Noname manuscript No.
(will be inserted by the editor)

Reducing User Input Requests to Improve IT Support Ticket Resolution Process

Monika Gupta · Allahbaksh Asadullah · Srinivas Padmanabhuni · Alexander Serebrenik

Received: date / Accepted: date

Abstract Management and maintenance of IT infrastructure resources such as hardware, software and network is an integral part of software development and maintenance projects. Service management ensures that the tickets submitted by users, i.e. software developers, are serviced within the agreed resolution times. Failure to meet those times induces penalty on the service provider. To prevent a spurious penalty on the service provider, non-working hours such as waiting for user inputs are not included in the measured resolution time, that is, a service level clock pauses its timing. Nevertheless, the user interactions slow down the resolution process, that is, add to user experienced resolution time and degrade user experience. Therefore, this work is motivated by the need to analyze and reduce user input requests in tickets' life cycle.

To address this problem, we analyze user input requests and investigate their impact on user experienced resolution time. We distinguish between input requests of two types: *real*, seeking information from the user to process the ticket and *tactical*, when no information is asked but the user input request is raised merely to pause the service level clock. Next, we propose a system that preempts a user at the time of ticket submission to provide additional information that the analyst, a person responsible for servicing the ticket, is

Monika Gupta
Indraprastha Institute of Information Technology, Delhi, India
E-mail: monikag@iiitd.ac.in

Allahbaksh Asadullah
Infosys Ltd., India
E-mail: allahbaksh_asadullah@infosys.com

Srinivas Padmanabhuni
Tarah Technologies, India
E-mail: spadmanabhuni@gmail.com

Alexander Serebrenik
Eindhoven University of Technology, The Netherlands
E-mail: a.serebrenik@tue.nl

likely to ask, thus reducing real user input requests. Further, we propose a detection system to identify tactical user input requests.

To evaluate the approach, we conducted a case study in a large global IT company. We observed that around 57% of the tickets have user input requests in the life cycle, causing user experienced resolution time to be almost twice as long as the measured service resolution time. The proposed preemptive system preempts the information needs with an average accuracy of 94–99% across five cross validations while traditional approaches such as logistic regression and naive Bayes have accuracy in the range of 50–60%. The detection system identifies around 15% of the total user input requests as tactical. Therefore, the proposed solution can efficiently bring down the number of user input requests and, hence, improve the user-experienced resolution time.

Keywords Software Process · Machine Learning · Process Mining · Service Level Agreement · Ticket Resolution Time

1 Introduction

Infrastructure support is an integral part of software development and maintenance projects. It is usually handled by the Information Technology Infrastructure Support (ITIS) team [5][15]. The ITIS team is responsible for effective deployment, configuration, usage, management and maintenance of IT infrastructure resources such as hardware, software, and network [7]. The ITIS team ensures stable conditions for the production systems, and enhances performance of the features and products created by developers [35].

ITIS service is supported by an information system that is, referred to as ITIS information system. It is a ticketing system to simplify logging of tickets by customers (here software developers), facilitate tracking, monitoring, and resolution of tickets by analysts. The software developers request IT infrastructure support for their business projects. Efficient servicing of requests is essential for the success of the business [6][7]. Organizations usually have a well-defined workflow to streamline the servicing of IT support tickets [7]. Typically, a user reports a ticket in the information system by selecting the ticket category such as hardware, software, and network. Every category has a corresponding ticket reporting template to capture information deemed necessary for servicing the ticket. The ITIS information system automatically asks the user to provide details in the template for the chosen category. The user often provides textual description for the requested service, and category-specific details. On the basis of the ticket category, service level resolution time gets associated with the ticket. Service level clock is used to measure the service resolution time for every ticket and can have two states: pause (stops measuring the time) and resume (continues measuring the time). On the basis of business criticality, the service level resolution time for every ticket category is agreed a priori between the service provider and the client (user) as part of service level agreement [1][6]. The ticket is assigned to an *analyst*, a person responsible for servicing the ticket within the associated service level resolution

time. It is crucial to service within agreed service levels because nonfeasance leads to penalty on the service provider [8]. The analyst can ask for user inputs while resolving the ticket. When this happens the state of the ticket changes to *Awaiting User Inputs (AUI)*. To prevent spurious penalty on the service provider, the service level clock pauses while the ticket is in the AUI state. Nevertheless, the time spent while remaining in the AUI state adds to the user experienced resolution time. Indeed, passing back and forth of tickets for getting user input slows down the resolution process and degrades user experience [1]. Therefore, users typically like to have their tickets serviced with the minimum interaction requirements [1].

Analysts might require user input for various reasons such as incomplete or unclear information provided by the user, input information requirements not being defined clearly and completely, resolution of some tickets requiring specific information which is not intuitive to user, and analysts not interpreting the user inputs correctly [1]. These user input requirements can be avoided by ensuring that the information required for ticket resolution is provided by the user at the time of ticket submission itself.

Further given the paramount importance of honoring service level agreement, marking a ticket as *Awaiting User Inputs* is used as a sneaky way to achieve the service level target of resolution time [1][74]. In general, analysts are guided to request user inputs only if they genuinely need information for ticket resolution. However, previous research suggests that there are cases of non-information seeking, *tactical user input requests*, performed merely for the sake of pausing the service level clock, and thus degrading the user experience [74]. Therefore, to reduce the overall number of user input requests, we need to handle tactical user input requests in addition to ensuring the availability of relevant information at the time of ticket submission.

> The work presented in this paper is motivated by the need to analyze and reduce the overall user input requests in IT support tickets' life cycle, thus improving the user experienced resolution time and thereby enhancing user experience.

1.1 Real Motivating Example

Fig. 1 illustrates a real example of ticket's life cycle from the information system of a large global IT company. As shown, a user reports a ticket in category *software* and sub-category *install* with a short description: "Please install following software on my machine. This is a CORE software as per EARC". EARC is a software classification system internal to the organization. The user also provides category specific details such as hardware asset id, platform, software name, and software version. Based on a priori agreed service levels, the resolution time of 9 hours gets associated with the ticket. Next, the ticket gets assigned to an analyst. The analyst starts working on the ticket

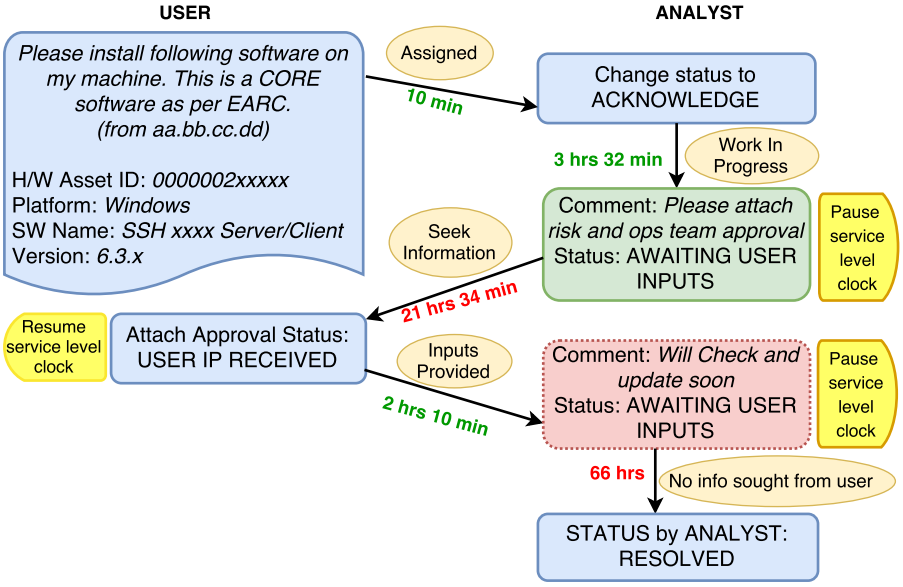

Fig. 1 A real example of ticket life cycle from a large global IT company, illustrating the problem of delay (here delay of 87 hours 34 min) in overall user experienced resolution time due to multiple user input requests by analyst.

and requests the user to "provide risk and ops team approval", changing the ticket status to *Awaiting User Inputs*. The user attaches the approval after 21 hours 34 min (as labeled on the edge in Fig. 1). This time is not counted towards service level resolution time but it is experienced by the user. Such *real*, information seeking user input requests could have been avoided if the user had been preempted at the time of ticket submission and required to provide the risk and ops team approval upfront. Some time after receiving the user's input, the analyst again changes the status to *Awaiting User Inputs* with the comment "Will check and update soon" (highlighted with the dotted outline). Inspecting this comment, we notice that no information is sought from the user; subsequently the ticket status changes to *Resolved* without any input from the user. The time for this transition, that is, 66 hours is also experienced by user but not included in the measured resolution time. Such *tactical*, non-information seeking user input requests need to be detected and handled separately. Summarizing, the resolution time measured by the service level clock that is, 5 hours 52 min (10 min + 3 hours 32 min + 2 hours 10 min) is significantly lower than the user experienced resolution time of 93 hours 26 min (10 min + 3 hours 32 min + 21 hours 34 min + 2 hours 10 min + 66 hours). Consequently, for the presented example, the measured resolution time is within the agreed threshold of 9 hours, i.e. there is no service level violation. However, the user did not experience the agreed service quality because of two user input requests. Both real and tactical user input requests add to the user experienced resolution time and need to be minimized.

1.2 Research Contributions

To analyze the IT support ticket resolution process and reduce overall user input requests, we make following research contributions:

1. To analyze the user input requests and their impact on user experienced resolution time, we discover runtime (reality) process from ticket data as opposed to measuring performance metrics, by novel applications of process mining [18]. Process mining consists of analyzing event logs generated from business process execution by the information systems [18][73]. The discovered process helps managers to decide if there is need for reducing real and tactical user input requests. Section 4 provides details on the process discovery and performance analysis.
2. To reduce *real* input requests in tickets' life cycle, an automated machine learning based preemptive model is designed. It preempts users at the time of ticket submission to provide additional information which the analyst is likely to ask, thus overcoming the limitations of ticket reporting templates. Section 5 presents the detailed model.
3. To reduce *tactical* input requests, a real time rule-based detection model decides whether the user input request by an analyst is *tactical* and identify its refined class to facilitate suitable actions against such requests. Section 6 focuses on the detection model.
4. To demonstrate the usefulness of the proposed solution approach for reducing user input requests, we conduct a case-study on ITIS data of a large global IT company. We triangulate our study by conducting a survey with both users and analysts. Sections 7 and 8 provide details on the case study.

2 Background and Related Work

In this section, we discuss the background of our work including the work related to ITIS process improvement and selection of technology used to address the problem.

2.1 ITIS Process Improvement

We start by discussing the work on ITIS process improvement. To be able to improve, it is important to measure the performance of the ITIS process. Standards for service organizations such as COBIT, ISO 20000, and IT infrastructure library, help to establish a set of practices and processes for effective and efficient service management [47][48][69]. They define objectives for IT infrastructure support and link them to high-level metrics such as the average ticket resolution time, and the number of ticket reopens. However, these high-level metrics are not sufficient to capture the finer grain level details required for investigating the root cause for poor performance and deciding on corrective actions.

Approaches based on mining ticket data for ITIS process improvement are more closely related to our work. Barash *et al.* focused on providing useful metrics to assess and improve performance of IT support ticket management process [6]. Bartolini *et al.* presented HANNIBAL, a business impact-driven decision support tool enabling business managers to make well informed decisions about the critical incident (ticket) management processes [7]. Li *et al.* demonstrated several statistical techniques for analyzing IT support ticket data to identify anomalies [44]. Bartsch *et al.* presented a Petri-net based approach in order to simulate service processes in terms of availability levels, thus assisting service providers and their customers when negotiating Service Level Agreement (SLA) during design time [8]. Ferreira *et al.* applied process mining to extract the existing process and assess whether a business process follows ITIL guidelines by conducting a real world case study [21]. Palshikar *et al.* proposed a domain driven-data mining approach to solve specific business problems such as overall workload reduction, improved ticket processing and better SLA compliance for ITIS services and validated on more than 25 large real-life ITIS datasets [49]. Also, a domain data mining approach is proposed to streamline the ITIS services in terms of levels (L1 or L2) at which tickets are handled for reducing service cost and improving service levels [48]. Weerdt *et al.* proposed a combination of trace clustering and text mining to enhance process discovery techniques with the purpose of retrieving more useful insights for incident management process [19]. While these works addressed the improvement of IT support ticket management process by leveraging data and process mining based techniques, they did not focus on analyzing and reducing user input requests. Their emphasized on achieving better service level compliance and meeting business objectives of IT support organization, by identifying inefficiencies.

