ORIENTAL JOURNAL OF
COMPUTER SCIENCE & TECHNOLOGY
An International Open Free Access, Peer Reviewed Research Journal
Published By: **Oriental Scientific Publishing Co., India.**
www.computerscijournal.org

ISSN: 0974-6471
June 2017,
Vol. 10, No. (2):
Pgs. 305-310

Automatic Feedback Generation in Softwar-e Performance Engineering: A Review

JAVAID IQBAL and SYED ABRAR UL HAQ

Department of Computer Science, University of Kashmir, Jammu and Kashmir, 190006, India.
*Corresponding author E-mail: iamjavaid@gmail.com

http://dx.doi.org/10.13005/ojcst/10.02.08

(Received: January 30, 2017; Accepted: March 30, 2017)

ABSTRACT

Automation in generation of architectural feedback from performance indexes like probability distributions, mean values and variances has been of interest to the researchers from last decade. It is well established that due to the complexity in interpreting the performance indices obtained from performance analysis of software architecture and short time to the market, an automated approach is vital for acceptance of architecture based software performance engineering approach by software industry. In last decade some work has beendone in this direction. Aim of this paper is to explore the existing research in the field, which will be valuable for researchers looking forward to contributing to this research.

Keywords: Software performance engineering; Feedback generation; Antipattern approaches; Rule based approaches.

INTRODUCTION

Among non-functional attributes of software, performance is one of the key attribute. It deals with the efficiency of system in dealing with time constraints and resource allocation under certain environmental conditions. Response time, throughput and utilization are some key performance indices. Most of the problems that projects report after their release are not crashes or incorrect responses, but rather system performance degradation or problems handling required system throughput[22]. Performance problems sometimes are so severe that they require considerable design changes.One of the major performance problems that occurred in recent times was the roll out of healthcare.gov website. Healthcare.gov got crashed during its launch on October 1st, 2013 and remained inactive for several weeks. It is reported that 9.47 million users attempted to register during the first week of the launch, but only 271,000 succeeded. Initial cost of healthcare.gov was estimated to be $600 million and reports suggest that more than $2 billion were spent on tuning.

Fixing these problems is costly and causes schedule delays, lost productivity, cost overruns, lost revenues, missed market windows

and damaged customer relations. To avoid this it is of great benefit to consider software performance issues from the beginning of software development life cycle. Architectural decisions made early in software development process have utmost impact on software performance. While good architecture cannot guarantee attainment of quality goals poor architecture can prevent their achievement[23]. Selection of optimal architecture for software system can not only produce the better performing software but can also save efforts and cost involved in tuning a software system at later stages. Tuning can improve performance, but tuning changes may require considerable implementation efforts that can cause delay in the timely delivery.

Software Performance Engineering (SPE) can be used to develop software system that meets its performance requirements. It is a proactive approach that uses quantitative techniques to predict the performance of software early in design phase to identify feasible options and eliminate poor ones before implementation begins[8].SPE can be divided into 3 phases, modeling phase, analysis phase and refactoring phase. First two phases comprises the forward path of SPE while the third phase is feedback generation or backward path. Approaches available for forward path have been surveyed in[5,6]. Quite well-assessed techniques are available to automatically generate performance models and solve them. Some of the tools like SHARPE, SPE'"ED, GreatSPN, TimeNET and Two Towers are available to support software performance model solution. In this paper we focus on the third phase i.e. refactoring phase. Our aim is to explore the existing research in the field of automated generation of feedback from performance indices generated from performance analysis.

