Software process evaluation: A machine learning approach

Ning CHEN
Nanyang Technological University

Steven C. H. HOI
Singapore Management University, chhoi@smu.edu.sg

Xiaokui XIAO
Nanyang Technological University

Software Process Evaluation: A Machine Learning Approach

Ning Chen, Steven C. H. Hoi, Xiaokui Xiao
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{nchen1,chhoi,xkxiao}@ntu.edu.sg

***Abstract*—Software process evaluation is essential to improve software development and the quality of software products in an organization. Conventional approaches based on manual qualitative evaluations (e.g., artifacts inspection) are deficient in the sense that (i) they are time-consuming, (ii) they suffer from the authority constraints, and (iii) they are often subjective. To overcome these limitations, this paper presents a novel semi-automated approach to software process evaluation using machine learning techniques. In particular, we formulate the problem as a sequence classification task, which is solved by applying machine learning algorithms. Based on the framework, we define a new quantitative indicator to objectively evaluate the quality and performance of a software process. To validate the efficacy of our approach, we apply it to evaluate the defect management process performed in four real industrial software projects. Our empirical results show that our approach is effective and promising in providing an objective and quantitative measurement for software process evaluation.**

***Keywords*-software process; defect management process; sequence classification; machine learning**

I. Introduction

A software process is a coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product [1]. Most software organizations adopt certain software process methodologies to access, support, and improve their development activities because empirical studies have shown that the quality of software processes are directly related to the productivity of an organization and the quality of software products [2], [3]. One key challenge faced by many organizations is how to evaluate a software process performed in a specified scope (ranging from a project to a department or an entire organization). This is termed as the "software process evaluation" task, which is usually done by conducting questionnaire-based assessment, or interviewing project team members, or randomly checking artifacts generated by former software projects (e.g., checking some source codes stored in software configuration management systems or checking some bug reports in test management systems, etc.) [4]. The conventional methods are based on subjective and qualitative investigations and have been adopted by the Capability Maturity Model Integration (CMMI) assessment [5].

Despite being widely used, the existing software process evaluation methods often suffer from numerous limitations. First, they require manual evaluation, which is rather usually *time-consuming*, especially for a software development organization with a large number of projects. For example, it is inefficient to manually examine all the artifacts produced by former projects, as the number of such artifacts could be enormous. Although one may circumvent this issue by examining only a random sample set of the artifacts, the inaccuracy incurred by sampling may cause misjudgments. Second, the existing methods suffer from the *authority constraints*. For example, if an external audit agency or a consultation agency is asked to evaluate the software processes of an organization, it may not be allowed to access every artifact directly due to privacy or security concerns. This may lead to inadequate checking of artifacts, which result in inaccurate evaluation results. Third, the existing approaches are often based on *subjective evaluation*, and hence, they suffer from biased evaluation results due to human factors in the evaluation process. Besides, the conventional methods usually have a high requirement of experienced evaluation experts to understand the specific software processes and design various questions for conducting a qualitative evaluation task.

To address the above limitations, this paper presents a novel semi-automated approach to software process evaluation using machine learning techniques. Since software processes can be broadly referred to a variety of concepts,we restrict the discussion of a software process as a systematic approach to the accomplishment of some software development tasks in this paper. The key idea is to model the series of activities or artifacts' states in a software process as sequential instances, and then formulate the evaluation task as a binary classification of the sequential instances into either "normal" or "abnormal". Based on this framework, we present a new quantitative measure, referred to as the "*process execution qualification rate*", as an objective evaluation of the quality and performance of a software process.

Our approach aims to overcome the limitations of conventional subjective evaluation methods. First, our approach is semi-automated, which (i) is more efficient than the conventional manual methods that requires a significant amount of human efforts, and (ii) alleviates the need of highly experienced evaluation experts on process evaluation. Second, our approach requires only the process execution history data and does not access artifacts directly, which alleviates the authority constraints as the process execution history data usually do not contain private contents. Third, by adopting the process execution qualification rate, our approach avoids making subjective judgements. To validate the efficacy of our approach, we apply our technique to evaluate the defect management process performed in four projects from a large software development center of a commercial bank in China. Our empirical results show that the proposed approach is effective.

In summary,this paper makes the following contributions:

- We propose a novel machine learning approach that may help practitioners to evaluate their software processes.
- A new quantitative indicator, referred to as process execution qualification rate, is proposed to evaluate the quality and performance of software processes objectively.
- We compare and explore different kinds of sequence classification algorithms for solving this problem.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents the problem statement. Section IV introduces the proposed machine learning approach to software process evaluation tasks. Section V presents the experimental results and discusses the limitations of validation. Finally, Section VI concludes this paper.

II. Related Work

Our work is closely related to previous works on software process research. In particular, there exist a variety of software process methodologies and models which contain a set of guidelines and practices for assessing and improving the general software process capability of large and small software development organizations [5]–[9]. The previous works mainly focus on high-level, comprehensive and general frameworks for software process evaluation tasks, while our work aims to present a concrete method for an important subtask of software process evaluation, which could be integrated into those methodologies and models. Formerly, qualitative evaluation methods [10], such as questionnaire-based assessment, interviewing and artifacts studies, are widely adopted by some process models, e.g., the Capability Maturity Model Integration(CMMI) [5]. CMMI was proposed by Software Engineering Institute (SEI)[1] and holds a very important position in software process research. It defines a series of best practices to guide software process assessment and improvement in software development organizations. Practices cover topics that include techniques, management, support and more. This model has been broadly used in various industrial organizations all over the world, and turns out to be especially effective for large organizations. The Standard CMMI Appraisal Method for Process Improvement (SCAMPI) [11] defines a formal framework to provide evaluation results relative to CMMI models. All the traditional qualitative methods mentioned above are used by SCAMPI. However, these methods have the critical drawbacks described in Section I, which motivates the study of our work. In this paper, we propose a new quantitative machine learning approach which could be a strong complementarity for traditional software process evaluation methods. In software process validation, Cook et al. [12] proposed a method for detecting difference between a formal process model and a process execution. In their work, the correspondence between an execution event stream (produced by process executing) and a model event stream (induced by the formal process model) is measured using string distance metrics. Two metrics, referred to as Simple String Distance (SSD) and Nonlinear String Distance (NSD) were developed. Since only one model event stream is derived from the process model in their approach, it may lead to big mistakes for process validation results when (i) two or more formal and reasonable process models exist for a same software process; (ii) process models are changing rapidly without updating the model definition in the documents; (iii) there are a number of non-trivial process branches. Different from their work, our proposed machine learning approach can address these limitations. Moreover, we propose a novel quantitative indicator. Similarly, Moor et al. [13] presented an approach to software validation that employed conceptual graph theory to compare differences between defined process models and actual process models. In summary, our work differs from the aforementioned software validation approaches in that we aim to learn some dynamic prediction models from a collection of updated process executions by machine learning techniques.