The BPI challenge 2013 investigated event logs from an incident and problem management system, provided by Volvo IT Belgium, by applying process mining [74]. One of the challenge's aim was to investigate "wait user abuse" that is, user input requests during resolution of tickets. The challenge answered high level questions such as frequency of user input requests, behavior per support team and per organization, and average delay in ticket resolution time due to user input requests [50][70]. The challenge also leveraged process mining techniques for analyzing wait user abuse. However, they did not distinguish between *real* and *tactical* user input requests and did not provide any solution to reduce overall user input requests in the IT support ticket resolution process.

To the best of our knowledge, this is the first work on investigating the extent of user input requests and distinguishing them as *real* and *tactical*, and analyzing the impact of user input requests on user experienced ticket resolution time, using process mining techniques. Further, the work proposes a preemptive and detection model to reduce real and identify tactical user input requests respectively. The approach is validated through a case study in a large global IT company.

2.2 Process Mining of Software Repositories

We advocate process mining for analyzing user input request patterns in the IT support ticket resolution process because process mining has been advocated as a means of analyzing process reality in previous studies [71]. Process mining takes an event log as input and is used for various purposes such as process discovery, conformance verification, case prediction, history based recommendations and organizational analysis [71]. It has already been applied to analyze business processes from multiple domains [71][72]. Process mining of software repositories has diverse applications and has attracted the attention of several researchers due to the availability of vast data generated and archived in e.g. issue tracking systems, version control systems, and mail archives. Process mining of software repositories can provide Capability Maturity Model Integration (CMMI) assessors with relevant information and can support existing software process assessment and improvement approaches [59]. Some of the business applications of process mining of software repositories are: uncovering runtime process models [28][40], discovering process inefficiencies and inconsistencies [2][28], observing project key indicators and computing correlation between product and process metrics [66], extracting general visual process patterns for effort estimation and analyzing problem resolution activities [41], integrating multiple IS for process mining from control flow and organizational perspective [29], assessing development process components of student projects [53] and combining multiple repositories for assigning role to developers [54]. Process mining is applied on software repositories for different purposes. The main difference is that the techniques are tailored for the software development and bug (issue) resolution process and this work focuses on IT support ticket resolution process with huge alternative process variant possibilities and associated service level agreement constraints. The process structure highlights the need to analyze the ITIS process leveraging meta data, i.e., service level clock state to identify inefficiencies, and to suggest corrective actions for improvement.

2.3 Application of Recommendation Systems in Software Engineering

Recommendation systems in software engineering focus on providing information which can be valuable for a task and improve the productivity [56]. Robillard *et al.* presented diverse applications of recommendation systems in software engineering [56]. Some of the applications include recommendation system for: requirements engineering tasks such as finding experts for development tasks [46] and supporting build process [60], source code based tasks such as the correct usage of APIs [79] and code refactoring [9], and bug related tasks such as duplicate bug detection [65][67] and bug triaging [26]. However, the application of recommendation systems for reducing user input requests (both real and tactical) in IT support ticket resolution process is not yet explored which is the focus of the presented work.

Here, preemptive and detection models can be interpreted as recommendation systems because the predicted values are used to provide actionable insights [56]. The preemptive model predicts the information that can be needed for resolving the ticket and hence, the user is recommended to provide the same information at the time of ticket submission. Similarly, the detection model classifies if a user input request by an analyst is tactical or not, thus recommending managers to take actions for preventing tactical user input requests. Effectively, both preemptive and detection models are used to provide information in a proactive way to improve the ticket resolution process. Proactive (preventive) management of a process often improves efficiency by eliminating rework and the cost associated with the delays [1].

2.4 Information Needs in Software Engineering

Information needs of software engineers, i.e., information that engineers are looking for when performing software engineering tasks, have been extensively studied in the literature. In particular, the studies have focussed on the information needs arising when software is being designed [32], comprehended [57], changed[63][64] and released [51], when bugs are being fixed [14][11][23][42], and when the development teams activities need to be coordinated [10][78][42]. Similarly to this line of research we consider information needs arising during software engineering activities, in particular those pertaining to the IT support tickets. The nature of the IT support tickets makes them similar to the bug reports in the issue tracking system. Hence, in the discussion below we focus on positioning our work with respect to the studies that investigate the information needs for issue tracking systems.

Bettenburg *et al.* [11] conducted a survey on developers and users from Apache, Eclipse, and Mozilla to identify the information that makes good bug reports. Further, they designed a tool Cuezilla that provides feedback to the user at the time of ticket reporting for enhancing bug quality. Yusop *et al.* [78] conducted a survey focused on reporting usability defects and the analysis of 147 responses reveals a substantial gap between what developers provide and what software developers need when fixing usability defects. These studies captured the information deemed important in the opinion of the users and developers. As opposed to this line of work we do not rely only on the intuition and domain knowledge of users and developers but perform data driven analysis. Therefore, we focus on what information developers need as opposed to what information developers believe they need.

Ko *et al.* [42] looked at thousands of bug report titles for several open source projects and identified fields that could be incorporated into new bug report forms. It analyzed only the titles of the bug reports, not the comments to determine the information asked during bug resolution. Breu *et al.* [14] identified eight categories of information needs by analyzing the interaction between developers and users on a sample of 600 bug reports from the Mozilla and Eclipse project. They found a significant proportion of these interactions were

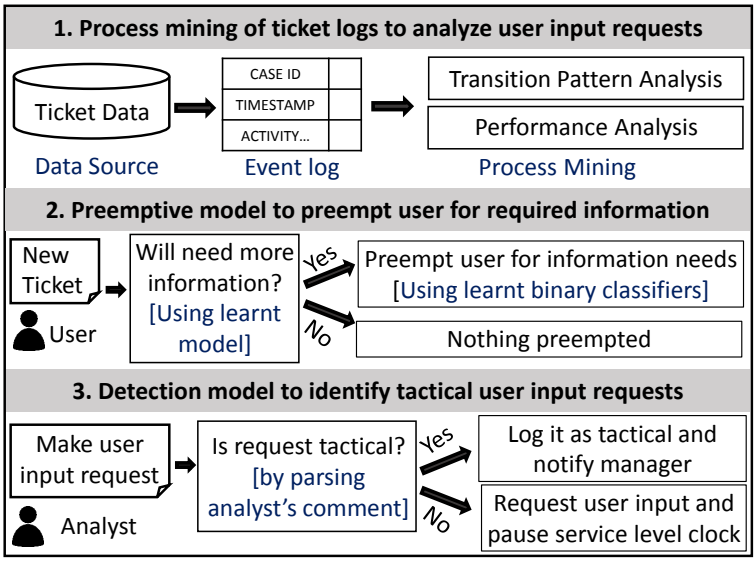

Fig. 2 Proposed approach involving three elements: 1. process mining of ticket logs to investigate process inefficiencies specifically user input request pattern, 2. preemptive model to preempt users with additional information needs at the time of ticket submission, and 3. detection model to identify tactical user input requests.

related to missing or inaccurate information. They observed some rhetorical questions (no information asked) however, did not consider them for detailed analysis. While the interaction between developers and users is analyzed, it is for a small sample of bug reports. We focus on IT support tickets and analyze the analysts comments for a large number of tickets.

3 Proposed Approach

To achieve the objective of analyzing and reducing user input requests in the ticket life cycle, we present an approach consisting of three elements as shown in Fig. 2. First, a **manager** analyses the user input requests in the ticket resolution process by applying process mining techniques on ticket data. This step helps the manager to make an informed decision regarding the need for reducing real or tactical or both user input requests. Accordingly, the preemptive model and the detection model are activated to reduce real user input requests and to identify tactical user input requests, respectively. To reduce *real* user input requests, the preemptive model preempts the **user** for required information at the time of ticket submission. To identify *tactical* user input requests, the detection model classifies the **analysts'** comment while marking a ticket as *Awaiting User Inputs* as tactical or not.

For analysis of user input requests, the process model is discovered from ticket data using process mining [71]. Discovered process model represents the

reality that is, the observed process behaviors. Each activity corresponds to a state in the process model. The transitions involving *Awaiting User Inputs (AUI)* as one of the states are investigated for the discovered process. For instance, to distinguish user input requests as real and tactical, we analyze the outgoing edges from *AUI*. If users provide inputs to user input request, the user input request is likely to be real and if the ticket is resolved by analysts without receiving any inputs from the user for the user input request, it is more likely a tactical user input request. To capture the delay caused due to user input requests, the performance is analyzed for the discovered process in terms of user experienced resolution time. Based on the transition pattern analysis and delays incurred by user input requests, a *manager* decides if there is need to reduce user input requests and of which type, real or tactical or both. Accordingly, the proposed preemptive and detection model are used to reduce real and tactical user input requests respectively and thus, to reduce the user experienced resolution time.

As depicted in Fig. 2, the *preemptive model* comes into play when a *user* submits a new ticket. The learnt model first determines if any user input request is likely to be made to service the ticket. If yes, the required information is demanded from the user. The preemptive model ensures that the information required for processing the ticket is asked upfront. The *detection model* is used when the *analyst* makes a user input request. It classifies the user input request as tactical or not by analyzing the analyst's comment when marking a ticket as *Awaiting User Inputs*. If the comment is detected to be tactical, it is logged in the ITIS information system and the manager can take appropriate actions, e.g., redefine service level resolution time or reassign the ticket.

This is a generic approach which can be adopted to reduce overall user input requests for any ITIS system. We discuss each of these elements in detail in the following sections.

4 Process Mining of Ticket Data: For User Input Request Analysis

To capture the extent of user input requests, investigate user response behavior to estimate the extent of real and tactical user input requests and observe their impact on ticket resolution time, process mining is applied on ticket data. Process mining of ticket data consists of data extraction and preprocessing to generate event logs followed by process discovery and performance analysis.

4.1 Data Extraction and Preprocessing to Derive Event Log

Data is downloaded from the ticket information system and is transformed to make it suitable for process mining [29]. One of the major challenges in applying process mining on software repositories is to produce a log conforming to the input format of a process mining tool [54]. Therefore, prior to applying process mining, an event log should be generated based on the information

from the ITIS information system. Event log consists of following attributes where each of them have their significance:

- *Case ID:* It uniquely identifies the case. It helps to visualize the life cycle of each case in discovered process models.
- *Activity*: Every event is related to some activity embarking the progress of case life cycle.
- *Time Stamp*: All events have an associated time stamp, a datetime attribute. It enables ordering of activities on the basis of execution time and allows analysis such as bottleneck identification that is, most time consuming transitions.
- *Other attributes*: Additional attributes can be useful for more interesting and diverse analysis.

In our study, the event log consists of events having four attributes: *Ticket ID*, *activity*, *time stamp*, and *service level clock state*. The fields are derived from extracted tickets' data (refer to Fig. 2) where *Ticket ID* uniquely identifies the ticket, and *activity* captures the progress of ticket life cycle e.g., logging of ticket, assignment of ticket to analysts, making a user input request and marking a ticket as resolved. Based on domain knowledge, the activities required for the given analysis and deemed to have an impact on overall process performance are captured as an event in the event log derived from ticket data. Unnecessary activities are not captured to avoid complexity that is, the derived process model may look like a spaghetti if there is a large number of activities [71]. All events have an associated *time stamp*, that is, the time when the activity is executed. Service level clock state, that is, *resume/pause* is stored for each event to capture its association with every activity. Hence, impact of an event on the measured service level resolution time.

To generate the log, we perform several preprocessing steps:

- *Mapping onto the event log:* All the data for ticket resolution process is recorded in the ITIS information system. Mapping of data fields to event log attributes needs to be done carefully depending on the type of analysis to be performed. For example,we select *Ticket ID* as case ID to visualize the life cycle of a ticket that is, the transition between different activities in the discovered process.
- *Selecting optimal level of granularity for activities:* Activities need to be recorded at the desired level of granularity so that detailed analysis can be performed. For example, on the basis of the reason for closing the ticket, we capture different activities: autoclosure of ticket if no action from the user in response to user input request, autoclosure if ticket is not explicitly closed by the user after resolution, and explicit closing of ticket by the user.
- *Resolving time stamp inconsistencies:* Data is captured in different time zones for global organization and thus needs to be handled carefully. We convert time stamps to a consistent timezone on the basis of the geographical location (captured for the ticket) where the ticket is submitted and resolved.