Literature review

In[4] Baldassari *et al.* introduces a PROTOB which has simulation capability for performance assessment and integrates the Petri net design notation into a CASE tool. In[15,16] Kazman *et al.*, proposed a scenario-based approach for the analysis of software architectures for various software quality attributes like modifiability, availability, security and performance. But the first SPE based approach for performance analysis was proposed by Smith and Williams in[23] in which software architecture is specified as class diagrams, deployment diagrams and sequence diagrams enriched with ITU MSC (Message sequence Chart). Software architectures are evaluated on the basis software execution model as well as system execution model using SPE'"ED tool. PASA (Performance Analysis of Software Architecture)[24] is proposed by C. Smith et al in which approach proposed in[23] is embedded. PASA is a scenario based approach which uses anti patterns to identify the performance problems in critical use cases and evaluates various architectural alternatives to find efficient one. PASA is solely manual and requires interaction between software developers and performance experts to solve performance problems. Framework for automated generation of feedback was first proposed by Cortellessa *et al.*, in[6]. 2 x 2 interpretation matrix and performance antipatterns are used for performance feedback generation in their approach. They used hierarchical approach for investigation of software performance. Flat requirements and services oriented requirements are considered for dividing a system into subsystems.Performance modeling has been done using LQN. Main drawback of this approach is that it uses restricted set of antipatterns and informal interpretation matrix. Performance antipatterns are also detected manually in the performance model. In[7] an automated approach to detect performance antipatterns is proposed. APML (antipattern modeling language) is introduced and is used to specify performance antipatterns to automatically detect them. In[9] Cortellessa *et al.* has proposed an approach to find antipatterns affecting performance requirements from antipatterns existing in the software model identified using[7]. Guiltiness factor for each antipattern is calculated and ranked antipattern list is generated. This approach is intended to be integrated with SPE approach to effectively detect and remove software antipatterns. In[8] java rule-engine application is used to detect software performance antipatterns from the software model represented in XML. Performance antipatterns are represented in XML. Static, dynamic and deployment view of software system are taken into consideration while detecting performance antipatterns. This work has been extended in[11] where antipatterns are formalized

using system independent rules and a case study is conducted. In[20] approach to automatically detect and solve performance antipattern from the software system modeled in PCM (Palladio Component Model) them is proposed. Paper defines a set of rules and actions to overcome the performance flaws. Set of thresholds are required for the formalization proposed in[11] .These formalizations may hide bad design as thresholds are not properly set. In[1] approach based on RBMS(Role-Based Modeling Language) has been used for refactoring of software architecture. Various Source Role Model (SRM) - Target Role Model (TRM) pairs have been defined and used to guide refactoring process. Challenges like context information and applicability of refactoring actions have been taken into consideration for refactoring process. In[21] round trip approach merging bottleneck analysis and performance antipatterns has been proposed. In this approach bottleneck analysis is done first to derive a system configuration where imbalance in resource allocation doesn't exist. If the performance problem still persists, the performance analysis based on performance antipatterns is conducted. By doing this solution space of antipattern based performance analysis is reduced by pruning the design alternative graph of antipatterns that involve only bottlenecks.A prototype called performance booster has been described in[27] in which several rules have been incorporated for diagnosis of performance. In this approach software architectural model, represented by annotated UML is translated into performance model (Layered queuing networks) and then analyzed. Set of rules have been defined to automate performance analysis and explore design changes. Improved software performance model is obtained from these rules and then transformed to software design model manually.

In[12] an evolutionary algorithm, EA4PO is proposed to find optimal solution for performance improvement. EA4PO is based on RPOM mathematical model and RSEF framework. RPOM is used to describe the mathematical relationship between usage of rules and optimal solution in performance improvement space. Execution of rule sequences is supported by the RSEF framework. EA4PO can help rule based performance optimization techniques to improve the quality of optimization by searching the larger solution space.In[14] an approach for transforming software architectural model to software performance model and software performance model back to software performance model has been introduced. In this paper Janus Transformation Language (JTL) is used which is a declarative model transform language and is tailored specifically to support bi-directionality of model transformation and change propagation. In this paper source meta-model is defined as a subset of UML 2.0 plus MARTE profile and the target meta-model is in Generalized Performance Interchange Format (GPMIF) representing Queuing network models. Both source and target meta-models have been encoded in Ecore format in order to be used with JTL transformation engine.In[3] a framework for supporting interpretation of software performance analysis results and generation of feedback in terms of software model refactoring based on software performance antipatterns has been introduced. In this framework a set of modules have been used which work in synergy within eclipse modeling framework (EMF).

Classification and Comparison

Various automated approaches have been proposed to tackle performance problems in the software system in early phases of software development. In last decade, focus has been given to automating generation of feedback and suggesting refactoring actions. These approaches can be classified on the basis of approaches used by then to search and detect performance problems and on the basis of model used for carrying refactoring action.

Classification on the basis of approach used to detect performance problems

On the basis of approach used to search and detect for performance problems, automated approaches for feedback generation can broadly classified into two categories. (1) Antipattern based approach and (2) Rule based approach.

Antipattern based approach

In antipattern based approach performance antipatterns are identified and removed from software architectural model in order to achieve performance goals. Performance antipatterns are bad practices that can negatively affect the performance of software system. Various domain

independent[25] and domain specific antipatterns[13,7] have been identified by researchers in recent years. This approach was first used in[24] for software performance improvement. In[1,7,8,9,11,20,21] various approaches have been proposed to automatically detect software performance antipatterns in software architectural model and remove them.