In addition, our work is related to software process studies that mine event logs stored in some software repositories to uncover the software process models. For instance, Rubin et al. [14] gave an overview of applications of process mining techniques to discover software development processes. Samalikova et al. [15] made use of a heuristic mining algorithm to construct the "real" CCB (Change Control Board) process model from event logs stored in software configuration management systems. They compared this mined "real" process model with the "official" process model and gave feedbacks to the development team. Although the nature of data studied in their work is similar to ours, the techniques used are different. In their approach, the key technique is applying an off-the-shelf process mining algorithm offered by ProM framework [16] to construct an explicit, "actual" process model from event logs, while our approach adopts machine learning techniques to learn a model to classify process executions into either "normal" or "abnormal". The research purpose of Samalikova et al.'s work is to use the analysis results of mined objective process models to improve the software processes in the project teams. They did case studies in a large industrial company in the Netherlands. Different from their research purpose, we are aiming at developing a quantitative approach to software process evaluation tasks. Extensive experiments and case studies are conducted in our work to validate the effectiveness and efficiency of our proposed approach. In summary, research on software process mining mainly focuses on the construction of explicit and "real" graph-based representation of software process models

[1]http://www.sei.cmu.edu/

using process mining algorithms from event logs, while our work aims at learning dynamic prediction models from event logs to evaluate the quality and performance of a software process objectively by using a new quantitative indicator.

Last but not least, our work is in general also related to the emerging studies that apply data mining and machine learning techniques in the literature of software engineering [17]. Specifically, the first category of related work in this field applies sequence mining algorithms to support software engineering tasks. For example, El-Ramly et al. [18] formulated the problem of recovering user scenarios from user interaction-traces as an instance of the "sequential pattern mining" problem, where an interaction trace is initially represented as a sequence of screen IDs and then a mining algorithm named IPM2 was developed to mine the qualified patterns. Lo et al. [19] proposed an efficient algorithm to mine a closed set of software iterative patterns from program execution traces. In addition, another category of related work employs classification algorithms to support and improve software engineering activities. For example, Clenland-Huang et al. [20] proposed the none functional requirements (NFR) classifier, which is used to classify none functional requirements from both structured and un-structured documents. Anvik et al. [21] explored various classification algorithms in solving bug assignment tasks, where bug reports (text documents) are classified into different categories defined by the names of developers who are appropriate to resolve some specific types of reports. Compared with the previous studies in this area, our work is common in that we all apply machine learning and data mining techniques to solve software engineering problems, but differs in that we present a machine learning approach to address a different problem in software engineering.

III. Problem Statement

In general, *software process evaluation* aims to assess the quality of some software processes performed in some specified scope (ranging from a project to a department or an entire organization). Typically, each organization has *software process specifications*, which provide explicit representation and definition of different kinds of software processes using formalism models, such as Petri nets, finite state machines, or flow diagrams associated with detailed text descriptions. Different aspects of processes are addressed in the specifications, such as artifacts, states of the artifacts, activities and responsible persons. Different software organizations may adopt different representation methods for software process specifications, which usually can be found in the quality manuals of an organization.

In our approach, the first step towards software process evaluation is to obtain raw data recording the detailed activities or artifacts' state of a software process. Most software development organizations are equipped with commercial or open-source computer-aided software engineering (CASE) tools for the whole software development life cycle (SDLC). For instance, software configuration management systems are designed to monitor and control changes in the software development; requirement management systems are used to trace, manage and control user requirements, etc. As an important part of CASE tools, *software repositories* store the data generated by all activities via SDLC. In our work, we obtain raw data contain software process execution history extracted from software repositories.

In software repositories, a key element is *process execution*, which is a sequence of snapshots of the actions executed by different roles or artifacts' states of the observed software process. As mentioned in Section I, software processes can be broadly defined as various concepts, to simplify our discussion, we restrict the discussion of a software process as a collection of process executions in this work. Accordingly, the goal of a software process evaluation task in our study is to assess the quality of some collection of process executions specified in some particular scope. For example, in this study, our goal is to evaluate a software defect management process from some collection of process executions stored in defect management repositories. By using process executions, we intend to apply machine learning techniques together with a quantitative indicator to evaluate the quality and performance of a software process objectively.

IV. Approach

In this section, we first give an overview of the proposed machine learning approach to software process evaluation, and then discuss each step of our approach in detail.

A. Overview

Given a collection of process executions, the key idea of our machine learning approach is to employ supervised sequence classification techniques to classify each process execution into "normal" or "abnormal". Here "normal" means that a process execution is qualified or is able to meet the organization's standards/criteria. We also propose a novel quantitative indicator named *process execution qualification rate*, which is calculated to evaluate the quality and performance of the software process. Fig. 1 illustrates the proposed framework of our machine learning approach to software process evaluation.

In a nutshell, the proposed approach consists of four major steps. The first step extracts raw data stored in software repositories. The raw data that we are interested in contain process execution history (e.g., defect state change history). The second step preprocesses the data by transforming relevant data to a set of sequences. The third step builds different sequence classifiers through a set of labeled training data and conducts classifier evaluation. The last step applies the best sequence classifier to classify all the rest sequences into either "normal" or "abnormal"; as a result, the process execution qualification rate can be calculated to evaluate the quality and performance of the specified software process.

