Concern-driven Reporting of Software Performance Analysis Results

Dušan Okanović, André van Hoorn, Christoph Zorn
University of Stuttgart, Germany

Fabian Beck
University of Duisburg-Essen, Germany

Vincenzo Ferme
Kiratech S.p.A., Paradiso (Lugano), Switzerland*

Jürgen Walter
University of Würzburg, Germany

ABSTRACT

State-of-the-art approaches for reporting performance analysis results rely on charts providing insights on the performance of the system, often organized in dashboards. The insights are usually data-driven, i.e., not directly connected to the performance concern leading the users to execute the performance engineering activity, thus limiting the understandability of the provided result. A cause is that the data is presented without further explanations.

To solve this problem, we propose a concern-driven approach for reporting of performance evaluation results, shaped around a performance concern stated by a stakeholder and captured by state-of-the-art declarative performance engineering specifications. Starting from the available performance analysis, the approach automatically generates a customized performance report providing a chart- and natural-language-based answer to the concern. In this paper, we introduce the general concept of concern-driven performance analysis reporting and present a first prototype implementation of the approach. We envision that, by applying our approach, reports tailored to user concerns reduce the effort to analyze performance evaluation results.

ACM Reference Format:
Dušan Okanović, André van Hoorn, Christoph Zorn, Fabian Beck, Vincenzo Ferme, and Jürgen Walter. 2019. Concern-driven Reporting of Software Performance Analysis Results. In *Tenth ACM/SPEC International Conference on Performance Engineering Companion (ICPE '19 Companion), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3302541.3313103

1 INTRODUCTION

Performance engineering approaches for prediction, profiling, and load testing provide powerful tools for developers and operations specialists to investigate timing- and resource-related runtime properties of software systems. The recorded data and derived findings are often complex and large. While existing tools provide some support for data analysis and report on potential problems, a considerable amount of work, requiring detailed domain knowledge, is left to be done manually (e.g., creating dashboards, extracting information from performance data). Various performance visualization approaches [9] can explain complex data and relationships between analysis results, and thereby make the manual process more effective and efficient. Still, current analysis approaches and visualizations are limited in providing an actionable answer to the users, because they do not consider the users' concern when selecting presentation techniques to be used. Moreover, the majority of approaches does not explain the complex data and findings shown in lists, tables, and visualizations. This makes performance engineering approaches hard to access and leverage, especially for people who only occasionally deal with performance-related issues.

Building on the vision of *Declarative Performance Engineering* (DPE) [20], the stakeholders (e.g., service developers, providers) should be able to state their concerns of interest in a declarative manner, while not necessarily being aware of the approaches used to analyze and to provide answers. In systems and software engineering, a concern is defined as an "*interest in a system relevant to one or more of its stakeholders*" and it "*pertains to any influence on a system in its environment, including developmental, technological, business, operational, (...) influences*" [1]. For example, a performance concern might be "*What is the response time of the software I am developing?*" To answer this, a common approach is to perform load-testing [10], that is, applying a production(-like) workload on the system that is being tested in order to assess its behavior. After the load test is executed, the data is provided in the form of tables and charts, from which the user can deduce what is the answer. However, the user needs to know which method or tool is appropriate for the analysis, and then how to formulate the concern using the available language of the chosen tool. Hence, a declarative approach is needed, which *i)* abstracts the underlying tools from the user, allowing for stating of performance-related concerns, and *ii)* provides comprehensible results tailored to these concerns. Especially non-experts in performance engineering would benefit from interfaces that can help in easier stating of their concerns, as well as from tailored presentations explaining the results.

In this paper, we propose the concept of *concern-driven reporting of performance analysis results* (Section 2): Users can declare a concern using natural language. Based on the stated concern, automated methods of performance engineering are used to analyze available data and provide results. The approach automatically generates a report that includes visualization of data, and an explanation of the results using natural language. To demonstrate our

ICPE '19 Companion, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6286-3/19/04…$15.00
https://doi.org/10.1145/3302541.3313103

*This work was done at the University of Stuttgart.