4.2 Process Discovery

As shown in Fig. 2, the preprocessed event log is imported to Disco [24] for runtime process discovery. The discovered process captures the ordering of activities to find a good characterization of the most common possible runtime process paths. Disco miner is based on Fuzzy Miner, a process mining algorithm that can be applied to less structured processes which is mostly the case for real-life environments [25]. To measure the extent of user input requests and distinguish them as *real* and *tactical*, we investigate the transitions involving *Awaiting User Inputs* state.

To understand the transition distribution, the percentage of transitions from the source state S to the destination state D is measured as:

$$TransitionPercentage, S_D = \frac{S \rightarrow D \ transition \ frequency \times 100}{Absolute \ frequency \ of \ S} \quad (1)$$

The *Awaiting User Inputs* state acts as D for incoming edges and S for outgoing edges. Incoming edges give us an intuition on the source state that is, the activities often followed by user input requests by analysts. Similarly, outgoing edges allow us to investigate user response behavior to user input requests by analysts and thus, possibility of requests being *real* and *tactical*.

4.3 Performance Analysis

We measure User Experienced Resolution Time (URT) and compare it with Service Level Resolution Time (SLRT) to capture the gap between them. To evaluate URT for ticket i, we use the total time elapsed between ticket assignment and final resolution of ticket, without excluding user input waiting time and non-business hours, as follows:

$$URT_i = ts_i(Resolved) - ts_i(Assigned) \quad (2)$$

5 Preemptive Model: For Real User Input Requests

The preemptive model is an automated learning based system deployed at the time of submitting a new ticket (refer to Fig. 2). It preempts the user to provide the information required for servicing the ticket. There are ticket reporting templates corresponding to the ticket category and subcategory chosen by the user. For example, if a user selects the category as *software* and subcategory as *install*, the ITIS information system automatically asks the user to provide details as per the corresponding template. While ticket reporting templates try to capture the required details, they have limitations motivating the need for the preemptive model:

– Users do not provide all the details asked in the initial template because of limited understanding or time [1]. A balance needs to be maintained

between two contradictory demands: ask as much information as possible as to help the analyst to service the ticket in the best possible way, and as little information as possible as not to put too much burden on the user submitting the ticket. It is not possible to make all the fields mandatory because this would make it difficult for users to submit their requests. The preemptive model can help in such situations by preempting only if the missed information is indeed crucial for processing the ticket. For example, if a user submits a ticket in the *software install* category and writes description as "install latest version of MS office" but leaves the version field blank, the system should allow the user to submit this ticket without preemption because an analyst can service the ticket by installing the latest version.
- If a user selects the wrong category for the ticket, the corresponding template will not capture the information required to resolve the ticket. The learnt model can preempt the required information because the ticket description provided by user is used as one of the features for preemption and the model does not rely on the chosen ticket category.
- Users tend to provide incorrect or unclear information [1] to pacify the system which the learnt model can preempt. For example, if a user mentions version as some random value such as *xx* or *2.3* for MS office then the learnt model still can preempt and indicate that the version has been asked for similar tickets.
- Some tickets have specific information needs which are not captured in the corresponding template. For example, if a user requests for installing a software that has a requirement such as approval for purchasing software license in case of specific proprietary software. Such information needs are not always intuitive for users hence can be preempted by the learnt model.

Effectively, the preemptive model should facilitate dynamic information collection for faster processing of the reported ticket. Since it is a preventive measure, it helps to improve efficiency by eliminating later interaction delays and enhances user satisfaction [1].

As depicted in Fig. 3, the major steps involved in designing the preemptive model are: preprocessing, feature extraction, training classification model and preemption for a new ticket at the time of submission. To preempt information needed for a given ticket, we learn the model to predict the following:

- **P1:** To process a given ticket, will there be user input request?
- **P2:** If there will be a user input request according to **P1**, what is the specific information that is likely to be asked by the analyst?

To train a supervised model for **P1**, the ticket is labeled as 1 if *Awaiting User Inputs* state is present in ticket life cycle otherwise 0. This information is derived from the event log extracted for each ticket.

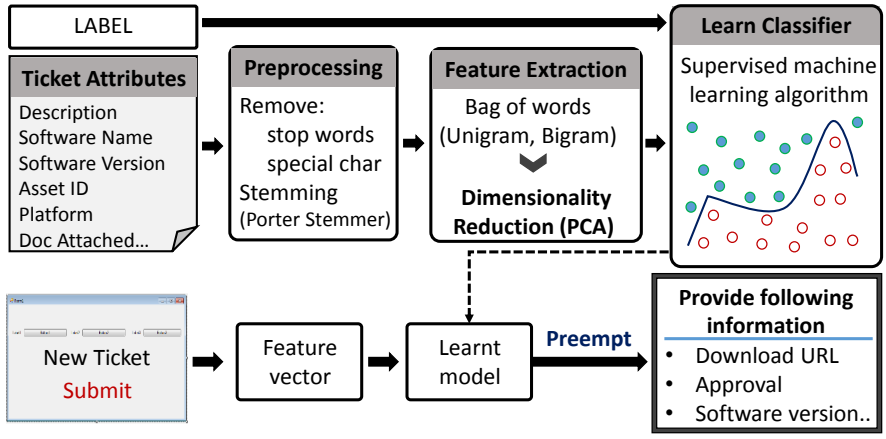

Fig. 3 Preemptive model to preempt users with additional information needs at the time of ticket submission with broadly two stages: training and preemption.

5.1 Ground Truth

To train and evaluate the model for **P2**, we establish the ground truth for information needs. The Ground Truth (GT) label for a ticket w.r.t a specific information, x is defined as follows:

$$GroundTruth, GT(x) = \begin{cases} 1, & \text{if x information asked in ticket's life cycle.} \\ 0, & \text{if x information not asked in ticket's life cycle.} \end{cases} \quad (3)$$

Firstly, information asked by the analysts (such as software name, software version, machine IP address, operating system, and manager approval) in the user input request comments are identified on the basis of managers' domain knowledge and manual inspection of the user input request comments. Manual inspection is performed by two authors (first and third) for disjoint set of comments (random sample of 1000 comments each) to identify the information needs. Information needs identified by both the authors are compared to create the consolidated list. Authors used different terms to represent the same information needs which were made consistent. Both the authors identified same information needs (that is, 23) with one exception that is, asking user the duration for which requested software will be used, identified by only one author as it is a rarely asked information. All the information needs mentioned by the managers turned out to be a subset of the consolidated list. The information needs solely identified by the authors are confirmed with the managers by presenting them the information needs along with the example analysts' comments where such an information is asked.

Comment annotation using keywords-based approach: Ground truth for every information need is established using a *keyword-based* approach [52]. A list of keywords corresponding to each information need is prepared iteratively.

Initial set of keywords is created using domain knowledge of the managers. For example, for information need software version, "software version, sw version, and software number" are some of the commonly used terms, thus, included in the keywords list. Porter stemming and case folding of comments and keywords is performed to improve the matching of keywords with the comments. If the comment contains the keywords, it is annotated with the corresponding information need. Thereafter, we (first and third author of paper) manually investigate the disjoint set of randomly selected unannotated comments (around 500 each) to identify the comments missed out using given set of keywords. Keywords are added to the initial set to capture the missed out comments. Also, disjoint set of annotated comments (50% of the total annotated by each author because it is typically a small set) is manually analyzed by the same two authors to eliminate wrongly annotated comments. Keywords are updated to distill the wrongly annotated comments. The comments are now annotated with the updated set of keywords. This process is repeated two to three times till very few/no updates are made in the set of keywords. Similarly, keywords are created for every information need. Keywords for an information can vary across the organization based on their specific terminologies.

Evaluate keywords based annotation: To evaluate the keywords based annotation, we decide to get a set of comments manually annotated and compare it with the keywords based annotation. We requested second year B.Tech in Computer Science students of the university for annotation. Each participant was promised a cash gift as token of gratitude. We received interest from nine students and shared the details with each of them. Three of them dropped out and the remaining six were given a short in person demonstration of the tool (screen shot made publicly available on github [27]) that we designed for convenient annotation. Each participant was given a set of 4000 different comments and 15 days time as agreed by students thus, make sure that they perform annotation without any pressure. Finally we received annotated files from five participants that is, 20000 annotated comments. First author randomly verified 50 annotated comments for each student for sanity check. For every information need, keywords based annotation is compared with annotation by students. We observed that annotation is consistent between the two for 90–95% comments for various information needs (we compare for five - software version, IP address, approval, operating system, and asking location/cubicle ID of the user). When manually inspected the inconsistently annotated comments, we found that in some cases keywords based annotation was incorrect and in some cases the student annotations were incorrect (attributed to human error) thus we ignored this inconsistency. This validates that the keywords based approach correctly annotates the comments.

To annotate the ticket, every user input request comment for a ticket is checked for its label and the ticket is labeled as 1 for the given information need if any of its comments are annotated with the same information otherwise it is labeled as 0.

5.2 Ticket Preprocessing

A ticket consists of a short description and fields capturing category-specific information about the ticket such as software name, version, platform, and attachment such as screen shot. Some of the ticket attributes can be free-form text data and hence require preprocessing as shown in Fig. 3. Common textual preprocessing practices such as case folding, stemming, stop words and punctuation removal are performed [56]. We perform stemming using Porter stemmer [55]. Removing classical stop words such as "a" and "the" is not sufficient because there are some stop words specific to the context. For context-specific stop word removal, we combine all the tickets into one text file and extract the term frequencies (tf) of the unique tokens from the text. We manually create a dictionary of stop words for given context which contains words such as "Dear", "Please", and "Regards".

5.3 Feature Extraction

As shown in Fig. 3, a bag-of-words feature model is used to represent each unstructured feature extracted from the ticket. A bag-of-words representation is known to extract good patterns from unstructured text data [80]. The bag-of-words model can be learnt over a vector of unigrams or bigrams or both extracted from text data. For instance, first we tokenize the description of ticket shown in Fig. 1 then stem the tokens that is, "following" is stemmed to "follow". After stemming, we remove the stop words: "on", "my", "this", "is", "a", "as", and "per". The resultant bag-of-words consists of unigrams: "please", "install", "follow", "software", "machine", "core", and "EARC". For most bag-of-words representations, gram (unigram or bigram) features found in the training corpus have weights such as binary or term frequency or term frequency-inverse document frequency [56][61]. We use bag-of-words feature with term frequency weights. Concatenation of bag-of-words features (both the unigrams and bigrams) with features corresponding to other fields such as platform name is used as the feature description for the entire ticket. Given the high-dimensional and sparse nature of the final representation, learning a classifier might be affected by the curse of dimensionality [12]. Therefore, we apply Principal Component Analysis (PCA), a feature vector dimension reduction technique with minimum information loss, such that 95% of the eigen energy is conserved [38]. For preemption, a feature vector extracted from the ticket submitted by a user is mapped to the reduced dimensional space learnt from the training data.

5.4 Training and Preemption

The preemptive system is learnt over features extracted using tickets' data at the time of submission. To address **P1**, a binary classifier is trained over a set

of labeled tickets to classify if user input request will be made for a given ticket or not. If the classifier predicts class as 1, i.e. a user input request will be made, the next question (that is **P2**) is to identify the specific information likely to be asked such as version number of a software or approval for processing. To identify the need for each of the possible information, an independent binary classifier is constructed. As shown in Fig. 3, when a new ticket is submitted the cumulative results of the learnt binary classifiers suggest the subset of information that could be further required to easily process the ticket. By dividing this complex task into simple binary classifiers, more flexibility is added to the preemptive model. If a new information need is identified in the future, a new binary classifier can be trained for the corresponding information without the need to retrain any of the existing classifiers. In our study, we have used a supervised learning model, Support Vector Machines (SVM) [16]. SVM is a binary linear classifier that attempts to find the maximum margin hyperplane, such that the distance of data points from either of the classes is maximized. Furthermore, it performs classification very effectively using a technique called a kernel trick, by implicitly mapping input data into a higher dimensional feature space, where linear classification is possible. SVM is a popular choice and is often used in the literature [4][20][37][45][65][77]. To compare the efficiency of SVM for the proposed preemptive system, we evaluate other commonly used classifiers such as naive Bayes [3], logistic regression [30], and random decision forest [33]. Performance of a classifier strongly depends on the value of its input parameters, whose optimal choice heavily depends on the data being used [68]. We choose the parameters for the classifiers using grid search [34] and heuristics [13].