B. Collecting Data

As mentioned before, there are various kinds of data stored in software repositories. For example, we can find technical documents in software configuration management systems;

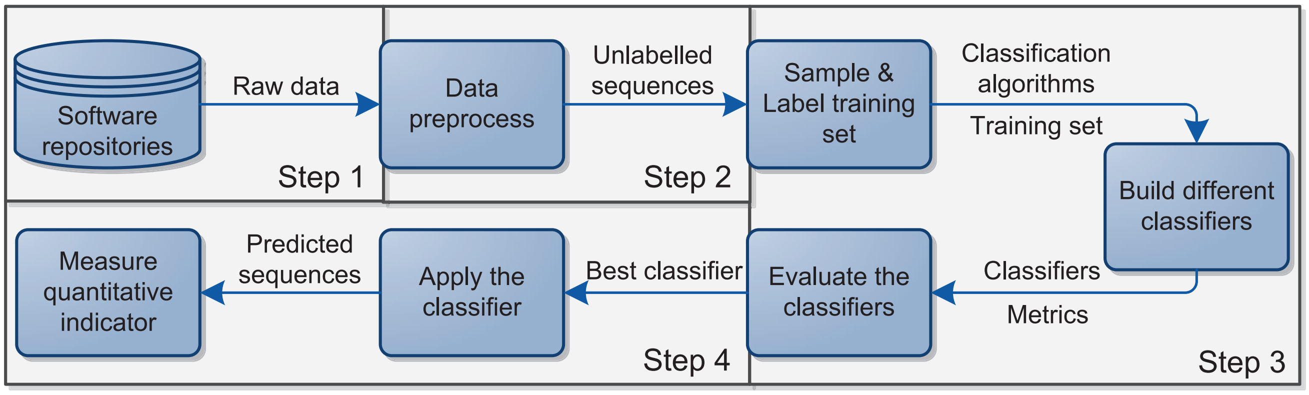


Fig. 1. The framework of the proposed machine learning approach to software process evaluation

bug reports can be extracted from test management systems; requirement change reports can be collected from requirement management systems, etc. We refer to these data that have not been processed as raw data, which can be either structured or non-structured, and usually can be exported through some functionality provided by CASE tools.

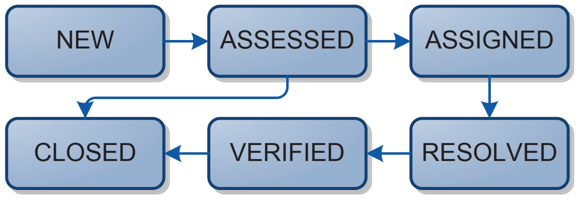
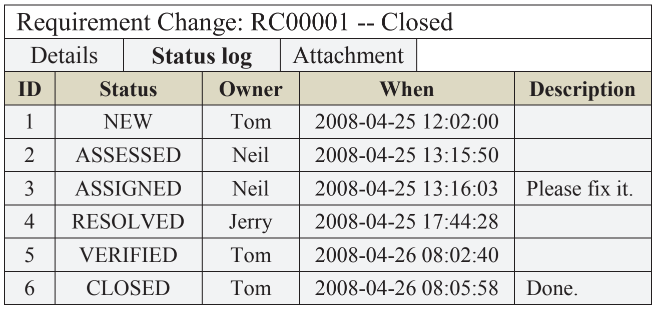


Fig. 2. Example of a simple requirement change process



Requirement Change: RC00001 -- Closed

Details | **Status log** | Attachment

ID	Status	Owner	When	Description
1	NEW	Tom	2008-04-25 12:02:00	
2	ASSESSED	Neil	2008-04-25 13:15:50	
3	ASSIGNED	Neil	2008-04-25 13:16:03	Please fix it.
4	RESOLVED	Jerry	2008-04-25 17:44:28	
5	VERIFIED	Tom	2008-04-26 08:02:40	
6	CLOSED	Tom	2008-04-26 08:05:58	Done.

Fig. 3. Example of some raw data from a software repository

The first step of our approach extracts the raw data from related software repositories according to the characteristics of the software process that we intend to evaluate. In this work, we are only interested in raw data that contain process execution history. The process execution history captures the evolutions of artifacts or activities of various tasks in the process. To better describe our approach in an intuitive way, Fig. 2 shows an example of a simple requirement change process we intend to evaluate. This process represents the state flow of requirement changes (RC); each node in the flow diagram represents a status of RC. To evaluate this process in a specified scope, in our approach, we should collect all the raw data needed from requirement management systems. Fig. 3 shows an instance of raw data (i.e., an RC) extracted from a certain requirement management system that contains process execution history (see the "Status Log" column). In general, for some particular software processes, we only need to extract raw data from one software repository. For some other types of processes, we may need to extract raw data from multiple different software repositories.

C. Data Preprocessing

This step preprocesses the collected raw data into well-structured format to facilitate subsequent machine learning tasks. Specifically, the idea is to convert every process execution from the raw data into a sequence-based instance. For ease of representation, we introduce an "alphabet" that consists of a set of symbolic values, each of which represents a status of some artifact or an action of some task in the software process. Such an alphabet comprises all the possible values/states that can appear in the process executions. Thus, each process execution can be represented as a sequence of symbolic values from the alphabet. Consider the example of the requirement change process shown in Fig. 2, we can determine an alphabet as "(N, S, A, R, V, C)", where each letter of "N, S, A, R, V, C" represents "NEW", "ASSESSED", "ASSIGNED", "RESOLVED", "VERIFIED", and "CLOSED", respectively.

TABLE I
EXAMPLE OF A SEQUENCE DATABASE

RC ID	sequence ID	sequences
1	S_1	$\langle N, S, A, C\rangle$
2	S_2	$\langle N, S, A, R, V, C\rangle$
...	...	...
N	S_N	...