Rule based approach

In rule based approach various rules are defined to detect and solve performance problems that exist in software architecture. Rules are aimed at detecting interaction between various resources (hardware or software) and suggest refactoring action required to solve the detected performance problem. This approach has been used in[12,27].

Classification on the basis of model used to carry refactoring actions

We can also classify available approaches for automated feedback generation on the basis of model used to carry refactoring actions in (i) Software architectural model based refactoring approach and (ii)software performance model based refactoring approach.

Software architectural model based refactoring approach

In this approach, alternative software architectural model with improved performance is searched to tackle with the performance issues that emerged from performance analysis. In this approach, problematic areas in software architectural model are identified and refactoring actions are suggested for it. Antipattern based approaches use this approach. Fig 1 shows the steps involved in this approach.

Software performance model based refactoring approach

In this approach, refactoring actions are done on software performance model to

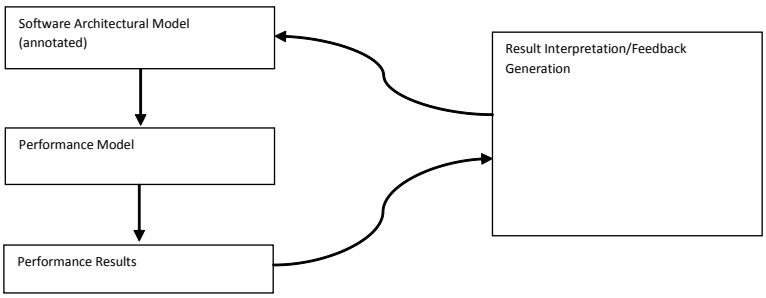

Fig. 1: Software architectural model based refactoring approach

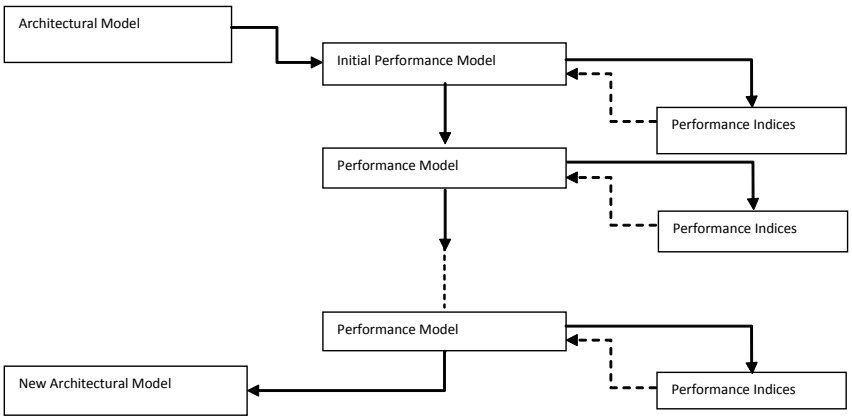

Fig. 2: Software performance model based refactoring approach

improve software performance. Once the software performance model with improved performance in achieved, performance model is then transformed into improved software architectural model. Rule based approaches use this approach. Fig 2 shows the steps involved in this approach.

CONCLUSION

In this paper, we have reviewed the research work done for automated feedback generation in software performance engineering. We have classified the approaches available for feedback generation on the basis of approaches used to find performance issues and the model used to take refactoring actions. Lack of automation in the interpretation of performance indices obtained from performance analysis is the main reason that software performance engineering is not being adopted by software industry overwhelmingly. In this scenario this paper will help researchers who are looking forward to contribute to this field.

Several problems exist in automation of feedback generation and need to be studied and solved. None of the approaches discussed in pervious section guarantee that the suggested refactoring action will improve the software performance and thus performance analysis is repeated after the refactoring is carried out. Process of refactoring is carried out till the required performance is attained or no more refactoring action could be suggested. In antipattern based approach, which use software architectural model based refactoring approach moving back and forth to the architectural model to search for better performing alternative may bring in high complexity in the whole process due to large number of possible. This problem has been tackled in software performance model based approach. But to transform software performance model to the software architectural model is complex task and more research has to be done to fully automate this process.

Quality attributes of software system are interdependent. While suggesting refactoring actions for performance improvement, other software quality attributes like availability, maintainability, testability, security, reliability etc are also to be taken into consideration and a tradeoff analysis between these attributes need to be performed. It is worth to conduct a research to design a framework where software performance is analyzed in consideration with other quality attributes.