5.5 Evaluation

A 50/50 *train/test split protocol* is followed to train and evaluate the classifier. In comparison with a more lenient protocol such as 80/20 split, the proposed split is less risk-prone in terms of generalizability [17]. To address the challenge of imbalanced class labels in train data, we perform random under-sampling of majority class as recommended by Japkowicz [36]. For creating the training set in a binary classification setting, 50% of data points are randomly taken from the minority class and an equal number of data points are randomly sampled from the majority class. Thus, it is ensured that the training data has an equal number of data points from both the classes. The remaining 50% data of the minority class and all the remaining points of the other class are included in the test split for evaluation (testing). To make a realistic estimation of the classifier performance and to avoid any training bias, we perform five times random sub-sampling cross validation (also called Monte Carlo cross-validation) where new training and test partitions are generated (at random) each time using the above protocol [76]. The evaluation metrics are averaged over the five rounds and the standard deviation is computed.

Accuracy places more weight on the majority class than on the minority class, thus prone to bias in case of imbalanced datasets [43]. Therefore, additional metrics such as precision and recall are used. Classes with label 1 and label 0 correspond to positive and negative classes respectively. TP and TN denote the number of positive and negative examples that are classified correctly, while FN and FP denote the number of misclassified positive and negative examples respectively. We evaluate performance of the learnt classification model on the test set using the following evaluation metrics:

Accuracy = (TP+TN)/(TP+FN+FP+TN)
Precision of positive detection = TP/(TP+FP)
Recall of positive detection = TP/(TP+FN)

6 Detection Model: For Tactical User Input Requests

Identifying tactical input requests is important because such requests degrade user experience as indicated by users in the survey conducted at the large global IT company (Section 8.2) and also evident from following user responses recorded in the ITIS information system of a large global IT company:

- "I have already provided all the necessary inputs. Please take actions."
- "Kindly let me know what inputs are required from my end. As mentioned in my earlier comments, I have already provided the necessary information but I still see the status as Awaiting user inputs. Its already been about a week since I submitted this request and the issue has not been resolved as yet. Request you to kindly do the needful."

While users and managers recognize tactical user input requests, following are the challenges in handling this practice thus, highlighting the need for automated detection system:

- After users recognize tactical user input requests and give feedback, the user experience has already been degraded. With the automatic detection system, it is possible to identify such requests in a proactive way and prevent users from receiving such requests thus, enhance user experience.
- Merely looking at the complaints will give biased impression because not every user will raise a complaint about such tactical user input requests. Raising complaints is an additional effort for the users which every user may not like to put. Moreover, many users (specifically new ones) are not familiar with the process and the fact that service level clock pauses when a ticket is in Awaiting User Inputs state thus, do not realize the need to complain about such experiences.
- A manager needs to look at the comments manually to decide if input request seeks any information or not, i.e. it is tactical. This control is human intensive and not always possible given the high number of input requests made by a team of analysts every day.

– Automatic detection allows to derive actionable insights thus, help managers make informed decisions. For example, if tactical requests are made by specific analysts then tackled at individual's level otherwise if practiced by majority of the analysts then take organization level decisions such as redefine service level resolution time limit or staff more analysts.

It is difficult to identify tactical user input requests because of the perpetual competition. For any technique to detect tactical user input requests, analysts will come up with ways to overcome them. The proposed detection model is an initial attempt to mitigate tactical user input requests. The detection model identifies tactical input requests in real time by analyzing analysts' comments when changing status to *Awaiting User Inputs*. For this, as shown in Fig. 2 we suggest to classify the user input requests using a keyword based rule classifier. Rules are a set of regular expressions derived to represent the keywords for tactical user input requests in a concise way. Rules are created for identifying user input requests where no direct information is asked. Domain knowledge of managers can be used to create an initial set of rules which can be updated iteratively by manually inspecting the tactical comments from the data corpus. Once the rule set is ready, any user input request by the analyst is checked against it for the classification. If the user input request is identified as tactical, it is logged in the ITIS information system and the manager is notified to take suitable actions.

As opposed to the preemptive component, we do not use machine learning because of the differences in the context: as part of preemptive model, information need is preempted for resolving a ticket at the time of ticket submission whereas in case of detection model, a comment by analyst during the ticket lifecycle is classified (not preempted) as tactical or not. For **P2**, the ground truth is created for a ticket by analyzing the analyst comments such as if version is asked in some comment then ground truth for a ticket is labeled as 1 w.r.t to class *version*. Therefore, the preemptive model takes ticket as input and preempts the information need using the learnt models. Unlike preemptive model, detection of tactical user input requests requires learning a classification model from labeled analyst comments. Since we manually create the ground truth label for tactical comments using keywords based approach (like done for **P2**), learning a classification model does not add value. Therefore, for given scenario, a set of rules to concisely represent the manually identified keywords for tactical user input requests is sufficient. Learning a classification model for tactical user input requests would have been an option if we had human annotated tactical user input requests available.

The detection model identifies whether an input request by the analyst belongs to one of the classes below. The classes are created on the basis of data analysis and discussion with the manager for a given case study. A separate set of rules is derived for each category.

– *Temporize:* The analyst indicates that the ticket will be handled soon and mentions things such as 'Work in progress', and 'Will check and update'.

Table 1 Experimental dataset details for the case study in a large global IT company.

Attribute	Value
Duration	One quarter of 2014
Total extracted *closed* tickets	593,497
Total categories	64
Closed tickets with category *Software*	154,092 (26%)
Total subcategories in *Software*	15
Total tickets with atleast one *AUI state*	88,039 (57%)

- *Invalid:* No valid character is present in the string. Comments such as empty strings or strings consisting of few special characters only.
- *Contact Me:* The analyst asks the user to contact her over phone or chat or in-person instead of asking for specific information.
- *Will Transfer:* The analyst informs the user that the ticket will be transferred to another analyst and marks the state as *Awaiting User Inputs*. The ticket is transferred to another analyst later instead of transferring directly.
- *Done So Check:* The analyst asks the user to check if the resolution is satisfactory. Ideally the analyst should mark the ticket as *Resolved* when done with resolution from their side and let the user reopen, if unsatisfied.

This classification helps managers to decide upon the appropriate course of action. For example, if the class is *Invalid*, the user input request can be blocked and if the class is *Contact Me* then it can be logged for clarification with the involved user and analyst to verify whether there was a need for contact.

For *evaluation*, we request managers to randomly select comments from the classified ones and indicate if they have been wrongly labeled. This ensures high precision but it is difficult to comment on recall. We do not know how many tactical comments are missed because of the incomplete class list or incomplete dictionary for a class.

Now, that we explained the details of the approach, we present the case study to illustrate its effectiveness.

7 Case Study: IT Support System of a Large Global IT Company

We performed a case study in a large global IT company to find out what is happening in the organization's ticket resolution process by applying process mining on ticket data and thus, derive actionable insights. We triangulated our results by conducting survey with users and analysts, to better understand the process in-practice and validate the data driven findings and inferences. Triangulation is known to improve the credibility and validity of the results [39][58]. Further, we demonstrated the usefulness of the proposed preemptive and detection model in reducing real and identifying tactical user input requests respectively. We chose this company for the following reasons: 1. it is a CMM level 5 company with a well-defined process in place, 2. it is a large global IT company with IT support as one of the crucial activities, 3. a large number of IT support tickets are reported, i.e. in order of a million per quarter, by diverse

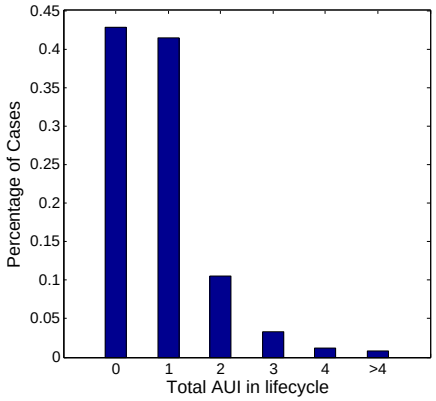

Fig. 4 User input request distribution - percentage of cases with given number of Awaiting User Inputs (AUI) state.

users, and 4. the service level metrics are continuously monitored by service level management and the organization has very high service compliance.

We download data of closed tickets for one quarter, archived in the organization's ticket system and store it in a relational database. We ignore open tickets because we want to analyze user input requests in the tickets' life cycle and the resolution time. Data includes the required information about a ticket starting from the time of ticket submission till it is closed. As summarized in Table 1, there are 593, 497 closed tickets labeled with 64 distinct categories such as software, desktop, and network. We select software category for the study because it is the most common category constituting 26% of total tickets and large enough to illustrate the proposed approach. The software category further has 15 subcategories such as software install, uninstall and upgrade.

The *Awaiting User Inputs* state is present in 57% of the tickets and 27.5% of them have multiple user input requests in their life cycle. With a total number of 125330 comments, we observe from the distribution curve depicted in Fig. 4 that the majority of the cases have one or two user input requests in the life cycle, and some cases have more than four user input requests. Similarly, in the Volvo IT organization ticket data investigated as part of the BPI challenge 2013 [74], the "wait-user" (user input requests) activity was present in 33% cases [50]. For many products, the "wait-user" activity is present in 67–84% of the cases, much more than the average of 33% [50]. This supports the observation for our data that user input requests are frequent in the ticket life cycle.

8 Experimental Results

In this section, we present our experimental results for the proposed approach including process mining of ticket data, and performance of the preemptive model and the detection model.

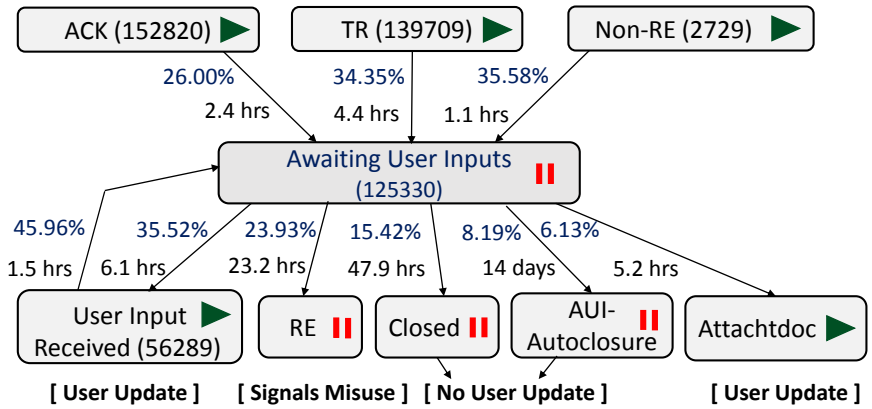

Fig. 5 *Awaiting User Inputs* state transition pattern showing user response classes where S_D and median transition time are edge labels. The state of the service level clock is indicated using play/pause icons. The activities are: ACK - ticket assigned to an analyst, TR - transfer of ticket to other analyst, Non-RE - user marks a ticket as not resolved, Awaiting User Inputs - analyst makes a user input request, User Input Received - user provides input for the user input request, RE - analyst marks a ticket as resolved, Closed - user closes the ticket, AUI-Autoclosure - ticket autoclosed as user did not provide inputs within the defined limit of 14 days, Attachtdoc - user attaches a document.

8.1 Process Mining of Ticket Data

As discussed in Section 4, we transformed data for all closed *software* category tickets (593,497 tickets) to make it suitable for process mining and analyze user input requests.

To generate the event log, we identified the activities to be captured for the analysis. All the important activities defined as part of ticket's life cycle are captured explicitly in the information system. We included in the event log a subset of the activities which we believe captures progression of tickets, can affect the performance and is sufficient for the analysis. Also we validated the list of activities with the manager. The list of activities along with the significance of each activity is made publicly available [27]. Ticket ID and time stamp corresponding to an activity is extracted from the information system. The service level clock state (resume/pause) for an activity is inferred from the documentation which clearly states the activities for which the service level clock pauses. For example, the service level clock pauses when asking for user inputs, marking a ticket as resolved, and closing a ticket.

Transition Pattern Analysis: We imported the event log to Disco for process model generation and made the derived process model for the complete process publicly available [27]. In Fig. 5, we present only the transitions involving *Awaiting User Inputs* activity for analyzing its transition patterns. We evaluate *Transition Percentage,* S_D (see Equation 2) for both incoming and outgoing edges, indicated as label in Fig. 5. State of service level clock is indicated using

play/pause icons and the median transition time is labeled on the edges to visualize time perspective.