Once the alphabet is chosen, the next is to convert each process execution from the raw data into a sequence of symbolic values. Consider the example in Fig. 3, the process execution history are contained in the "Status log" field. For this example, we first extract a sequence of statuses

⟨NEW, ASSESSED, ASSIGNED, RESOLVED, VERIFIED, CLOSED⟩ according to the temporal order, and then convert it to a sequence of symbolic values ⟨N, S, A, R, V, C⟩ by following the alphabet. After processing all the process executions, we obtain a sequence database which contains a set of (unlabeled) sequences S= $\{S_1, S_2 \ldots S_N\}$, where each sequence S_i is an ordered list of symbolic values from the alphabet. Table I shows an example of sequence database with N sequences.

D. Building Sequence Classifiers

The data preprocessing step generates a sequence database that consists of N unlabeled sequences whose class labels are unknown. Our goal is to learn some classification model by machine learning algorithms to automatically predict the class labels of these sequences. Sequence classification has been extensively studied in the literature of machine learning and data mining [22]–[24]. In general, we must address three key issues of a sequence classification task: (i) create a training set of labeled sequences, (ii) represent each sequence of varied length into a fixed-dimension vector, and (iii) choose a machine learning algorithm to build a classifier on the training data. We discuss our approach to addressing each of these issues in detail below.

1) Sampling and Labeling A Training Data Set: This task is to form a training data set by sampling a subset of unlabeled sequences from the entire database, and then manually label the sampled sequences. There are several issues in this task. First of all, we need to decide a class label set to label the sequences. Since we formulate this problem as a binary classification task, we have two unique class labels, {Normal, Abnormal}, where "Normal" implies that the process execution represented by this sequence is reasonable or meets the organization's standards, and "Abnormal" implies that the process execution represented by this sequence is irrational or violates the organization's standards and could cause potential negative impacts.

The second issue of this task is to choose an appropriate size of the training data set, denoted as N_{tr}, which is smaller than the database size N. If N_{tr} is too large, it would be time-consuming for performing the manual labeling task; on the other hand, if N_{tr} is too small, the training data may not be enough to build a good classifier. Typically, this size is determined empirically. We will discuss this issue further in our empirical study.

The third issue is to assign an appropriate class label to an unlabeled sequence manually. One common approach is to simply request an evaluation staff to examine if the process execution represented by the sequence has correctly followed the software process specifications given by the organization and adhered to the principles of the process methodology adopted by the organization. However, it is often not enough. One reason has to do with some ambiguous cases in the software process specifications. Another reason is that, software processes in an organization are usually changing rapidly; therefore, the given software process specifications may be outdated. Besides, typically in a large organization, some departments or project teams may not always adopt the standard processes of the organization due to various reasons [25]; these specific processes are usually not recorded in the organizational software process specifications. Due to the above reasons, sometimes it is not easy to assign a label clearly. For such cases, it would be necessary for the evaluation staff to consult and discuss with some experienced domain experts in the organization.

2) Feature Representation: Classical classification algorithms in machine learning and data mining often work on data represented in vector space, while the sequence examples in the database are not represented in vector space, in which they have varied lengths and the order concern of their elements. To apply the existing classification algorithms, we need to find some appropriate feature representation technique to map each of the sequence example into an example in some vector space of fixed dimensions. To this purpose, we adopt the well-known *k-grams* feature technique in our approach, which has been widely used in other domains [26]–[28].

Specifically, given a long sequence, a short sequence segment of any k consecutive symbols is called a k-gram, which is selected as a feature in order to keep the order of the elements in the sequence. After determining a set of k-grams, each sequence can be represented as a fixed-dimension vector of the presence and the absence of the k-grams (or as a vector of the frequencies of the k-grams appeared in the sequence). Let us illustrate the idea by the example in Fig. 2. The alphabet representing the statuses is "(N, S, A, R, V, C)", i.e., the number of unique symbols $m = 6$. Assuming that $k = 2$ is chosen for the k-grams model, we have $6^2 = 36$ unique k-grams. These k-grams include "{NN, NS,. . . , CV, CC}". Now consider a sequence S_i= ⟨N, S, A, R, V, C⟩. We slide a window of size $k = 2$ across the sequence to obtain the set of all the short sequences of length $k = 2$, i.e., {NS,SA,AR,RV,VC}; as a result, we get the feature vector corresponding to sequence S_i, as shown in Table II.

TABLE II
FEATURE VECTOR OF A SEQUENCE S_i= ⟨N, S, A, R, V, C⟩ BASED ON A 2-GRAM MAPPING METHOD

	NN	NS	...	SA	...	AR	...	RV	...	VC	...
S_i	0	1	0	1	0	1	0	1	0	1	0

3) Training Classifiers by Machine Learning Algorithms: After obtaining a training data set of N_{tr} examples in a vector space, the next step is to apply a machine learning algorithm to build classifiers on the training data. In our approach, we adopt three well-known and representative classification algorithms in machine learning, i.e., decision trees (C4.5) [29], Naive Bayes (NB) algorithm [30], and Support Vector Machines (SVM) [31], all of which have been widely used in a variety of real-world classification tasks. We will discuss the detailed comparison of these three classification algorithms in our experiments.

E. Evaluating Different Classifiers

After training a set of classifiers by different machine learning algorithms, the next step is to evaluate the performance of these classifiers in order to find the best classifier. To achieve this purpose, we first need to choose some performance metrics for comparisons. The first set of metrics include precision, recall, and F-measure, which are defined as follows:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP+FP}, \quad \text{Recall} = \frac{TP}{TP+FN} \\ \text{F}-\text{measure} &= \frac{2*\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}} \end{aligned}$$

where TP, FP, FN represent the numbers of true positives, false positives, and false negatives, respectively. In addition, we also adopt the well-known "Area Under ROC Curve" (AUC) [32] as another metric for evaluation. Finally, we also adopt the "Root Mean Square Error" (RMSE), a widely-used measure of the differences between values predicated and the actual truth values.

F. A New Quantitative Indicator

Based on the proposed classification framework, we propose a new quantitative indicator named as *process execution qualification rate* as an objective measure for software process evaluation, which is able to overcome the limitations of conventional evaluation methods.

