RESEARCH ARTICLE

EVALUATION OF METHODS AND PROCEDURE OF SOFTWARE DEVELOPED PROCESS MODELS

Md. Asghar Ali and Dr. Ajay Jain
Department of Computer Application, Dr. A. P. J. Abdul Kalam University, Indore.

ABSTRACT

A software engineering (SE) process is a set of activities that leads to the creation of a product of software. In some cases, these activities may involve making new software from scratch in a standard development language. In other systems, a lot of new software is made by integrating and configuring existing systems, as well as by putting together and putting together off-the-shelf software or system components. This is how software systems come and go. They go through a series of passages that account for their beginning, first development, productive operation and maintenance, and retirement from one generation to the next.

Keywords: Software, software product, software engineering, software process, process models

INTRODUCTION

Software development is the process of planning, making, testing, and putting an information system into place. A software process model is an abstract representation of a process methodology. It is used to show how the process should be done and in what order. [1] Security is a very important development of making software that people can trust, but it isn't the only thing. Security should not be thought of as the only thing that a system must have in order to be considered trustworthy. These models are used to make software that is safe to use,[2]

Software development methods can be broken down into two main groups: Agile Development and Plan-Driven Development [3]. Both the Plan-driven and the Agile methodologies have processes, procedures, and documentation in them. So, in order to finish a software development project, the following method takes a different approach to using the above components.

"Plan-Driven" development is a better way to build things because you can write down what you need before you start designing. This means that you should write down the whole design before you start coding. "Agile" development is better when you don't know what the final product will look like in advance. [4] In the first case, it is assumed that it is possible to plan and design the whole project to be built from the start. In the second case, it is assumed that the project's requirements must be investigated for each separate component of the product to be built. Plan-driven development can be broken down into waterfall development and plan-driven incremental development, but it can also be mixed up.

Many different development models have been used to reach different goals. These models show which methods are best for each software system's goals. They also show how the whole software development process should be done. A software process methodology is a way to run a software project (e.g. RUP etc.). They say exactly what, when, and how different things should be made. They may not be very clear, but they set out what each member of a project team must do. [5]

RESEARCH METHODOLOGY

In general, software process evaluation is meant to look at how well some software processes are done in a certain area (ranging from a project to a department or an entire organization). In most cases, each organization has software process specifications that show how different types of software processes are represented and defined in terms of formalism models, such as finite state machines, Petri nets, or flow diagrams, along with detailed text descriptions. Specifications talk about a lot of different activities about how things work, like what artifacts are, how they work, and who is in charge of them. Depending on which software company you work for, you might see different ways of displaying software process specifications in the quality manuals of the company you work for.

In our approach, the first software we do when we want to evaluate a software process is get raw data about the activities and artifacts that make up the process. In most software development companies, there are either commercial or open-source computer-aided software engineering (CASE) tools that can be used for the SDLC. For example, software configuration management systems are used to keep track of and control changes in the software development process; requirement management systems are used to keep track of, manage, and control the needs of users. Software repositories are an important part of CASE tools because they store all of the data that comes from all of the activities in the SDLC. In our work, we get raw data from software repositories that show how software processes have worked.
Process execution is an important part of software repositories. It is a sequence of snapshots that show how different roles or artifacts are doing at different points in the software process that is being watched. Software processes can be thought of as many different things. To make things easier, we only talk about a software process as a group of process executions in this work. Software process evaluation tasks in our study are meant to look at the quality of a goal of process executions that are collection of a specific scope. To put it another way: In this study, our goal is to evaluate at a software defect management process from a collection of process executions that are stored in defect management repositories. When we use process executions, we want to use machine learning techniques and a quantitative indicator to evaluate the quality and performance of a software process in a way that is objective.