Incoming edges to Awaiting User Inputs: From Fig. 5, we observe that analysts seek inputs when assigned a new ticket (*ACK*) or when a ticket is transferred (*TR*) to them from other analysts for 26.00% and 34.35% of the instances respectively. It indicates that as they start working on the ticket, they identify a need for additional information hence start by asking for inputs. Interestingly, *Awaiting User Inputs* is a successor state for *User Input Received* (for 45.96% of the instances) signaling that input from user leads to another user input request. The most common source states for *Awaiting User Inputs* are *ACK*, *TR*, and *User Input Received*, together constituting around 90% $((152820 \times 0.26 + 139709 \times 0.3435 + 56289 \times 0.4596)/125330)$ of the total incoming transitions (refer Fig. 5). *Non-RE* (user marks a ticket as not resolved after analyst says its resolved) is followed by user input request in 35.58% of the instances but constitutes merely 0.7% $(2729 \times 0.3538/125330)$ of total user input requests.

Outgoing edges from Awaiting User Inputs: We explore outgoing edges and classify destination states broadly into the following two classes:

- *User update:* We observe that users provide inputs, *User Input Received* (comment from user) or *AttachtDoc* (document attached by user), for around 42.00% of the user input requests. We conjecture that user input requests with these destination states are mostly information seeking (real) thus, updated by user. For around 15.42% of the instances, the user does not provide information and explicitly closes a ticket instead, i.e., the destination state is *Closed*.

- *No update:* As shown in Fig. 5, 23.93% of the total *Awaiting User Inputs* state transit to *RE* (resolved) without any update from user. We conjecture that such user input requests are more likely to be the *tactical* ones as the analysts managed to resolve the ticket without receiving user inputs. For 8.00% of the cases, a ticket is auto-closed, i.e. destination state is *AUI-Autoclosure* because no user action is performed in response to the user input request within the defined time limit of 14 days (as enforced in the given information system).

The above conjecture w.r.t. destination state for Awaiting User Inputs will be revisited in Section 9.

Performance Analysis We consider for analysis only the cases which are resolved and never reopened. There are 105,539 such cases for which we evaluate user experienced resolution time.

The software category has tickets with three service resolution time thresholds as per the organization's service level agreement: 9 hours, 18 hours and 36 hours. We group tickets into three categories on the basis of same service level resolution time. As shown in Table 2, *a high percentage of cases have user experienced resolution time more than the agreed service level resolution time.*

Table 2 Gap between Service Level Resolution Time (SLRT) and User Experienced Resolution Time (URT).

Class	#Cases	SLRT	Median URT	Cases with URT>SLRT
1	16,110	9 hours	21 hours	62.46% cases
2	83,939	18 hours	26.34 hours	72.49% cases
3	5,490	36 hours	77.84 hours	76.14% cases

However, the service level violation is recorded for very few cases[1] because the waiting time is not counted towards the measured resolution time. The median resolution time experienced by user is 21 hours, 26.34 hours, and 77.84 hours for cases with service level resolution time of 9 hours, 18 hours and 36 hours (refer to Table 2) respectively. Similarly, as analyzed in Volvo IT organization ticket data for the BPI challenge, on an average 34% of the user experienced resolution time was due to user input requests [70]. This highlights that the user experienced resolution time is much higher than the measured resolution time due to user input requests, thus a bottleneck in the ticket resolution life cycle.

We observe from Fig. 5 that the median transition time for *Awaiting User Inputs* to *RE*, potentially tactical is much higher (23.2 hours) than the median transition time for *Awaiting User Inputs* to *User Input Received* (6.1 hours) and *Attachtdoc* (5.2 hours), potentially real. Therefore, while both real and tactical user input requests add to the user experienced resolution time, tactical user input requests cause relatively more delay.

To summarize, from process mining analysis, we observe that 57% of the tickets have user input requests in the life cycle. Users provide input to around 42% of the total user input requests which we consider as potentially real requests. For around 23% of the cases, the ticket is resolved without any user inputs, thus corresponding to potentially tactical requests. User input requests cause a significant gap between the measured resolution time and the user experienced resolution time. The findings clearly highlight the need to reduce real and tactical user input requests.

Next, we triangulate our results by conducting survey with users and analysts of the organization.

8.2 Survey of Users and Analysts

We designed two short surveys (Tables 3 and 4) to understand users' and analysts' experience with the IT support process and in particular user input requests. The choice of options for Q3 and Q4 in Table 3 and 4 respectively was made after discussions with the manager and validated with randomly selected participants for each of the respondents' groups till all the pilot study respondents were satisfied. Still to make sure that survey respondents can provide applicable reasons if not included in the given list, we provided *other* as

[1] We cannot reveal exact numbers because of confidentiality

Table 3 User Survey Results [95 Responses]: User experience with the IT support services provided in last six months.

Q1: In last six months, how many tickets have you submitted?
No request at all - 0 Up to 5 - 43.16%
Up to 10 - 28.42% More than 10 - 28.42%

Q2: How often were you asked to provide at least one input?
Never - 1.05% Rarely - 10.53% Sometimes - 46.31%
Often - 26.31% Always - 15.79%

Q3: Below is the list of possible reasons explaining why you were asked to provide inputs. Rank given reasons from 1 to 5 where 1-most frequent and 5-least frequent. NA if Not Applicable.

Reasons	**1**	**2**	**3**	**4**	**5**	**NA**
1. Complex issue that required specific information	**33**	**28**	11	13	0	10
2. Initial request you submitted was incomplete or unclear	11	16	**29**	**26**	1	12
3. You felt the asked input was unnecessary to resolve the issue	9	16	**27**	**28**	2	13
4. Approvals	**39**	22	13	10	1	10
5. Others.....						

Q4: Were any of your tickets auto closed because you could not provide inputs within the time constraint of 14 days as enforced in ticket system?
Yes - 28.42% No - 71.58%

Q5: Did you ever decide to leave a ticket unresolved because you were asked to provide inputs multiple times?
Yes - 23.16% No - 76.84%

a free form option. We could not include the option of unnecessary user input requests in the analysts' survey for company policy reasons. The survey was administered after analyzing the findings from the process mining of ticket data to triangulate the results. We sent an email with survey to 40 *randomly selected analysts and* 125 *users* (from diverse project teams) from the organization. We received a total of 28 (around 70%) responses from analysts and 95 (around 76%) responses from users. Relatively high response rate can be attributed to the fact that these are very short objective survey and many reminders were sent. Survey and anonymized responses are made publicly available [27].

Analysis of Users' Survey: Table 3 presents the results of the users' survey consisting of five mandatory questions. This survey is about the users' experience with IT support services provided to them in the last six months. All the survey participants submitted ticket in the last six months out of which 57% users submitted more than 5 tickets (refer to Q1 in Table 3). User responses to Q2 show that *most of the users are asked to provide inputs* (*sometimes to always*). Responses to Q4 and Q5 confirms that users decide to leave tickets unresolved or let them auto close when asked to provide multiple inputs. We contacted the users who replied *Yes* to Q4 or Q5 over email and asked to provide the reasons for their choice. Based on the responses, we identify the following reasons behind the decision:

– users give up if it is not crucial to get the ticket resolved and they find it difficult to provide the information asked.

- users submit a new ticket which gets assigned to a different analyst and resolved with fewer or no user input requests. This is more common in the cases where they feel that unnecessary user input requests are being made by the analyst.
- users find alternative means of solving the problem that is, discuss with colleagues or search online.

Results of Q3 from the survey reveal that majority of the users believe that the most frequent reasons to ask for inputs are *missing approvals* and *need for specific information to resolve complex issues*. Users also expressed that the inputs were asked because they provided incomplete or unclear information at the time of ticket submission. Around 86% users (only 13 select *Not Applicable* out of 95) agree that *asking for unnecessary inputs* is one of the reasons though not the most frequent one. We received eight *other* answers. Following is the list of consolidated reasons mentioned in *other* option:

1. When a ticket needs to be transferred to another analyst, the service level clock is paused by marking ticket as *Awaiting User Inputs* till it is transferred to the other analyst.
2. In order to avoid a service level breach, tickets are closed without proper resolution and then reopened. For reopened ticket, analysts again ask all the previously provided information.
3. A ticket is marked as *Awaiting User Inputs* when analysts make request for license purchase to the concerned authorities.
4. The analyst asks to provide information such as software version, physical address of system and download URLs to process ticket.

The above additional reasons mentioned by the users are specific cases of requesting user input. The answers (points 1 and 2 above) highlight the different ways for pausing service level clock without asking any information from the user. We deem those user input requests as tactical. In other cases, the user quote the information they were asked to provide for the ticket resolution.

Analysis of Analysts' Survey: Analysts were asked four mandatory questions on the basis of their experience with IT support ticket resolution in the last six months as shown in Table 4. The majority of the survey participants (75%) have more than a year of work experience with IT service team of the organization. Responses to Q2 and Q3 reveal that while *analysts often ask for user inputs*, asking inputs multiple times for the same ticket is perceived as relatively infrequent. Comparison between ranking of Q3 from Table 3 and Q4 from Table 4 brings out interesting findings. While user ranked incompleteness of the initial report as a comparatively less frequent reason, analysts consider it as the most frequent reason. It highlights the difference between the expectations of analyst and of the user [11]. Reasons 3 and 4 in Q4 (refer to Table 4) support the observation of having *Awaiting User Inputs* as a subsequent state after *User Inputs Received* in Fig. 5. We received five *other* answers for Q4 and the consolidated reasons are as follows:

Table 4 Analyst Survey Results [28 Responses]: Analysts' experience with IT support ticket resolution in last six months.

Q1: For how many years have you been working as an analyst?
<1 yr - 25.00% Up to 3 yr - 46.43%
Up to 5 yr - 14.29% >5 yr - 14.29%

Q2: For approximately what percentage of all the tickets, do you need to ask for user inputs atleast once?
<20% : 10.71% 21–40% : 28.57% 41–60% : 35.71%
61–80% : 3.57% 81–100% : 21.43%

Q3: For approximately what percentage of all the tickets, do you need to ask for user inputs more than once?
<20% : 50.00% 21–40% : 14.29% 41–60% : 17.86%
61–80% : 17.86% 81–100% : 0%

Q4: Below is the list of possible reasons to ask for user inputs.
Rank given reasons from 1 to 5 where 1-most frequent and 5-least frequent.
NA if Not Applicable.

Reasons	**1**	**2**	**3**	**4**	**5**	**NA**
1. Incomplete information provided in the initial report	**18**	6	3	1	0	0
2. Complex issue that required specific information	4	**11**	7	6	0	0
3. Dependency between user inputs thus, need to ask for inputs sequentially	4	4	**12**	8	0	0
4. User provided wrong or unclear information	2	7	6	**12**	1	0
5. Others.....						

1. Users submit a ticket in a wrong category, thus the analyst needs to change the category after analyzing the ticket and ask the user for the required information accordingly.
2. Users miss required attachment such as approval and license.

The answers highlight the points missed by users thus, analysts had to ask for user inputs.

The survey supports the findings from the ticket log analysis that analysts frequently mark ticket as *Awaiting User Inputs* (Q2 of Table 3 and Table 4) for two reasons: seek inputs from users to process tickets (Q3 of Table 3 and Q4 of Table 4), and for the sake of pausing service level clock (Q3 of Table 3). The frequent user input requests degrade user experience (from Q4 or Q5 of Table 3) despite of very high service level compliance. This reinforces the need to address the problem by ensuring that maximum information required for resolution is asked upfront and misuse of user input requests is discouraged. To achieve this, we evaluate the effectiveness of the proposed preemption and detection model.

8.3 Preemption Model

To demonstrate the usefulness of the proposed preemption, we conducted experiments on a total of 96,756 closed tickets belonging to the most frequent subcategory, that is, install, within the software category. We chose this because every category has different information requirements thus for the illustration purposes we believe one category is sufficient. The approach can be similarly

Table 5 Number of class-wise data points in the ground truth and test-train split for the proposed preemptive model. Class 1: if the information is asked in the ticket life cycle, Class 0: information is not asked in the ticket life cycle.