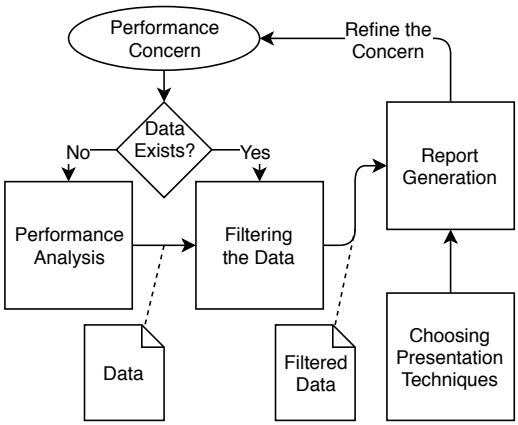


Figure 1: Control flow for the concern-driven reporting.

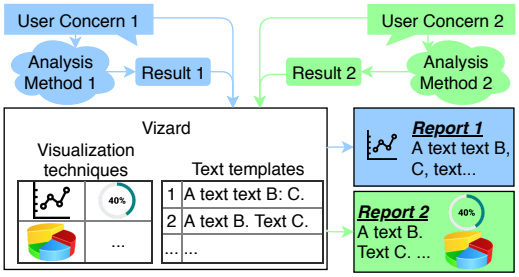


Figure 2: An illustration of the concern-driven software performance reporting.

approach, we present our proof-of-concept implementation, the Vizard tool, focusing on few specific types of performance concerns, but implementing all parts of the concept (Section 3). We believe that such tailoring of performance analysis reports to include only the data relevant to answer the stated concern is of major importance, because the amount of data provided in current approaches can be overwhelming and counterproductive.

2 CONCERN-DRIVEN PERFORMANCE REPORTING

An overview of our concern-driven report generation is provided in Figure 1. The process starts with the stakeholder stating a *performance concern*. Based on this concern, a *performance analysis* tool provides a result, including the relevant performance data. In this step, we rely on the support from Declarative Performance Engineering (DPE) [19] to select a tool or a list of suitable tools to perform the analysis. Performance analysis can require running an experiment (e.g., profiling, load testing) or performance prediction using model-based approaches. The interaction between our approach and underlying tools requires an abstraction layer to translate concerns into configurations, used by the selected tools. To collect the data, we rely on the approaches such as OPEN.xtrace [14].

As discussed before, the amount of data can be large, and therefore hard to grasp. To tackle this, *data filtering* ensures that only the data from the analysis result that is directly related to the stated concern is included in the report. After that, the *report is generated*, using *presentation techniques* that were, again, chosen based on the stated concern. If needed, the data that was filtered out from the report can also be made available to the user upon request. The user can further *refine the concern*, and a new report is generated based on the existing data (available immediately) or new data (available after another execution of the performance analysis of choice).

In Figure 2, we show how we envision the report generation in our approach. A central component is the *Vizard*, which has three responsibilities. First, it has to choose visualization techniques to be used in the report, filtering and adjusting to the available data and the stated concern. In performance engineering, data is collected over time and is usually visualized as a time series in a line chart. To represent communication between components in software (on different levels of abstraction and with different granularity), graphs and hierarchies can be used. Additionally, if source code of the analyzed software is available during analysis, it can be used as (simple or decorated) text, e.g., to show the developer where and how to apply the fix. We plan to employ the classification by Isaacs et al. [9] to select the visualization based on the origin of data (e.g., hardware, software), type of the data (e.g., time series, execution traces), and stated concerns (e.g., *"What is the maximum number of users that can be handled?"*, *"What is the bottleneck?"*, *"What is the root-cause of response time increase?"*) and manifest it in a rule-based selection mechanism.

The second task is to generate a natural language description of the results. We plan to use natural language generation [6] and rely on template-based natural language generation techniques. We prepare templates related to different visualizations and concerns, then fill in the template according to the concern and the available performance data.

The last step is to combine generated visualizations and language description into tailored reports. For this, we plan to define different widget-like components realized by one or more charts, and a generated text description. We then define customized reports according to different user's concern supported by our approach. These components can be reused and filled in according to the available data and the given user's concern.