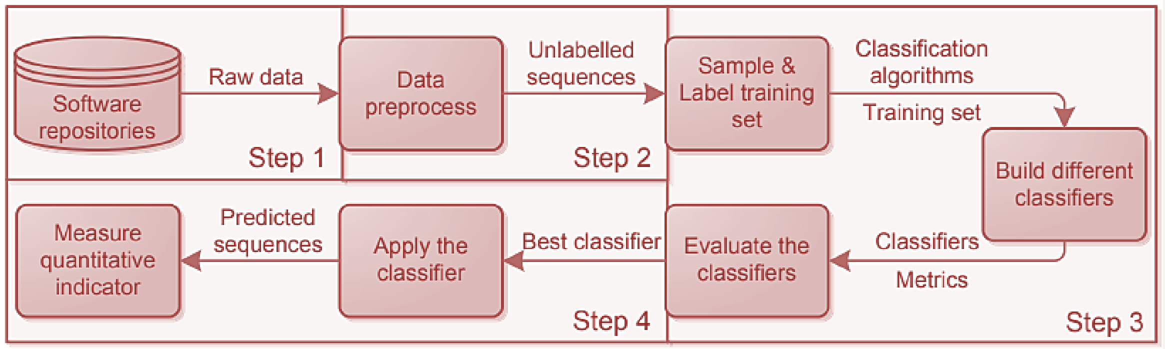

Figure 1: The framework of the proposed machine learning approach to software process evaluation

In our machine learning approach, the main idea is to use supervised sequence classification techniques to classify each process execution as either "normal" or "abnormal" when there are a lot of them. When a process is "normal," it means that it meets the standards or criteria set by the organization. We also come up with a new way to evaluate the quality and performance of the software process. We call it the process execution qualification rate. Figure 1 shows the proposed framework of our machine learning process for evaluating software processes. In a nutshell, the proposed method has four main steps. The first step is to get the raw data from software repositories. The raw data that we are interested in is the history of how the process was done (e.g., defect state change history). The second step transforms the data into a set of sequences so that it can be used for the next step. Building different sequence classifiers through labeled training data is the third step. Then, the classifiers are tested against each other. The last step uses the best sequence classifier to put the rest of the sequences into one of two groups: "normal" or "abnormal." As a result, the process execution qualification rate can be used to evaluate the quality and performance of the software process.

RESULT AND DISCUSSION

To evaluate out how well the proposed method works, we run a lot of experiments on four real-world software projects. We use our technique to evaluate at how the defect management process works in each of these projects. In the first place, we give a quick overview of these four projects and the defect management process that will be used to look at them. In the next part, we explain how we did the experiment, then we show our discussion, results, and some case studies in more detail.
Take on an experimental testbed that has four software projects on it. People from the same department of the company worked on these four projects. They came up with them in different ways. CMMI level-2 means that this organization has a certain level of process management, which is why this organization was rated this way. In addition, we want to evaluate at how the defect management process is done in

this department and on each project. If you want to know more about these four projects, look at Table 1. The last column shows how many bugs/defects each project had.
Figure 2 shows the defect management process flow that this company uses. This process is shown in the defect management process specification that this company uses. In order to keep things simple, we only show how the defect management process works by different roles, and we don't go into much more detail because there isn't enough space. Nodes are shown in this flow diagram. There are two types of nodes: "Status" node, which shows the possible status of a defect and the person who is responsible for it, and "Decision" node, which shows how a person might make a decision that affects how a defect's state changes. Starting with "New," the state of a defect changes with different decisions.
In order to figure out how well the defect management process worked for the projects in Table 1, we get the raw data for the changes in the status of the defects from the repository of the test management system that this organization used. This system was based on HP Quality Center (HQC) version 9.0.

