1109/ICPM.2019.00015

[39] Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Proc. First Intl. Conf. on Runtime Verification (RV 2010). Lecture Notes in Computer Science 6418, pp. 122–135. Springer (2010). https://doi.org/10.1007/978-3-642-16612-9_11

[40] Plotkin, G. D.: A structural approach to operational semantics. J. Log. Algebraic Methods Program. **60-61**: 17–139 (2004)

[41] Razo-Zapata, I.S., Chew, E.K., Proper, E.: VIVA: A visual language to design value co-creation. In: 20th Conf. on Business Informatics (CBI), pp. 20–29. IEEE (2018). https://doi.org/10.1109/CBI.2018.00012

[42] Rosenbaum, M.S., Otalora, M.L., Ramírez, G.C.: How to create a realistic customer journey map. Business Horizons **60**(1): 143–150 (2017). https://doi.org/10.1016/j.bushor.2016.09.010

[43] Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson (2020)

[44] Sato, D. M. V., De Freitas, S. C., Barddal, J. P., Scalabrin, E. E.: A survey on concept drift in process mining. ACM Computing Surveys (CSUR) **54**(9): 1–38 (2021)

[45] Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3): 379–423 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[46] Terragni, A., Hassani, M.: Analyzing customer journey with process mining: From discovery to recommendations. In: Proc. 6th Intl. Conf. on Future Internet of Things and Cloud (FiCloud 2018), pp. 224–229. IEEE (2018). https://doi.org/10.1109/FiCloud.2018.00040

[47] Terragni, A., Hassani, M.: Optimizing customer journey using process mining and sequence-aware recommendation. In: Proc. 34th Symposium on Applied Computing (SAC 2019), pp. 57–65. ACM Press (2019). https://doi.org/10.1145/3297280.3297288

[48] Thrane, C., Fahrenberg, U., Larsen, K.G.: Quantitative analysis of weighted transition systems. Journal of Logic and Algebraic Programming **79**(7): 689–703 (2010). https://doi.org/10.1016/j.jlap.2010.07.010

[49] Tueanrat, Y., Papagiannidis, S., Alamanos, E.: Going on a journey: A review of the customer journey literature. Journal of Business Research **125**: 336–353 (2021). https://doi.org/10.1016/j.jbusres.2020.12.028

[50] Vandermerwe, S., Rada, J.: Servitization of business: adding value by adding services. European Management Journal **6**(4): 314–324 (1988). https://doi.org/10.1016/0263-2373(88)90033-3