Preempted	Ground Truth		Train Set		Test Set	
Information	**class 1**	**class 0**	**class 1**	**class 0**	**class 1**	**class 0**
Awaiting User Inputs	55,398	41,358	20,679	20,679	34,719	20,679
Software Version	1,174	95,582	587	587	586	94,996
Approval	3,686	93,070	1,843	1,843	1,843	91,227
IP Address	2,750	94,006	1,375	1,375	1,374	92,632

applied to other categories because the overall characteristics are common across the categories [1].

To learn the model for **P1** (to process the ticket, will there be user input request), tickets are labeled on the basis of *Awaiting User Inputs* state in life cycle. As shown in Table 5, $55,398$ (57.25%) tickets belong to class 1, that is, have at least one user input request in the life cycle. The bias due to tickets with only tactical user input requests in the life cycle will affect the outcome of $P1$ because the model is learnt to predict class for a ticket as 1 (that is, some information will be asked) even if it had just tactical user input requests. However, outcome of $P2$ will take care of this limitation: if $P1$ predicts class as 1, binary classifiers for every information need will be executed and all of them will give the output as 0 because none of those information were asked for the given ticket. Therefore, user will not be preempted to provide any information. Moreover, there are only 3117 tickets (that is, around 6% of total tickets in class 1 for $P1$) which just had tactical user input requests and still were assigned ground truth label as 1 thus, ignored to eliminate such cases. Effectively, the preemptive model remains independent of detection model and accurately preempts user for the additional information needs.

We observe from comments that different information is asked by analysts such as manager approval, operating system, location or cubicle ID, machine ID, IP address, project code, purpose of download, problem screen shot, download URL, software name and software version. Information requested by the analysts can, therefore, be categorized into three categories: information such as the IP address or the machine ID that can be derived automatically; information such as software name or software version that is explicitly asked in the ticket template; information such as the manager approval that is not explicitly asked in the ticket template but might be requested by the analyst under specific circumstances.

Ground truth is labeled for the information needs using a keyword-based approach (as discussed in Section 5.1). Table 6 presents the example information needs for each category along with the example keywords. Amanat (a Urdu word meaning fidelity) id is one of the keyword for Machine ID because it is a term used in the company to refer to a machine. To represent each category we address **P2** (what specific information is likely to be asked) for the IP address, software version, and manager approval, corresponding to 3.5%, 1.5% and 4.8% of the $77,333$ user input requests derived from $55,398$ tickets. All the keywords

Table 6 Different category of information needs with example information needs and their corresponding example keywords. Category 1 - information that can be derived automatically, Category 2 - information that is explicitly asked in the ticket template, and Category 3 - information that is not explicitly asked in the ticket template but might be requested by the analysts.

Category	Information Need	Example Keywords
Information that can be derived automatically	IP Address	ip, ipv4, ipaddress
	Machine ID	Amanat id, mac id, machine name, machine id, hardware id, asset id, hw name
	Location of user	location details, building no., cubicle id
Information that is explicitly asked in the ticket template	Software Version	Software version, sw ver, software number
	Operating System	OS details, platform, Operating system details
	Software Name	Software name, sw name
Information that is not explicitly asked in the ticket template but might be requested by the analysts	Manager Approval	DM Appr, DM's approval, approval DM, approval PM, project manager approval, PM approval
	Software License	PO details, PO number, license details, Software asset id, Purchase order
	Screen Share	msra, share screen

for IP address, software version and manager approval are made publicly available [27], however, labeled data could not be shared because of company policy concerns. We notice from ground truth in Table 5 that a relatively small percentage of tickets belong to class 1, i.e. the data is imbalanced, likely to overfit to the majority class [31].

The following information is extracted from a user submitted ticket: description, software name, software version, platform, doc-attached and time of reporting. Doc-attached is a binary field indicating the presence or absence of an attachment. Platform is a categorical attribute with seven unique values such as Windows, Linux and Unix. Time of reporting is mapped to three ranges in a day, that is, morning (before noon), afternoon (from noon to 4 PM), and evening (after 4 PM). Overall we have 21 possible values for time corresponding to the seven days of the week. Description, software name and software version are free-form text fields. The data from all the three fields for a given ticket are concatenated and preprocessed using case folding, stemming (using the Porter stemmer [55]) and stop words removal. We make the manually created stop words dictionary, on the basis of term frequency, for given context publicly available [27]. Also we remove all the punctuation marks except period because period is used in the IP address mentioned in the description.

As shown in Table 5, the testing and training data is created as per the 50/50 *train/test split protocol* with random under-sampling of majority class (cf. Section 5). Preprocessed text field for tickets in training data is represented as a term frequency vector of both unigrams and bigrams. Many unigrams and bigrams have very low frequency adding to the feature sparsity thus, we eliminate them by setting the term frequency threshold as 150. We started with a low threshold and tried for random values such as 50, 100, 150 and 200, and observed that 150 works the best given the trade-off between the model

Table 7 PCA is applied for reducing feature dimension and reduced feature vector is used for training. Table presents number of features before and after applying PCA.

Preempted Information	#Features before PCA	#Features after PCA
Awaiting User Inputs	1357	493
Software Version	33	17
Approval	119	50
IP Address	96	42

Table 8 Table showing the performance of the prediction model by comparing various popular classifiers with the proposed SVM. The best results are from SVM for any information need. NB - Naive Bayes, LR - Logistic Regression, RDF - Random Decision Forest, SVM - Support Vector Machine.

	Evaluation Metric	Awaiting User Inputs	Version	Approval	IP Address
NB	Accuracy	54.38 ± 0.09	62.19 ± 0.54	70.41 ± 0.29	60.41 ± 0.72
	Precision	53.42 ± 0.08	63.71 ± 2.29	73.54 ± 0.23	63.21 ± 1.24
	Recall	68.48 ± 0.32	57.58 ± 7.03	63.75 ± 1.20	49.92 ± 1.06
LR	Accuracy	61.85 ± 0.17	62.84 ± 0.98	72.55 ± 0.53	63.13 ± 0.47
	Precision	62.21 ± 0.11	65.78 ± 1.33	76.98 ± 0.80	65.95 ± 0.52
	Recall	60.37 ± 0.45	53.63 ± 2.91	64.35 ± 0.75	54.30 ± 1.46
RDF	Accuracy	99.51 ± 0.01	91.36 ± 0.92	97.73 ± 0.30	96.58 ± 0.42
	Precision	99.83 ± 0.02	94.78 ± 1.45	99.36 ± 0.24	98.39 ± 0.56
	Recall	99.20 ± 0.03	87.60 ± 2.84	96.07 ± 0.75	94.72 ± 0.72
SVM	Accuracy	99.83 ± 0.01	94.96 ± 0.69	99.28 ± 0.17	98.69 ± 0.14
	Precision	99.94 ± 0.02	95.59 ± 0.92	99.56 ± 0.23	98.89 ± 0.20
	Recall	99.73 ± 0.02	94.28 ± 1.82	99.00 ± 0.24	98.49 ± 0.26

computation time and the performance. The reduced feature set of unigrams and bigrams is concatenated with the other three features (platform, doc-attached and time of reporting) to represent a ticket. Thereafter, we reduced the dimension of the ticket feature vector by applying PCA. We notice from Table 7 that the feature length is different for models corresponding to different information needs because the training data set is different.

Using the list of features and the labeled training data set, a SVM is trained with different kernels such as linear, polynomial and Radial Basis Function (RBF) kernel, using LIBSVM [16]. We found experimentally that RBF kernel performed the best with $c = 8$ and $g = 2$ where c and g are the input parameters. A grid search is performed using a validation set and $c = 8$ and $g = 2$ are obtained as the best set of optimal parameters [34]. The performance of the learnt classification model is shown in Table 8 on the test set using evaluation metrics discussed in Subsection 5.5. Further, the performance of the proposed SVM classifier is compared with some baseline and popular classifiers in the literature such as naive Bayes, logistic regression, and Random Decision Forest (RDF) (refer to Table 8). For logistic regression the threshold hyperparameter is manually fine-tuned to be 0.5. Breiman *et al.* discuss some experimental heuristics to tune the parameters of RDF [13]. Based on those intuitions, the parameters of RDF used in our experiments are number of trees = 200, bootstrap ratio = 0.7, and subset of features per tree = 0.6. The average results

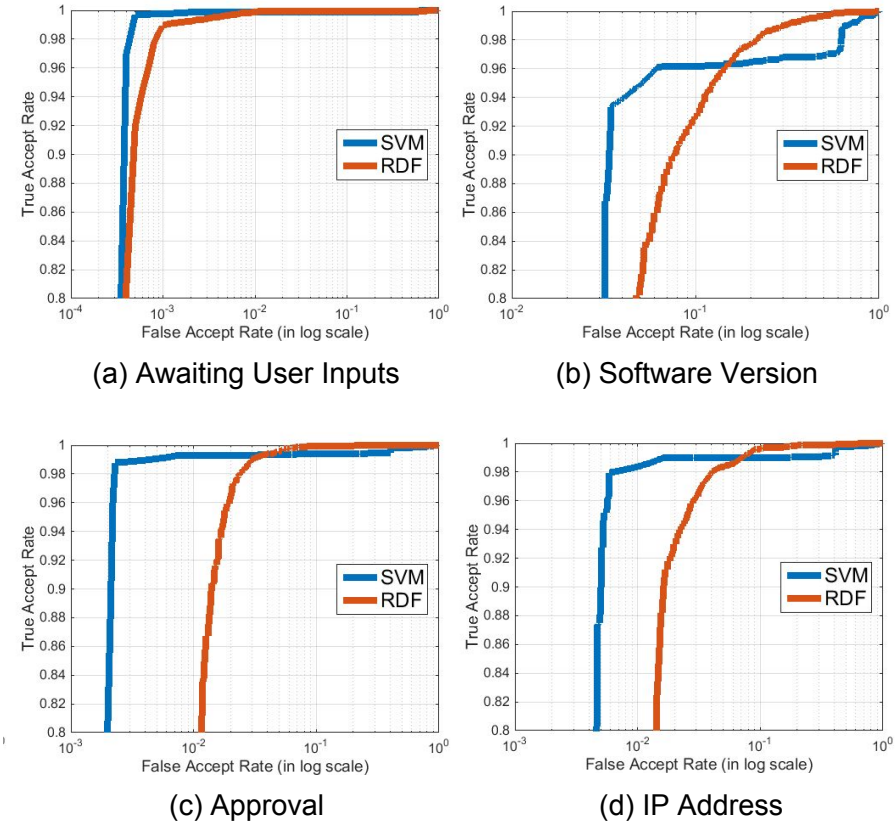

Fig. 6 ROC for SVM and RDF based preemtive model to illustrate their performance for different information needs: (a) Awaiting User Inputs, (b)Software Version, (c) Approval and (d) IP Address. The y-axis is cut at 0.8 to zoom in the point of bending for ROC curves.

obtained over five-times repeated random subsampling for all the classifiers are tabulated in Table 8. ROC is presented in Fig. 6 for SVM and random decision forest classifier as they perform better for all the four models. The major observations drawn from the results are as follows:

1. The proposed SVM classifier provides the best overall classification accuracy in the range of 95–99% for all the information needs. SVM is kind of expected to perform best with optimal parameter values since it is regarded as one of the best classifier in the literature for text classification tasks [22][4][20][37][45][65][77]. It can be observed that both precision and recall of the classifier are high suggesting that the classifier is not biased towards any particular class. It is possible because random under-sampling of majority class is performed at the time of training to handle imbalanced class problem.
2. It can be observed that an ensemble learning based classifier such as Random Decision Forest (RDF) performs comparable to SVM. Thus, SVM is not

Table 9 Categories in tactical user input requests with total comments in each class, percent of total comments, and example keywords.

Class	#Comments	% of comments	Example keywords
Temporize	6272	8.11%	in progress, working, will do it
Invalid	645	0.83%	No alphanumeric character
Will Transfer	482	0.62%	transfer to, assign to
Contact Me	21081	27.26%	ping me when free, call me @
Done So Check	4414	5.70%	Installed, Completed, Check

a strict choice for choosing the classifier of the preemptive system. SVM performs a kernel trick to project the feature space into a suitable higher dimensional space where linear classification is possible, while RDF combines the classification results of multiple individual classifiers making the classification decision robust. The logistic function of the regression classifier tries to fit a linear boundary in the provided feature space, thus, can lead to an approximate classification. Hence, logistic regression performs poorly when compared with SVM and RDF. Since a sparse feature representation is obtained from the bag-of-words model, models such as naive Bayes perform poorly in trying to fit a distribution for the data.