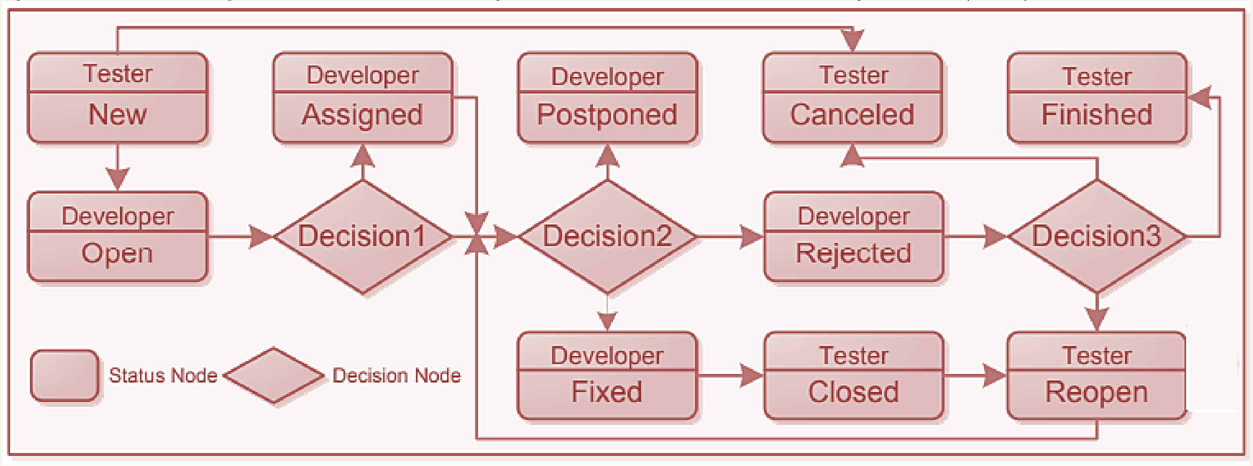

Figure 2: The defect management process flow extracted from a software process specification

Based on the processed data and the ground truth labels, we run a lot of tests to see how well our technique works. The following are some of the things we want to find out: (1) What would be the best way to classify sequences in our case? The more training data we have, the better we are at our task. (2) How good is the proposed framework at predicting how well a process will be done?

Table 1. Summary of the four projects

Project ID	**Description**	**No of defects**
1	Wealth Management System(Phase1)	478
2	Wealth Management System(Phase2)	665
3	Electronic Commercial Draft System	1010
4	Electronic Commercial Draft System	450

A Naive Bayes (NB) classifier, a decision tree, and a Support Vector Machine (SVM) are all used in the experiment. They are all used to classify our defect management process evaluation task (SVM). We use the C4.5 implementation for the decision tree, and we use a linear kernel for the SVM with the penalty parameter C set to 10. All 2622 sequences in the database are used in the standard 10-fold cross validation setting. We use this to see how well different classifiers do.

Table 2. Result of Software Evaluation

	F-measure	Recall	Precision	RMSE	AUC
NB	.920	.911	.929	.291	.945
SVM	.986	.996	.976	.143	.961
DT	.938	.947	.929	.284	.850

From Table 2 that SVM is the best of the three algorithms when it comes to all of the performance metrics. DT and NB aren't as good when it comes to different performance metrics. Using SVM, we looked at the results it got. We found that the F-measure and AUC values were both 98.6% and 96.16%, which means that the proposed scheme can learn an effective classifier for the classification task. The very positive result is partly due to the nature of the data: a lot of the sequences are the same. Another thing we do is look at the classification results for unusual sequences, which are sequences that appear less than five times in the database. We find that the classification performance is also pretty good.