Specifically, in the above classification scheme, we build a set of classifiers on a training data set, and choose the best classifier of the highest classification performance to label the remaining unlabeled sequences in the database. After obtaining the predicted class labels of all the sequences in the database, we define the *process execution qualification rate*, denoted as $P \in [0, 1]$, as follows:

$$P = \frac{\#predicted\ normal\ sequences}{total\#sequences} \times \frac{precision}{recall} \qquad (1)$$

The above definition indicates that P represents the percentage of actual "normal" sequences among all the sequences in the database. The larger the value of P, the better the quality and performance of a software process. This quantitative indicator provides an objective measure to evaluate the quality and performance of a software process.

Remark. In the above definition, it is important to note that both true "precision" and "recall" on the test set are unknown during the software process evaluation phase for a real-world application (unless we manually label all the test sequences). In order to calculate an estimated process execution qualification rate without manually labeling the test sequences, we adopt the precision and recall values on the training set as the estimated precision and recall values to calculate the value of P. In our experiments, we will evaluate the difference between the estimated qualification rate and the true qualification rate under different classification settings.

V. Experiments

To evaluate the effectiveness of the proposed approach, we conduct an extensive set of experiments by applying our technique to evaluate the defect management process performed in four real-life software projects. First, we briefly describe the profiles of these four projects and introduce the defect management process to be evaluated. We then describe the experimental setup followed by presenting our experimental results and discussions as well as some case studies in detail.

A. Experimental Testbed

We adopt an experimental testbed that consists of four software projects of a large software development center of a commercial bank in China. These four projects were developed by different project teams from the same department of the organization. The organization is assessed at CMMI level-2, which means this organization has a certain level in process management. Besides, we intend to evaluate the defect management process conducted in this department and in each individual project. Table III summarizes the characteristics of these four projects, where the last column shows the number of bugs/defects found in each project.

Fig. 4 shows the defect management process flow defined in the defect management process specification used in this organization. For simplicity, we only show the state flow of defect management process by different roles, and omit some detailed descriptions and explanations due to space limitation. As shown in this flow diagram, there are two kinds of nodes: "Status" node which represents the possible status of defect and the responsible role, and "Decision" node which represents the possible decisions that can affect the state change of a defect. The status of a defect starts from "New" and evolutes along with different decisions.

TABLE III
SUMMARY OF THE FOUR PROJECTS IN OUR EXPERIMENTS

Project ID	Description	No. of defects
1	Electronic Commercial Draft System	1019
2	Wealth Management System(Phase1)	478
3	Wealth Management System(Phase2)	665
4	Financial Leasing System(Phase2)	460

B. Experimental Setup

To perform the evaluation of the defect management process of the projects shown in Table III, we collect the raw data of the defect status change history from the repository of test management system adopted by this organization, which was based on the HP Quality Center (HQC) version 9.0. Fig. 5 shows an example of the raw data of the defect history log collected in our experiments, from which we can extract the sequence of statuses $\langle$NEW, OPEN, ASSIGNED, FIXED, CLOSED$\rangle$ to represent this particular defect process execution. Further, we generate an alphabet based on the process flow diagram as shown in Fig. 4, which consists of a total of 10

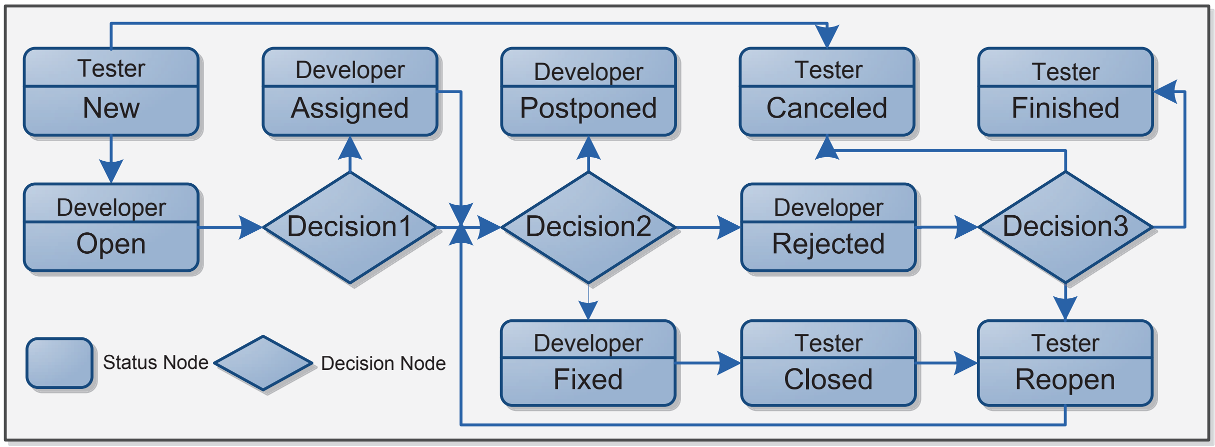


Fig. 4. The defect management process flow extracted from a software process specification(Detailed explanations were omitted).

Defect ID: 2722	Summary: Error		Project Name: Electronic Commercial Draft System	
History Log:				
Field name	**Date of update**	**Modificator**	**Old value**	**New value**
Defect status	2009-08-27 17:52:48		Fixed	Closed
Closed data	2009-08-27 17:52:48	Song		2009-08-27 17:52:48
Defect status	2009-08-27 09:32:11	Wang	Assigned	Fixed
Defect status	2009-08-26 18:15:30	Lin	Open	Assigned
Developer	2009-08-26 18:15:30	Lin		Wang
Defect status	2009-08-26 17:52:48	Song	New	Open
Developer	2009-08-26 17:52:48	Song		Lin

Fig. 5. Example of a defect history log from an industrial project in our empirical study.

symbols. Based on this alphabet, we then convert the sequence of statuses into a sequence of symbols. Further, we adopt the k-grams model (For simplicity, we choose k=2 in our experiments. We will discuss the problem of appropriate k values in Section V.G), as described in Section IV, to represent each sequence by a 100-dimensional vector. As a result, we form a sequence database of 2622 unlabeled examples represented in a 100-dimensional vector space. To facilitate the classification experiments, we collect the ground truth labels by manually labeling these unlabeled sequences into "normal" or "abnormal" based on the standards adopted by the organization (1985 sequences are labeled as "normal" and 637 sequences are labeled as "abnormal"), in which we had resolved some ambiguous cases by consulting the experts from this organization (For example, we label sequences contain "REOPEN" more than 2 times as "abnormal" in this context). In all the experiments, we take "normal" as positive class and "abnormal" as negative class.