REFERENCES

1. Arcelli, D; Cortellessa, V;Trubiani, C. (2012): Antipattern-based model refactoring for software performance improvement. In International Conference on the Quality of Software Architectures (QoSA), pages 33–42.
2. Arcelli, D; Cortellessa, V (2013): Software model refactoring based on performance analysis: better working on software or performance side? arXiv preprint arXiv:1302.5171.
3. Arcelli, D;Cortellessa, V. (2015): “Assisting Software Designers to Identify and Solve Performance Problems,” in First International Workshop on the Future of Software Architecture Design Assistants (FoSADA), WICSA and CompArch 2015, Montréal, Canada, CA.
4. Baldassari, M., *et al* (1989): “PROTOB: A hierarchical object-oriented CASE tool for distributed systems,” in Proc. Europ. Software Eng. Conf, 1989, Coventry, England.
5. Balsamo, S., *et al* (2004): M. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Software Eng*. **30**, 5, 295–310.
6. Cortellessa, V.; Frittella, L. A (2007): Framework for Automated Generation of Architectural Feedback from Software Performance Analysis. In Proceedings of the Formal Methods and Stochastic Models for Performance Evaluation, Fourth European

Performance Engineering Workshop, EPEW 2007, pp. 171–185.

7. Cortellessa, V., *et al* (2009): Approaching the model-driven generation of feedback to remove software performance flaws. In Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on, pages 162–169. IEEE.
8. Cortellessa, V.;Marco, A. Di;Trubiani, C. (2010a): Performance Antipatterns as Logical Predicates. In IEEE International Conference on Engineering of Complex Computer Systems, pp. 146–156.
9. Cortellessa, V., *et al* (2010b): A Process to Effectively Identify "Guilty" Performance Antipatterns. In Fundamental Approaches to Software Engineering, pp 368–382.
10. Cortellessa, V.; Di Marco, A.; Inverardi, P. (2011): Model-Based Software Performance Analysis. Springer, Heidelberg
11. Cortellessa, V; Marco, A. Di;Trubiani, C. (2012): An approach for modeling and detecting software performance antipatterns based on first-order logics. Journal of Software and Systems Modeling. DOI: 10.1007/s10270-012-0246-z.
12. Du, X, et at (2015): "An Evolutionary Algorithm for Performance Optimization at Software Architecture Level", In IEEE Congress on Evolutionary Computation (CEC):
13. Dudney, B., *et al* (2003): J2EE antipatterns.
14. Eramo, R, *et at* (2012): "Performance-driven architectural refactoring through bidirectional model transformations," in QoSA, pp. 55–60.
15. Kazman, R., *et al* (1996): Scenario-based analysis of software architecture. *IEEE Software* **13** (6), 47–56.
16. Kazman, R., *et al* (1998): The architecture tradeoff analysis method. In:Proceedings of the 4th International Conference on Engineering of Complex Computer Systems. IEEE Computer Society Press, Montery, CA, pp. 68–78.
17. Koziolek, H. (2010): "Performance Evaluation of Component-based Software Systems: A Survey," *Perform. Eval.*, **67**, (8), pp. 634–658.
18. Smith, C.U.(2015): "Software Performance Engineering Then and Now: A Position Paper", Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development.
19. Tate, B., *et al* (2003): EJB.
20. Trubiani, C;Koziolek, A. (2011): Detection and solution of software performance antipatterns in palladio architectural models. In ICPE, pages 19-30.
21. Trubiani, C., *et al* (2014): "Exploring synergies between bottleneck analysis and performance antipatterns," in ICPE, 2014, pp. 75–86.
22. Vokolos, F. I; Weyuker E. J.(1998): "Performance Testing of Software Systems," Proceedings, First ACM SIGSOFT International Workshop on Software and Performance, Santa Fe, NM ,pp. 80–87
23. Williams, L. G; Smith C. U (1998): Performance evaluation of software architectures, Proceedings of the 1st international workshop on Software and performance, p.164-177, Santa Fe, New Mexico, USA
24. Williams, L. G; Smith, C. U. (2002): PASA: An Architectural Approach to Fixing Software Performance Problems, Proc. of CMG international conference.
25. Williams, L. G; Smith, C. U. (2003): More new software performance antipatterns: Even more ways to shoot yourself in the foot. In: Computer measurement group conference.
26. Woodside, Murray; Franks , Greg; Petriu, Dorina (2007): The future of software performance engineering, in: Future of Software Engineering, FOSE'07, IEEE Computer Society, Los Alamitos, CA, USA, pp. 171–187
27. Xu, J (2008): Rule-based automatic software performance diagnosis and improvement, in: Proc 7th ACM Int. Workshop on Software and Performance, Princeton, NJ, pp. 1–12.