3. Fig. 6 shows the ROC curve plotted between the false accept rate (in log scale) and true accept rates, comparing the performance of SVM and RDF classifiers across all information needs. The ROC curve shows the trade off between sensitivity (also called recall) and specificity, providing the number of true detects for a given number of fall-outs. For all the information needs, it can be observed that SVM performs better than RDF by correctly detecting more than 95% of the test cases.
4. It is to be noted that the test data is an unseen data for the classifier. Thus, the performance of the classifier, as shown using the test data, is equivalent to the performance of the classifier as deployed in a real-time environment.

We have made the trained preemptive model and code publicly available [27] and they can be used by other researchers in their experiments.

8.4 Detection Model

Set of rules is derived iteratively for each of the five categories using the approach suggested in Section 6. For example, rule for category transfer is comment description should be like *transfer to* or *assign to*. Example keywords in generating rules for each category are shown in Table 9 and the complete set of rules for reference are made publicly available [27]. The given data set, that is, $77,333$ analyst comments corresponding to $96,756$ closed tickets for subcategory install are classified using the designed rule based classifier. We perform stemming, case folding of comments to ensure that matching is case insensitive and remove special characters. The total number of data points classified to each of the categories is summarized in Table 9. Around 42.52% of the total user input requests are classified to one of the five listed

Table 10 Real example comments for each category of tactical user input requests.

Class	Example Comments
Temporize	"We are trying to find the solution for the problem.We shall get back you soon." "Please provide sometime it will be done asap" "Will check and update the status."
Invalid	"...", "-", " "
Will Transfer	"Will Assigned to L1 Team. They will reach you shortly." "Transferring to concerned person." "This request is not under my scope of work. I will contact admin and transfer it to correct analyst."
Contact Me	"Please ping me when you are available." "Please Ping/Call me once you are at your desk and free so that we can work on your request." "You seem offline. Please contact me once you are available."
Done So Check	"The requested software has been installed. Please check and close the request." "Please check and update." "It has been done. Kindly check it."

categories. Managers expressed that while it is very useful to know comments from Contact Me category, it needs to be tackled differently as compared to other tactical categories. This is because they believe that while no direct information is asked in such comments, the analyst asked user to contact them. Therefore, there is high possibility that the analyst asked for inputs in follow up communication with the user over phone or chat. Hence, whether it is truly tactical or not also depends on the reason for asking user to contact them which is not captured in the comment therefore, cannot be concluded as clear case of tactical. As a result, comments from other four categories constituting around 15.27% of the total user input requests are considered as tactical. The most frequent tactical category is *temporize* constituting 8.11% of the total of user input requests. Interestingly, 645 input requests consist of non-alphanumeric characters such as dash, periods, and NULL. Table 10 presents some of the comments from the IT support system which are classified in the presented classes using the proposed detection model.

For evaluation, we requested two managers with experience (as manager in the same organization) of 3–5 years to independently and randomly pick around 100 comments each from the classified ones. They are requested to make sure that the sample contains comments from all the five categories shown in Table 10. They manually inspected the sampled comments and indicated if the comment is wrongly classified to a category. In all cases the managers agreed that the comment classified to a category indeed belongs to the same. Both the managers mentioned that it is really useful to have categories within tactical because each category may need to be tackled differently. Though the completeness is not guaranteed with this evaluation, the detection model precisely classifies comments to categories of tactical requests which can be handled accordingly.

Table 11 Destination state transition analysis for different types of user input requests. Most frequent destination state for each type of user input request is highlighted in bold.

Type of user input request / dest. state	User Update			No Update		
	User Input Received	**Attach doc**	**Closed**	**Resolved**	**AUI - Autoclosure**	**Transfer**
1. Real user input request (44439)	**18224 (41.0%)**	4340 (9.8%)	7429 (16.7%)	5678 (12.8%)	4432 (10.0%)	2222 (5.0%)
2. ContactMe (21081)	5469 (26.0%)	186 (0.9%)	3093 (14.7%)	**9202 (43.6%)**	1143 (5.4%)	1191 (5.6%)
3. Done So Check (4414)	941 (21.3%)	58 (1.3%)	**1517 (34.4%)**	**1466 (33.2%)**	226 (5.1%)	97 (2.2%)
4. Invalid (645)	160 (24.8%)	4 (0.6%)	29 (4.5%)	**362 (56.1%)**	8 (1.2%)	62 (9.6%)
5. Temporize (6272)	1873 (29.9%)	84 (1.3%)	514 (8.2%)	**2569 (41.0%)**	289 (4.6%)	715 (11.4%)
6. Will Transfer (482)	46 (9.5%)	4 (0.8%)	27 (5.6%)	21 (4.4%)	12 (2.5%)	**350 (72.6%)**

9 Destination State Analysis

We analyze the comments classified to one of the tactical categories and the ones not classified to any tactical class (referred to as real user input request) for the destination state. Table 11 presents transition of different user input request types to most frequent destination states. We test if there exists relationship between user input request type and destination state using chi-square test for independence because both are categorical variables and every cell has expected value more than 5. The p-value for the significance test is too low to be computed (less that 0.01) thus, the two variables are significantly related to each other. From Table 11, we make the following observations:

- Most frequent destination state (highlighted with bold) for tactical categories (2–5 in Table 11) is from No Update that is, Resolved, AUI-Autoclosure, and Transfer. However, most frequent destination for real user input requests is from User Update. This validates our conjecture that if no update from user for a user input request then it is more likely to be tactical and if user provides any update then it is more likely to be a real user input request.
- For real user input requests, user inputs are received for majority (around 41%) of the cases. In some cases (around 17%), tickets are closed by users without getting resolved.
- For Contact Me user input requests, either the user gave some inputs (as destination state is User Input Received for 26.0% comments) or the ticket was Resolved (for 43.6% cases) based on interaction between the user and the analyst outside ticketing system (not recorded in database).
- Done So Check type user input requests have Closed (34.4%) and Resolved (33.2%) as most frequent destination state. Closed indicates that the user was satisfied with the resolution hence closed. Many a times, user confirms

with a comment in response to this input request, hence, User Input Received is also quite frequent destination state. Apart from this, Resolved is frequent state indicating that the user did not confirm the resolution and analyst marked it as Resolved in sometime.
- For Invalid user input requests which are clear case of tactical input requests, the most frequent destination state is Resolved (around 56.1%). In cases where the destination state is User Input Received, the input from user is an expression of displeasure.
- For Temporize user input requests, maximum transitions are to state Resolved (for around 41.0% times), that is, analyst actually needed no information and misused the label. For instances with the destination state as User Input Received, users mostly clarify with analysts the information they are supposed to provide.
- Will Transfer user input requests are often followed by Transfer (for 72.6% times) of ticket with few exceptions.

The p-value (less than 0.01) and above observations validate our conjecture that destination state gives an indication about the type of user input requests. Therefore, manager can leverage the transition pattern (output of process mining) to decide if there is a need to reduce real or tactical or both type of user input requests.

10 Discussion

Based on organization size, culture and needs, the information from the proposed preemption and detection model can be leveraged in different ways.

10.1 Applications of Preemptive Model

Some of the applications of the proposed preemptive model are as follows:

- *Recommendation System:* The user is preempted with the potential information needs. However, it is up to the user to choose to provide the suggested information or not. If user provides the information on preemption, it can reduce delay due to later user input requests by analyst.
- *Mandatory System:* The user is not allowed to submit the ticket without providing all the preempted information. This guarantees reduction in user input requests because it is not left on the user to choose if (s)he wants to act on the preempted information. However, such a strict system makes it difficult for a user to report an issue if some of the preempted information is not available in hand. Also for such a system it is crucial to achieve very high precision.
- *Feedback for Template Improvement:* If the system preempts specific information needs more often, it is likely that the field to collect that information is either missing or not clear in the initial ticket template. Therefore, the

team can use this information to upgrade the template for more efficient ticket reporting.

10.2 Applications of Detection Model

Detection of tactical user input requests can be utilized in the following ways:

- *Notification System:* The manager is notified if the system detects that an analyst is trying to make a tactical user input request. The manager can look at the comment and if it is confirmed to be tactical, suitable actions (such as reassign the ticket) are taken based on the organizational policies. However, it is difficult for a manager to manually inspect all such suspicious cases for large organizations.
- *Logging System:* No immediate action is taken for the detected tactical user input requests, rather a log is maintained along with the analysts' information. It is analyzed by manager on a monthly or quarterly basis to understand if practice of making tactical user input requests is specific to some individuals or spans to a majority of analysts, and thus, to take appropriate actions. In the first case, analysts who make such requests more often are handled at individual level. In the other case, the information can be used by service level management to better estimate feasible service level resolution time and negotiate with clients accordingly.
- *Blocking System:* If the system detects a tactical user input request, it is blocked and not sent to the user. It ensures that no such requests degrade user experience. However, if a request is wrongly detected as tactical, the analyst is forced to paraphrase the comment such that it clearly seeks information.

Reduction in ticket resolution time depends on the application of a preemptive and detection model thus, we cannot estimate effective reduction in resolution time. However, given the observation that user input requests cause user experienced resolution to be much higher than measured service resolution time, reduction in user input requests will definitely lead to significant reduction in resolution time.

In case of the preemptive model, we achieve very high accuracy using SVM and RDF but it is not overfitting because the test data is different from the training data. This performance is achieved after fine tuning the parameters to optimal values for given data otherwise for some parameter values, the performance was no better than naive Bayes and logistic regression.

It is possible that an analyst seeks information which is not really required to resolve a ticket. However, it looks like a real user input request because in the given solution only non-information seeking user input requests are detected as tactical. Such cases go undetected with the given solution.

11 Threats to Validity

The threats for the presented work are as follows:

- *Threats to external validity:* External validity is concerned with the generalizability of the results to other settings [62][75]. Following are the threats to external validity for each element of the approach:
  - *Process mining of Ticket Data:* We analyze the transition pattern for awaiting user inputs and conjecture that there are both real and tactical user input requests. Tactical user input requests are observed in Volvo IT organization ticket data [74] and the data for the large global IT company investigated as part of this case study. However, tactical user input requests can be very rare or not present in other IT support ticket data for other organizations with different characteristics (such as small sized, less work load, and lenient service level resolution time limit).
  - *Survey of Users and Analysts:* We surveyed participants, that is, users from diverse project types and analysts working on different category of tickets. Nevertheless, they are part of the same organization using similar processes and guidelines. Though having diverse participants from the same organization enables to explore experiences from several perspectives, organizational culture may create a bias.
  - *Preemptive Model:* We evaluated performance of the preemption model for install subcategory within software category on real data for a large global IT company. Proposed model needs to be trained separately for each subcategory because different subcategories have different information requirements. The performance of the preemption model may vary with different ticket dataset. More so because performance of a classier strongly depends on the value of its input parameters, whose optimal choice heavily depends on the data being used. Therefore, the proposed preemptive model may preempt information needs in different contexts less accurately leading to smaller reduction in later user input requests.
  - *Detection Model:* The detection model classifies user input requests to refined tactical classes. However, the list of classes is not exhaustive and can vary with the organization.
- *Threats to conclusion validity:* Conclusion validity refers to whether the conclusions reached in the study are correct [62][75]. Threats to conclusion validity of the presented work are as follows:
  - *Preemptive Model:* Evaluation of the preemptive model is done on the test data but not in production. Since test data is unseen, we believe that the performance is close to reality and the model efficiently reduces user input requests. However, it depends on the way the preemptive model is applied in practice. For instance, if applied as recommendation system then there will be a performance improvement only if users choose to provide the preempted information.

  - *Detection Model:* While the detection model has high precision, it is difficult to guarantee high recall because it is possible that some infrequent tactical request classes are missed out.

- *Threats to construct validity:* Construct validity refers to the degree to which the factors under consideration in the designed experiment simulate the real conditions of their use [62][75]. While IT information system is the recommended communication channel between analysts and users, there can be communication over chat and telephone. The data for the communication over those channels is not accessible for analysis because of confidentiality and privacy reasons.
  We distinguish between analysts requesting information outside the ITIS and users proving the information outside the ITIS. The former is unlikely to happen as the SLA clock would not be affected. The latter would not affect the detection model, as the detection model analyzes the comments made when the analysts mark a ticket as Awaiting User Inputs. However, the validity of preemption model might have been affected as follows:
  - *Preemption Model:* Preemption model inherently reflects the data it has been trained upon and since no information is available about the communication outside ticket tracking system, the preemptive model might not adequately reflect the information needs expressed in such communication.