Based on the above processed data and the ground truth labels, we conduct extensive experiments to evaluate the efficacy of our technique. In particular, we aim to answer the following questions: (1) What would be the best classification algorithm for sequence classification in our problem? (2) How does the amount of training data affect the classification performance in our task? (3) What is the accuracy of the estimated process execution qualification rate based on the proposed framework? Next we discuss each of our experiments to address these issues in detail.

TABLE IV
EVALUATION RESULTS

	Precision	Recall	F-measure	AUC	RMSE
C4.5	0.929	0.947	0.938	0.850	0.284
NB	0.929	0.911	0.920	0.945	0.291
SVM	0.976	0.996	0.986	0.961	0.143

C. Comparison of Classification Algorithms

The first experiment is to compare the performance of three different classification algorithms for our defect management process evaluation task, including decision tree, Naive Bayes (NB) classifier, and Support Vector Machine (SVM). We adopt the C4.5 implementation for decision tree, and employ a linear kernel for SVM with the penalty parameter $C = 10$. We adopt the standard 10-fold cross validation setting on the whole database of 2622 sequences to evaluate the performance of different classifiers. Table IV presents the experimental results of comparison with respect to the metrics of precision, recall, F-measure, AUC, and Root Mean Square Error (RMSE).

From the results in Table IV, we can see that among the three algorithms, SVM achieves the best performance for all the performance metrics, while C4.5 and NB perform diversely

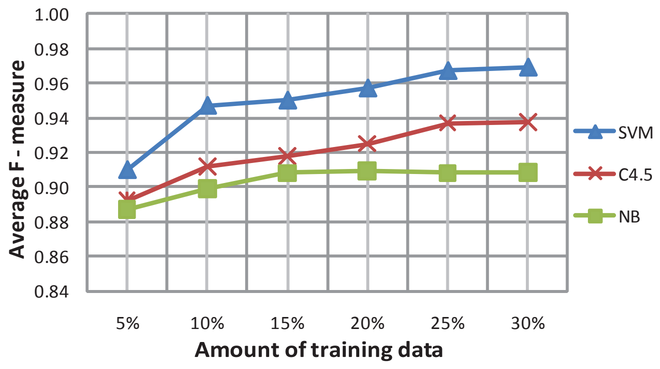


Fig. 6. Effect of training data size on F-measure

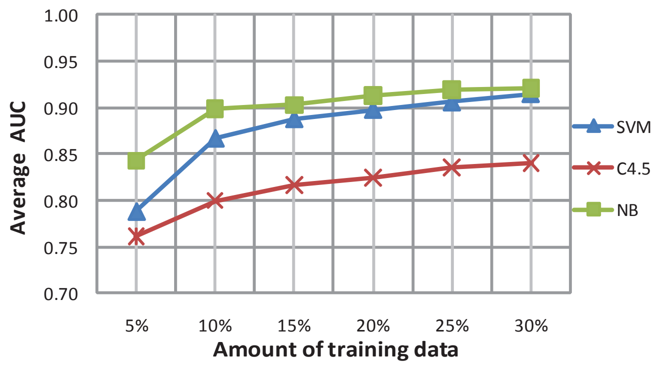


Fig. 7. Effect of training data size on AUC

under different metrics (C4.5 performs slightly better in terms of F-measure and RMSE, while NB performs better in terms of AUC score). By looking into the results achieved by SVM, we find that the F-measure and AUC values attained 98.6% and 96.1% respectively, which indicates that the proposed scheme is able to learn an effective classifier for the classification task. The very positive result is partly derived from the nature of the data: a large portion of the sequences are repetitive. We further informally investigate the classification result for unusual sequences (sequences that appear less than 5 times in the database), and find that the classification performance is also fairly good.

D. Classification Performance with Varying Amount of Training Data

The second experiment is to examine how the amount of training data affects the classification performance. To this purpose, for each experimental trial, we randomly sample 30% sequences from the database as the candidate training data, and the remaining 70% sequences as the test set; among the candidate training examples, we randomly choose a subset of training data to build the classifiers by varying the percentage of training data from 5% to 30%. We repeat the above experimental trial 20 times and calculate the average performance over these 20 trials. Fig. 6 and Fig. 7 summarize the results of average F-measure and AUC under varying amounts of training data, respectively.

From the experimental results, we can draw some observations. First, among the three algorithms, similar to the previous results, SVM achieves significantly better F-measure performance than the other two algorithm; while NB achieves the better AUC performance than both SVM and C4.5, especially under small amount of training data. When the size of training data is large enough, SVM can achieve a comparable or better AUC performance than that of NB. Second, we found that, for all the three classifiers, increasing the size of training data generally leads to an improvement of the classification performance. But the improvement becomes minor when the size of training data is larger than some threshold (e.g. 25%). Since it is often time-consuming and expensive to label the training data manually, in practice, we should empirically determine an appropriate amount of training data to trade off between the labeling cost and the classification performance.

E. Estimated Indicator with Varying Amount of Training Data

The third experiment is to evaluate the accuracy of the quantitative indicator of the estimated process execution qualification rate, i.e., the P indicator. Following the same setup as the second experiment, we evaluate the accuracy of the estimated P values by randomly choosing a subset of candidate training data (ranging from 5% to 30%) to build the classifiers. P is calculated according to formula (1) defined in Section IV, where the estimated precision and recall values are obtained on the training set. We adopt the Root Mean Square Error (RMSE) to measure the difference between the actual P^* values and the estimated P values of the process execution qualification rate. Fig. 8 shows the experimental results.