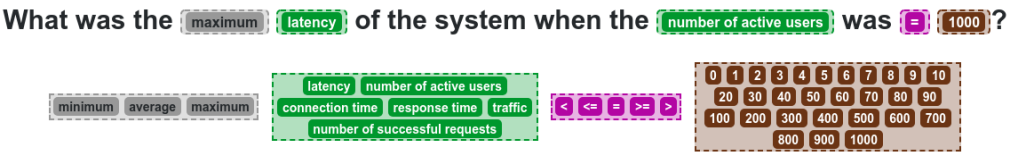


Figure 3: Creating a concern in the prototype implementation.

3 VIZARD TOOL

In this section, we present our preliminary work on the Vizard tool, which implements the main ideas of the approach. Our prototype implementation consists of *a*) a concern specification and *b*) a report generation.

Concern specification. To specify the concern, a user can first select an example which will serve as a template for specifying concerns. The current version supports specifying concerns typical for load testing, but we plan to extend it to other software performance analysis methods. Concern specification involves dragging parameters from categories into blank fields of the chosen template (Figure 3). After the query is created, the user can choose which (appropriate) performance analysis method should be used. Currently, only the load testing with JMeter[1] or Locust[2] is implemented. Based on the example from the introduction ("*What is the maximum latency of a software service with 1000 active users?*"), we will use JMeter to provide us with data for the report. Depending on the level of expertise, users can enter additional details about the experiment. For non-expert users, default values are provided. For performance experts with the knowledge of JMeter, it is possible to directly edit the script file to be used. When the concern is specified, the user can download the script file and use it to run the experiment. In the future, we plan to automate this process using the DPE tooling [17].

Report generation. The results of the experiment and the stated concern are uploaded to the report generation tool. Based on the results, the tool will provide generated text with details on the experiment (Figure 4a), an answer in a natural-language form (Figure 4b), as well as the most important data to support the claims in it (Figure 4c). For the data, we provide an explanation what each shown metric represents and how it influences the performance.

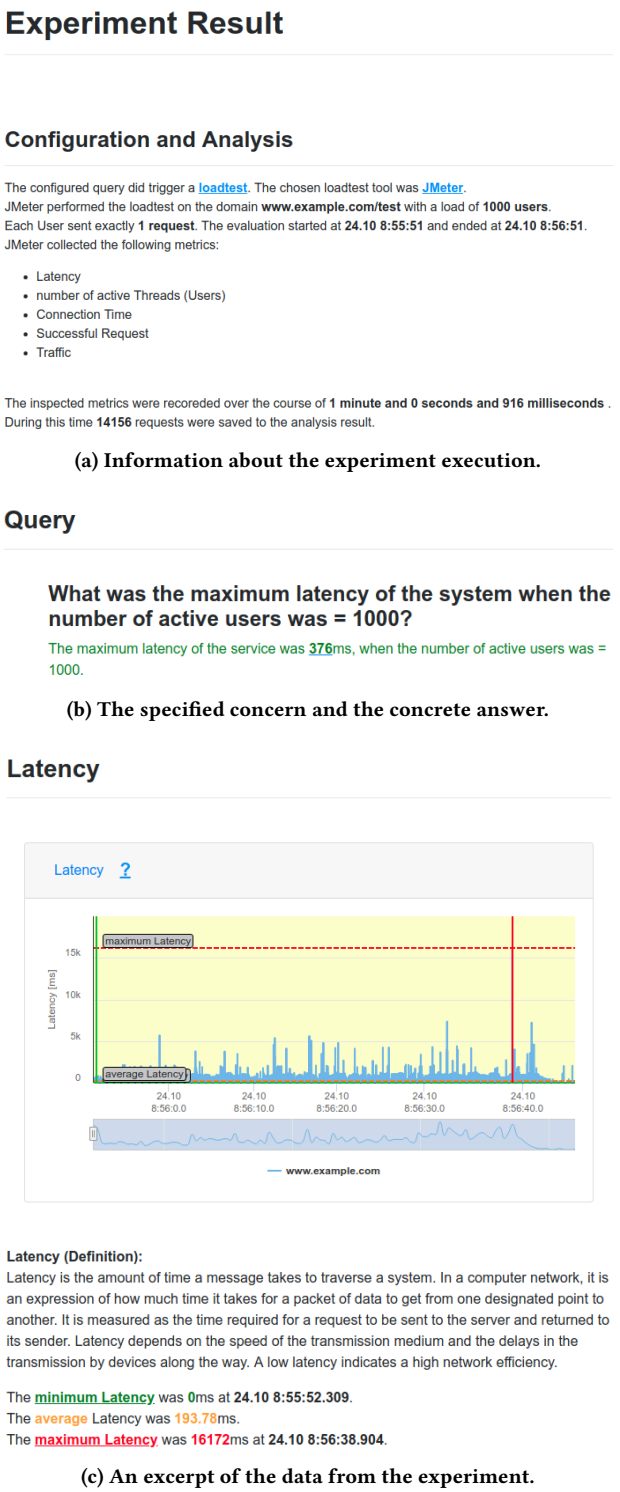
The user can now analyze the report and interact with it. To support the "Refine the Concern" connection from Figure 1, we allow the user to change the concern. If the available data can be used to answer this new concern, the report will be updated with a new answer. Visualizations in the report are also interactive, i.e., the user can inspect particular values or change observation intervals.