12 Conclusions and Future Work

We analyzed ticket data to capture the process reality more specifically user input requests made during ticket resolution life cycle by applying process mining. Also we studied the impact of user input requests on overall user experienced resolution time. There is a need to ensure that the information required for ticket resolution is collected from the user upfront thus, reducing real user input requests. However, users do not have a clear idea on what information will be required for resolving a specific ticket. Therefore, an SVM classifier based preemptive model was learnt to preempt users with the need for additional information during the time of ticket submission. Also we noticed non-information seeking, tactical user input requests for the sake of service level compliance. The rule based detection model identifies such input requests, thus can be discouraged. Performance of the proposed preemptive model and detection model on the real world data for a large global IT company shows the effectiveness of our solution approach in reducing the number of user input requests in tickets' life cycle.

In the existing approach, every information seeking user input request is considered as real irrespective of whether it is really required to resolve the ticket or not. In the future we plan to extend the detection model to also identify the cases where unnecessary information is asked, which is another way of making tactical requests. It requires further investigation of user input

requests and understanding of information actually being used for resolving the ticket.

Acknowledgements The work presented in this paper is supported by Prime Minister's Fellowship, SERB, CII, and Infosys Limited. The authors are thankful to the participants of both the surveys and Charlotte Ramon, an intern at Infosys Ltd. for help with conducting the survey. Thanks to Anush Sankaran for help with the preemptive model. We thank Prof. Tom Mens for his feedback on the early version of this manuscript. We acknowledge Prof. Pankaj Jalote, the PhD adviser of first author, and Anjaneyulu Pasala, the industry mentor of first author for the valuable feedback.

References

1. Rob Addy. *Effective IT service management: to ITIL and beyond!* Springer-Verlag New York, Inc., 2007.
2. Burcu Akman and Onur Demirors. Applicability of process discovery algorithms for software organizations. In *35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 195–202. IEEE, 2009.
3. Jeff Anderson, Saeed Salem, and Hyunsook Do. Striving for failure: an industrial case study about test failure prediction. In *37th International Conference on Software Engineering*, volume 2, pages 49–58. IEEE, 2015.
4. John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *28th International Conference on Software Engineering*, pages 361–370. ACM, 2006.
5. Alain April, Jane Huffman Hayes, Alain Abran, and Reiner Dumke. Software maintenance maturity model (SMmm): the software maintenance process model. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(3):197–223, 2005.
6. Gilad Barash, Claudio Bartolini, and Liya Wu. Measuring and improving the performance of an IT support organization in managing service incidents. In *International Workshop on Business-Driven IT Management*, pages 11–18. IEEE, 2007.
7. Claudio Bartolini, Cesare Stefanelli, and Mauro Tortonesi. Business-impact analysis and simulation of critical incidents in IT service management. In *International Symposium on Integrated Network Management*, pages 9–16. IEEE, 2009.
8. Christian Bartsch, Marco Mevius, and Andreas Oberweis. Simulation of IT service processes with Petri-nets. In *International Conference on Service-Oriented Computing*, pages 53–65. Springer, 2008.
9. Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. Automating extract class refactoring: an improved method and its evaluation. *Empirical Software Engineering*, 19(6):1617–1664, 2014.
10. Andrew Begel, Thomas Zimmermann, Yit Phang Khoo, and Gina D. Venolia. Discovering and exploiting relationships in software repositories, September 8 2015. US Patent 9,129,038.
11. Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *International Symposium on Foundations of Software Engineering*, pages 308–318. ACM, 2008.
12. Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128, 2006.
13. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
14. Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310. ACM, 2010.
15. João Caldeira and Fernando Brito e Abreu. Influential factors on incident management: Lessons learned from a large sample of products in operation. In *International Conference on Product Focused Software Process Improvement*, pages 330–344. Springer, 2008.
16. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

17. Patricia S. Crowther and Robert J. Cox. A method for optimal division of data sets for use in neural networks. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 1–7. Springer, 2005.
18. Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors. *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, volume 99 of *Lecture Notes in Business Information Processing*. Springer, 2012.
19. Jochen De Weerdt, Seppe Vanden Broucke, Jan Vanthienen, and Bart Baesens. Leveraging process discovery with trace clustering and text mining for intelligent analysis of incident management processes. In *Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.
20. Karim O. Elish and Mahmoud O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.
21. Diogo R. Ferreira and Miguel Mira Da Silva. Using process mining for ITIL assessment: a case study with incident management. In *13th Annual UKAIS Conference*, pages 1–16.
22. Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. Anger and its direction in collaborative software development. *ICSE NIER*, pages 11–14.
23. Vahid Garousi, Ebru Göçmen Ergezer, and Kadir Herkiloğlu. Usage, usefulness and quality of defect reports: an industrial case study. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, number 39, pages 1–6. ACM, 2016.
24. Christian W. Günther and Anne Rozinat. Disco: Discover your processes. *BPM (Demos)*, 940:40–44, 2012.
25. Christian W. Günther and Wil van der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *Business Process Management*, pages 328–343. Springer, 2007.
26. Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *32nd International Conference on Software Engineering*, volume 1, pages 495–504. IEEE, 2010.
27. Monika Gupta. Artifacts for ITIS ticket analysis. `https://github.com/Mining-multiple-repos-data/TicketExperimentalDataset`. Accessed: June 1, 2017.
28. Monika Gupta and Ashish Sureka. Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies. In *7th India Software Engineering Conference*, pages 1–10. ACM, 2014.
29. Monika Gupta, Ashish Sureka, and Srinivas Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In *11th Working Conference on Mining Software Repositories*, pages 122–131. ACM, 2014.
30. Frank E. Harrell. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Springer Science & Business Media, 2013.
31. Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.
32. James D. Herbsleb and Eiji Kuwana. Preserving knowledge in design projects: What designers need to know. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 7–14. ACM, 1993.
33. Tin Kam Ho. The random subspace method for constructing decision forests. *Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
34. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. 2003.
35. Michael Hüttermann. *DevOps for developers*. Apress, 2012.
36. Nathalie Japkowicz. The class imbalance problem: Significance and strategies. In *International Conference on Artificial Intelligence*, 2000.
37. Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning*, pages 137–142. Springer, 1998.
38. Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
39. Chang Jae Kang, Young Sik Kang, Yeong Shin Lee, Seonkyu Noh, Hyeong Cheol Kim, Woo Cheol Lim, Juhee Kim, and Regina Hong. Process mining-based understanding and

analysis of Volvo IT's incident and problem management processes. In *BPIC@ BPM*, 2013.
40. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Activity mining for discovering software process models. *Software Engineering, Fachtagung des GI-Fachbereichs Softwaretechnik*, 79:175–180, 2006.
41. Patrick Knab, Martin Pinzger, and Harald C. Gall. Visual patterns in issue tracking data. In *New Modeling Concepts for Todays Software Processes*, pages 222–233. Springer, 2010.
42. Andrew J. Ko, Brad A. Myers, and Duen Horng Chau. A linguistic analysis of how people describe software problems. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134. IEEE, 2006.
43. Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.
44. Ta Hsin Li, Rong Liu, Noi Sukaviriya, Ying Li, Jeaha Yang, Michael Sandin, and Juhnyoung Lee. Incident ticket analytics for IT application management services. In *International Conference on Services Computing*, pages 568–574. IEEE, 2014.
45. Ana Rocío Cárdenas Maita, Lucas Correa Martins, Carlos Ramón López Paz, Sarajane Marques Peres, and Marcelo Fantinato. Process mining through artificial neural networks and support vector machines: a systematic literature review. *Business Process Management Journal*, 21(6):1391–1415, 2015.
46. Alan Moraes, Eduardo Silva, Cleyton da Trindade, Yuri Barbosa, and Silvio Meira. Recommending experts using communication history. In *International Workshop on Recommendation Systems for Software Engineering*, pages 41–45. ACM, 2010.
47. Denilson Cursino Oliveira and Raimir Holanda Filho. A time and financial loss estimation using a highly parallel scheduling model for IT change management. In *International Symposium on Integrated Network Management-Workshops*, pages 1–9. IEEE, 2009.
48. Girish Keshav Palshikar, Mohammed Mudassar, Harrick M. Vin, and Maitreya Natu. Streamlining service levels for IT infrastructure support. In *International Conference on Data Mining Workshops*, pages 309–316, 2012.
49. Girish Keshav Palshikar, Harrick M. Vin, Mohammed Mudassar, and Maitreya Natu. Domain-driven data mining for IT infrastructure support. In *International Conference on Data Mining Workshops*, pages 959–966, 2010.
50. Zbigniew Paszkiewicz and Willy Picard. Analysis of the volvo IT incident and problem handling processes using process mining and social network analysis. In *BPIC@ BPM*, 2013.
51. Shaun Phillips, Guenther Ruhe, and Jonathan Sillito. Information needs for integration decisions in the release process of large-scale parallel development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1371–1380. ACM, 2012.
52. Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th working conference on mining software repositories*, pages 348–351. ACM, 2014.
53. Wouter Poncin, Alexander Serebrenik, and Mark G. J. van den Brand. Mining student capstone projects with FRASR and ProM. In *International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, pages 87–96. ACM, 2011.
54. Wouter Poncin, Alexander Serebrenik, and Mark G. J. van den Brand. Process mining software repositories. In *European Conference on Software Maintenance and Reengineering*, pages 5–14, 2011.
55. Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
56. Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. *Recommendation systems in software engineering*. Springer, 2014.
57. Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, pages 255–265. IEEE Press, 2012.
58. Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

59. Jana Samalikova, Rob J Kusters, Jos JM Trienekens, and AJMM Weijters. Process mining support for capability maturity model integration-based software process assessment, in principle and in practice. *Journal of Software: Evolution and Process*, 26(7):714–728, 2014.
60. Adrian Schröter, Irwin Kwan, Lucas D Panjer, and Daniela Damian. Chat to succeed. In *International Workshop on Recommendation Systems for Software Engineering*, pages 43–44. ACM, 2008.
61. Sam Scott and Stan Matwin. Feature engineering for text classification. In *International Conference on Machine Learning*, volume 99, pages 379–388, 1999.
62. Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. *Guide to Advanced Empirical Software Engineering*, volume 93. Springer, 2008.
63. Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34. ACM, 2006.
64. Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.
65. Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *32nd International Conference on Software Engineering-Volume 1*, pages 45–54. ACM, 2010.
66. Wikan Sunindyo, Thomas Moser, Dietmar Winkler, and Deepak Dhungana. Improving open source software process quality based on defect data mining. In *Software Quality. Process Automation in Software Development*, pages 84–102. Springer, 2012.
67. Ashish Sureka and Pankaj Jalote. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific Software Engineering Conference*, pages 366–374. IEEE, 2010.
68. Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*, pages 321–332. ACM, 2016.
69. Tapanee Treeratanaporn. Information Technology Service Management (ITSM) in education. *Walailak Journal of Science and Technology*, 12(9):739–747, 2015.
70. Peter Van den Spiegel, Leen Dieltjens, and Liese Blevi. Applied process mining techniques for incident and problem management. In *BPIC@ BPM*, 2013.
71. Wil van der Aalst. Process mining - discovery, conformance and enhancement of business processes. *Springer*, 2011.
72. Wil van der Aalst, Hajo A. Reijers, Ton Weijters, Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, Minseok Song, and Eric Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.
73. Jan Martijn E. M. Van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *International Conference on Applications and Theory of Petri Nets*, pages 368–387. Springer, 2008.
74. Boudewijn F. van Dongen, Barbara Weber, Diogo R. Ferreira, and Jochen De Weerdt. Business process intelligence challenge, 2013.
75. Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.
76. Qing-Song Xu and Yi-Zeng Liang. Monte Carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, 56(1):1–11, 2001.
77. Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. Developer prioritization in bug repositories. In *34th International Conference on Software Engineering*, pages 25–35. IEEE, 2012.
78. Nor Shahida Mohamad Yusop, John Grundy, and Rajesh Vasa. Reporting usability defects: do reporters report what software developers need? In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, number 38, pages 1–10.

79. Cheng Zhang, Juyuan Yang, Yi Zhang, Jing Fan, Xin Zhang, Jianjun Zhao, and Peizhao Ou. Automatic parameter recommendation for practical API usage. In *34th International Conference on Software Engineering*, pages 826–836. IEEE Press, 2012.
80. Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.