Some observations can be drawn from the results. First of all, similar to the previous experiment, for all the three algorithms, increasing the amount of training data generally leads to the decrease of the RMSE value, indicating a more accurate estimation of the P value. Second, among the three classification algorithms, SVM achieves considerably better performance than the other two algorithms, especially when the amount of training data is small. Finally, by examining the results achieved by SVM, we found that the proposed classification based indicator is able to estimate a fairly accurate value of P (i.e., with a very small RMSE value) when the amount of training data is sufficient.

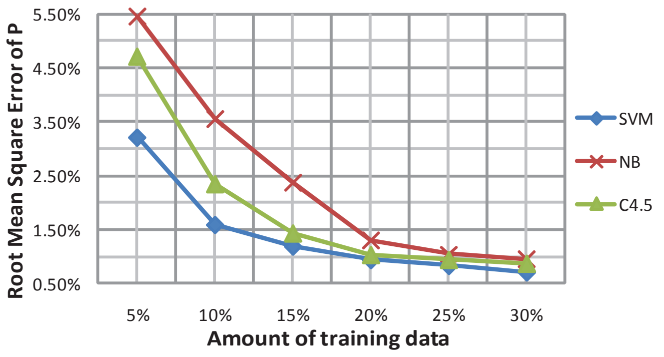


Fig. 8. Effect of training size on predicted P

F. Case Studies

We conduct case studies to examine the efficacy of the proposed indicator by (1) examining if the estimated value of the indicator is close to the true value, and (2) exploring if the proposed indicator is able to differentiate the quality of process among different projects in the same organization.

First, we show some case studies to examine the difference between the estimated P and the actual P^* values by inspecting different amounts of training data in detail. In particular, we study two cases of different settings: (i) case of small-sized training data (sampling 10% of all data for training), and (ii) case of medium-sized training data (sampling 20% of all data for training). For each of the two cases, we adopt SVM as the classification algorithm, repeat each experiment 20 times, and summarize the average results in Table V, where $\#S$ denotes the total number of sequences, $\#TN$ denotes the actual number of normal sequences, P^* denotes the actual value of process execution qualification rate, $\#PN1$ denotes the predicted number of normal sequences in case (1), and P1 denotes the predicted process execution qualification rate in case (1), while $\#PN2$ and $P2$ denote the predicted number of normal sequences and the predicted process execution qualification rate in case (2), respectively.

TABLE V
EXPERIMENTAL RESULTS FOR OUR CASE STUDY

	Project1	Project2	Project3	Project4	Department
#S	1019	478	665	460	2622
#TN	913	295	499	278	1985
P*	89.5%	61.7%	75.0%	60.4%	75.7%
#PN1	958 ±16.3	333 ±5.9	534 ±7.8	285 ±3.9	2110 ±28.2
P1	88.7% ±1.2%	65.9% ±1.8%	76.0% ±1.4%	62.8% ±2.3%	76.1% ±1.0%
#PN2	941 ±11.9	307 ±5.6	509 ±10.8	282 ±2.8	2039 ±18.5
P2	89.6% ±0.8%	61.9% ±1.1%	74.2% ±1.5%	59.4% ± 0.6%	75.5% ± 0.8%

From the results, we found that when providing sufficient amount of training data (e.g. 20%), we can achieve a highly accurate prediction of the process execution qualification rate. For example, considering the whole department, the predicted $P2 = 75.5\%$ is very close to the actual $P^* = 75.7\%$ value computed from the ground truth. Even for the case of using only small-sized training data, we still can achieve a satisfied prediction, e.g., for the whole department, $P2 = 76.1\%$ is still close to the actual $P^* = 75.7\%$. These promising results again validate the efficacy of the proposed technique for software process evaluation.

Further, according to the predicted values of the proposed quantitative indicator, we can also compare the quality of the defect management process performed in different projects of the same department. From Table V, we can conclude that the defect management process performed in Project 1 is significantly better than that performed by the other three projects. Specifically, we found that Project 2 and Project 4 have relatively low process execution qualification rates, indicating that these two projects have a high chance of executing irrational processes or violating some organization rules/standards. To verify this result, we conduct some further analysis by checking the labeled sequences of these two projects, and found that there indeed exist at least three serious problems in both Project 2 and Project 4: (i) defects were not closed or closed without fixed; (ii) defects were reopened for many times; (iii) defects were canceled without analysis.

G. Limitations of Validation

There exist four primary limitations in this work. First, we validate our approach based on the defect management process performed in four software projects from a software development center of a commercial bank in China. Despite the promising classification and prediction performance, it is unknown if the approach can also achieve the similar good performance when being applied to other more complicated software process evaluation tasks, where much longer and complex sequences may appear in different organizations. Furthermore, in our validation, an informal experiment shows that the k value in feature representation has little effect on the classification performance. However, for much longer and complex sequences, the problem of appropriate k value may arise. Future work will examine the applications of our approach to other more complicated and challenging software processes from various organizations. Besides, we will study the relation between k value and classification performance in great detail.

The second limitation relates to the question of classification performance for unusual sequences. As mentioned in Section V.C, the very positive result is partly result from the nature of the data. The classification performance for unusual sequences is not systematically analyzed in our validation. We will conduct more experiments to address this question in our future work.

Third, as emphasized in the remark on formula (1) in Section IV.F, for real-world applications, in order to calculate the estimated process execution qualification rate, we use the "precision" and "recall" values on the training set to approximate both values on the test set. Although the experiments and case study show that we can estimate a fairly accurate process execution qualification rate using formula (1), this approximate assumption may not hold in some real-world situations. This will be left primarily to our future detailed empirical studies.

Last, different organizations adopt different kinds of CASE tools, which are either open-source or commercial. However, there is no standard for the formats and contents of process execution histories stored in different software repositories of CASE tools. Many potential problems may occur during the data preprocess phase [15], which may affect the efficacy of our approach to some extent. Future work will be needed to empirically study the impact of different kinds of CASE tools.