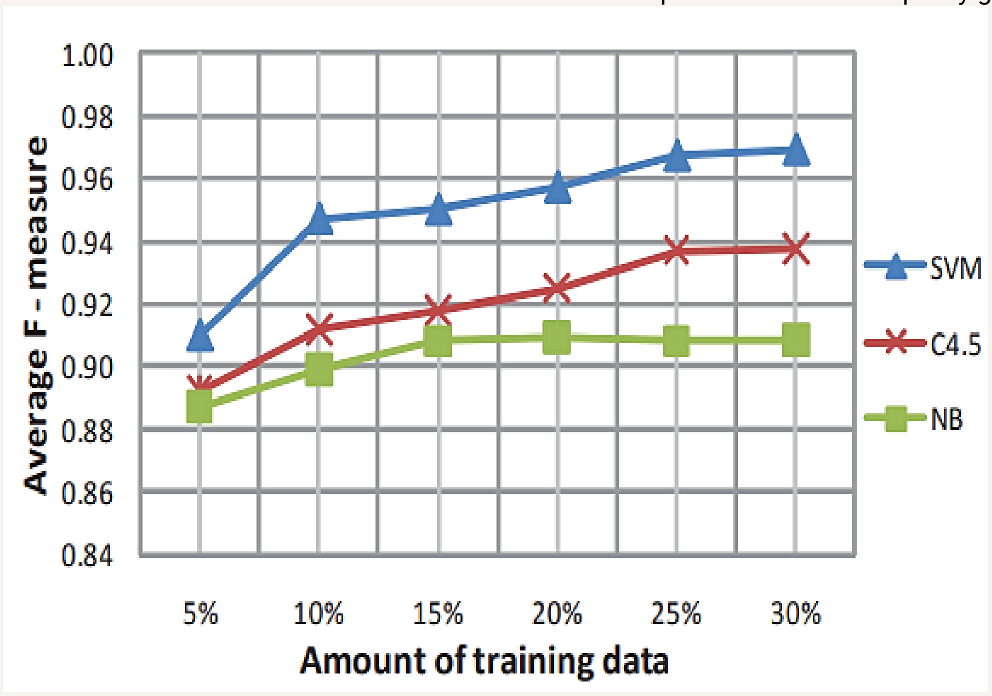

Figure 3: Effect of training data size on F-measure

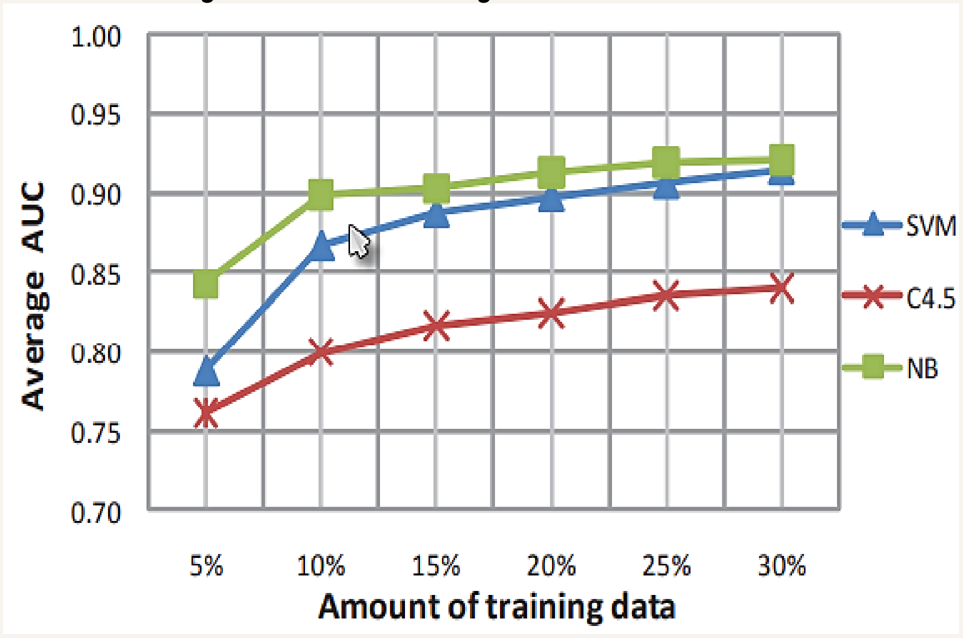

Figure 4: Effect of training data size on AUC

CONCLUSION

Here, we provide a machine learning-based semi-automated method to software process assessment, in which we frame the evaluation job as a classification issue that can be performed using machine learning techniques. A new quantitative indicator, the "process execution qualification rate," is presented as an objective assessment of software process quality using the proposed framework. We discovered good findings from our early empirical investigations, which supported the empirical usefulness of our approach for software process assessment. Compared with the usual subjective assessment approaches, the proposed machine learning methodology has various benefits, and possibly offers a new area of employing machine learning techniques to aid automated software process evaluation in software engineering. Using our current strategy, we want to tackle more challenging software process assessment jobs in the future, as well as look into more advanced machine learning approaches. Our proposed machine learning technique for software process assessment tasks will be compared to existing conventional approaches in a full comparison study.