For inexperienced users, we also provide help by adding links to explanations of some performance analysis terms, as can be seen in Figure 4a. These links are a part of the report template.

4 RELATED WORK

An overview of the state-of-the-art of (both open-source and commercial) application performance management (APM) tools [7, 8] shows that these tools provide visualization of collected data. The data is presented using numeric values, time-series charts, or node-link diagrams. However, a significant effort has to be invested to set up dashboards that will provide an overview of the systems' performance, and allow for detection and diagnosis of problems.[3] It is often left to the user to manually make sense of the available data and to find the root causes of performance problems, although some

[1] Apache JMeter, https://jmeter.apache.org/
[2] Locust, https://locust.io/
[3] Performance Dashboard Design: How to Put Data to Work, http://www.uxbooth.com/articles/performance-dashboard-design-how-to-put-data-to-work/



(a) Information about the experiment execution.

(b) The specified concern and the concrete answer.

(c) An excerpt of the data from the experiment.

Figure 4: An excerpt from the report based on the stated concern.

tools provide bottleneck detection or alerting, e.g., of pre-configured threshold violations. The stakeholders have to know what they are looking for, often guided by their previous experience, but also prejudice. Similar to our approach, the PAVO tool [18] implements ideas from DPE to abstract the underlying analysis approaches from the user, and to provide suitable visualization techniques for a given concern. However, it does not provide any interpretation of data.

While natural language generation and visualization are both established fields, we are only aware of one approach that has already combined techniques from both areas for software execution data: Beck et al. [4] generate reports on method executions, with a focus on textual explanations of dynamic call dependencies and simple visualizations, only marginally covering performance information. There are also works that use textual report generation and text summarization (without visualization) in other software engineering scenarios, for instance, to automatically generate code documentation [11, 12, 16], commit messages [5], or release notes [13]. Integrating performance data into software code for performance reporting, small visualizations can be embedded into the code view [2, 3]. Beyond specific solutions that focus on integrated reporting, there exists a variety of performance visualization techniques, but for a general overview we refer to Isaacs et al. [9].

5 CONCLUSION AND FUTURE WORK

In this paper, we proposed our vision of generating comprehensive performance analysis reports tailored to user concerns. Instead of being swamped with data, tailored answers allow to better understand and solve performance problems. Our approach employs concern-driven analyses in combination with natural language generation and visualization techniques. We also presented Vizard, a prototype implementation of our approach.[4]

Next steps in our research include investigating further software performance concerns and proper ways of presenting the answers to them. We will identify typical concerns and express them using existing formalism, such as the ones proposed by Walter et al. [20] or Schulz et al. [15]. This will be done by investigating different use cases, i.e., how different performance analysis tools allow their users to state concerns and how the results are reported. As we need to access the data in these reports, we have to develop a model that will be able to contain metrics and other data from different tools.