VI. CONCLUSION

This paper proposed a semi-automated approach to software process evaluation using machine learning techniques, in which we formulate a software process evaluation task as a sequence classification problem that can be resolved by machine learning techniques. Based on the proposed framework, we present a new quantitative indicator, i.e., process execution qualification rate, as an objective evaluation of the quality of software process. We found encouraging results from our preliminary empirical studies, which validated the empirical efficacy of our technique for software process evaluation. Compared with the conventional subjective evaluation methods, the proposed machine learning approach has several advantages, and potentially opens a new direction of applying machine learning techniques to facilitate automated software process evaluation in software engineering. Our future work will apply our technique to more complicated software process evaluation tasks and investigate more sophisticated machine learning techniques. Additionally, we will conduct a comprehensive comparative study to analyze the benefits and drawbacks of prior-art traditional approaches and our proposed machine learning approach to software process evaluation tasks.

ACKNOWLEDGMENT

This work was supported by Nanyang Technological University under SUG Grant M58020016 and AcRF Tier 1 Grant RG 35/09. Special thanks to Quanxi Mi for sharing the raw data sets. We also thank the anonymous reviewers for their greatly helpful comments.

REFERENCES

[1] A. Fuggetta, "Software process: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, 2000, pp. 25–34.

[2] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay, "An empirical analysis of productivity and quality in software products," *Manage. Sci.*, vol. 46, pp. 745–759, June 2000.

[3] M. Cataldo and S. Nambiar, "On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality," in *ESEC/FSE'09*, Amsterdam, The Netherlands, 2009, pp. 101–110.

[4] L. J. Osterweil, "Software processes are software too, revisited: an invited talk on the most influential paper of icse 9," in *Proc. 19th international conference on Software engineering (ICSE'97)*, Boston, Massachusetts, 1997, pp. 540–548.

[5] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI : Guidelines for Process Integration and Product Improvement*, 1st ed. Addison-Wesley Professional, 2004.

[6] K. El Emam, J.-N. Drouin, and W. Melo, *SPICE. The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society, Los Alamitos, 1998.

[7] F. J. Pino, C. Pardo, F. García, and M. Piattini, "Assessment methodology for software process improvement in small organizations," *Inf. Softw. Technol.*, vol. 52, pp. 1044–1061, October 2010.

[8] C. G. von Wangenheim, A. Anacleto, and C. F. Salviano, "Helping small companies assess software processes," *IEEE Softw.*, vol. 23, pp. 91–98, January 2006.

[9] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st ed. Prentice Hall, Oct. 2002.

[10] M. Q. Patton, *Qualitative Research and Evaluation Methods*. Thousand Oaks, EUA : Sage, 2002.

[11] SCAMPI Upgrade Team, "Standard CMMI Appraisal Method for Process Improvement (SCAMPI) A, Version 1.3: Method Definition Document," Software Engineering Institute / Carnegie Mellon University, HANDBOOK CMU/SEI-2011-HB-001, 2011.

[12] J. E. Cook and A. L. Wolf, "Software process validation: quantitatively measuring the correspondence of a process to a model," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, pp. 147–176, April 1999.

[13] A. d. Moor and H. Delugach, "Software Process Validation: Comparing Process and Practice Models," in *Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06)*, 2006, pp. 533–540.

[14] V. Rubin, C. W. Günther, W. M. P. Van Der Aalst, E. Kindler, B. F. Van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Proc. Intl. Conf. on Software Process (ICSP'07)*, Minneapolis, MN, 2007, pp. 169–181.

[15] J. Samalikova, R. Kusters, J. Trienekens, T. Weijters, and P. Siemons, "Toward objective software process information: experiences from a case study," *Software Quality Control*, vol. 19, pp. 101–120, March 2011.

[16] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst, "The prom framework: A new era in process mining tool support," in *Applications and Theory of Petri Nets 2005*. Springer Berlin / Heidelberg, 2005, vol. 3536, pp. 1105–1116.

[17] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," *Computer*, vol. 42, pp. 55–62, August 2009.

[18] M. El-Ramly, E. Stroulia, and P. Sorenson, "From run-time behavior to usage scenarios: an interaction-pattern mining approach," in *ACM SIGKDD Conf. (KDD'02)*, Edmonton, Alberta, Canada, 2002, pp. 315–324.

[19] D. Lo, S.-C. Khoo, and C. Liu, "Efficient mining of iterative patterns for software specification discovery," in *ACM SIGKDD Conf. (KDD'07)*, San Jose, CA, 2007, pp. 460–469.

[20] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *Prof. 14th IEEE Intl. Requirements Engineering Conference*, 2006, pp. 36–45.

[21] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th International Conference on Software Engineering (ICSE'06)*, Shanghai, China, 2006, pp. 361–370.

[22] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explor. Newsl.*, vol. 12, pp. 40–48, November 2010.

[23] G. Dong, *Sequence Data Mining*. Berlin, Heidelberg: Springer-Verlag, 2009.

[24] S. Bandyopadhyay, U. Maulik, L. B. Holder, D. J. Cook, and S. Sarawagi, "Sequence data mining," in *Advanced Methods for Knowledge Discovery from Complex Data*, ser. Advanced Information and Knowledge Processing. Springer London, 2005, pp. 153–187.

[25] N. Ramasubbu and R. K. Balan, "The impact of process choice in high maturity environments: An empirical analysis," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09, 2009, pp. 529–539.

[26] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, vol. 6, pp. 151–180, August 1998.

[27] W. B. Cavnar and J. M. Trenkle, "N-Gram-Based Text Categorization," in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 161–175.

[28] B. Y. Cheng, J. G. Carbonell, and J. Klein-Seetharaman, "Protein classification based on text document classification techniques," *Proteins: Structure, Function, and Bioinformatics*, vol. 58, no. 4, pp. 955–970, Mar. 2005.

[29] J. R. Quinlan, *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1993.

[30] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Mach. Learn.*, vol. 29, pp. 103–130, November 1997.

[31] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, Sep. 1998.

[32] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve," *Radiology*, vol. 143, pp. 29–36, 1982.