REFERENCES

1. Ambriola, V., R. Conradi and A. Fuggetta, Assessing process-centered software engineering environments, ACM Trans. Softw. Eng. Methodol. 6, 3, 283-328, 1997.
2. Balzer, R., Transformational Implementation: An Example, IEEE Trans. Software Engineering, 7, 1, 3-14,1981.
3. Balzer, R., A 15 Year Perspective on Automatic Programming, IEEE Trans. Software Engineering, 11,11,1257-1267, 1985.
4. Balzer, R., T. Cheatham, and C. Green, Software Technology in the 1990's: Using a New Paradigm, Computer,16,11, 39-46, 1983.
5. Basili, V.R. and H.D. Rombach, The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Trans. Soft. Engr., 14, 6, 759-773, 1988.
6. Basili, V. R., and A. J. Turner, Iterative Enhancement: A Practical Technique for Software Development, IEEE Trans. Software Engineering, 1,4, 390-396, 1975.
7. Batory, D., V. Singhal, J. Thomas, S. Dasari, B. Geraci, M. Sirkin, The GenVoca model of software-system generators, IEEE Software, 11(5), 89-94, September 1994.
8. Bauer, F. L., Programming as an Evolutionary Process, Proc. 2nd. Intern. Conf. Software Engineering, IEEE Computer Society, 223-234, January, 1976.
9. Beck, K. Extreme Programming Explained, Addison-Wesley, Palo Alto, CA, 1999.
10. Bendifallah, S., and W. Scacchi, Understanding Software Maintenance Work, IEEE Trans. Software Engineering, 13,3, 311-323, 1987.
11. Bendifallah, S. and W. Scacchi, Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork, Proc. 11th. Intern. Conf. Software Engineering, IEEE Computer Society, 260-270, 1989.
12. Biggerstaff, T., and A. Perlis (eds.), Special Issues on Software Reusability, IEEE Trans. Software Engineering, 10, ,5, 1984.
13. Boehm, B., Software Engineering, IEEE Trans. Computer, C-25,12,1226-1241, 1976.
14. Boehm, B. W., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, N. J., 1981
15. Bolcer, G.A., R.N. Taylor, Advanced workflow management technologies, Software Process--Improvement and Practice, 4,3, 125-171, 1998.
16. Budde, R., K. Kuhlenkamp, L. Mathiassen, and H. Zullighoven, Approaches to Prototyping, Springer-Verlag, New York, 1984.
17. Chatters, B.W., M.M. Lehman, J.F. Ramil, and P. Werwick, Modeling a Software Evolution Process: A Long-Term Case Study, Software Process-Improvement and Practice, 5(2-3), 91-102, 2000.
18. Cook, J.E., and A.Wolf, Discovering models of software processes from event-based data, ACM Trans. Softw. Eng. Methodol. 7, 3 (Jul. 1998), 215 - 249
19. B. Curtis, H. Krasner, V. Shen, and N. Iscoe, On Building Software Process Models Under the Lamppost, Proc. 9th. Intern. Conf. Software Engineering, IEEE Computer Society, Monterey, CA, 96-103, 1987.
20. Curtis, B., H. Krasner, and N. Iscoe, A Field Study of the Software Design Process for Large Systems, Communications ACM, 31, 11, 1268-1287, November, 1988.
21. DiBona, C., S. Ockman and M. Stone, Open Sources: Voices from the Open Source Revolution, O'Reilly Press, Sebastopol, CA, 1999.
22. Fogel, K., Open Source Development with CVS, Coriolis Press, Scottsdale, AZ, 1999.

23. Garg, P.K. and M. Jazayeri (eds.), Process-Centered Software Engineering Environment, IEEE Computer Society, pp. 131-140, 1996.