To evaluate the approach, we will design and perform a user study. The study should involve both non-experts and experts. However, performing a controlled experiment evaluating our approach against existing ones would be challenging due to the many variables involved in the design and the lack of comparable systems. In contrast, we suggest a qualitative approach where the participants—given a realistic scenario—can select different elements for the reports from a list of alternative choices (e.g., text representations and visualizations of data, explanations, and solution recommendations). We might discover differences between the information needs of non-experts and experts and can use the results to fine-tune and tailor the reports.

[4] Available at: https://github.com/DECLARE-Project/Vizard

ACKNOWLEDGMENT

The work in this paper is a part of the project "Visual Reporting of Performance and Resilience Flaws in Software Systems", supported by Baden-Württemberg Stiftung, by the German Research Foundation (DFG) in the Priority Programme "DFG-SPP 1593: Design For Future—Managed Software Evolution" (HO 5721/1-1 and KO 3445/15-1), by the German Federal Ministry of Education and Research (grant no. 01IS17010, ContinuITy), and by the Swiss National Science Foundation project (178653).

Special thanks goes to Matthias Popp, for his support in the development of this approach.

REFERENCES

[1] ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E)*, 2011.
[2] S. Baltes, O. Moseler, F. Beck, and S. Diehl. Navigate, understand, communicate: How developers locate performance bugs. In *Proc. Int. Symp. on Empirical Soft. Eng. and Measurement*, ESEM, pages 1–10, 2015.
[3] F. Beck, O. Moseler, S. Diehl, and G. D. Rey. In situ understanding of performance bottlenecks through visually augmented code. In *Proc. Int. Conf. on Program Comprehension*, ICPC, pages 63–72, 2013.
[4] F. Beck, H. A. Siddiqui, A. Bergel, and D. Weiskopf. Method Execution Reports: Generating text and visualization to describe program behavior. In *Proc. IEEE Working Conf. on Soft. Visualization*, VISSOFT, pages 1–10, 2017.
[5] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *Proc. Int. Working Conf. on Source Code Analysis and Manipulation*, SCAM, pages 275–284, 2014.
[6] A. Gatt and E. Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.
[7] C. Haight and F. D. Silva. Gartner's magic quadrant for application performance monitoring suites. http://www.gartner.com/, 2016.
[8] C. Heger, A. van Hoorn, M. Mann, and D. Okanovic. Application performance management: State of the art and challenges for the future. In *Proc. Int. Conf. on Perf. Eng.*, ICPE, pages 429–432. ACM, 2017.
[9] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer. State of the Art of Performance Visualization. In *EuroVis - STARs*, 2014.
[10] Z. M. Jiang and A. E. Hassan. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.
[11] P. W. McBurney and C. McMillan. Automatic documentation generation via source code summarization of method context. In *Proc. Int. Conf. on Program Comprehension*, ICPC, pages 279–290. ACM, 2014.
[12] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker. Automatic generation of natural language summaries for Java classes. In *Proc. IEEE Int. Conf. on Program Comprehension*, ICPC, pages 23–32. IEEE, 2013.
[13] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proc. Int. Symp. on Foundations of Soft. Eng.*, FSE, pages 484–495. ACM, 2014.
[14] D. Okanović, A. van Hoorn, C. Heger, A. Wert, and S. Siegl. Towards performance tooling interoperability: An open format for representing execution traces. In *Proc. Eur. Workshop Comp. Perf. Eng.*, pages 94–108, 2016.
[15] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, and C. Pautasso. Behavior-driven load testing using contextual knowledge—approach and experiences. In *Proc. Int. Conf. on Perf. Eng.*, 2019.
[16] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for Java methods. In *Proc. IEEE/ACM Int. Conf. on Automated Soft. Eng.*, ASE, pages 43–52, 2010.
[17] J. Walter, S. Eismann, J. Grohmann, D. Okanović, and S. Kounev. Tools for declarative performance engineering. In *Int. Conf. on Perf. Eng.*, pages 53–56, 2018.
[18] J. Walter, M. König, S. Eismann, and S. Kounev. PAVO: A Framework for the Visualization of Performance Analyses Results. In *Proc. Symp. on Soft. Perf.*, SSP, 2016.
[19] J. Walter, A. van Hoorn, and S. Kounev. Automated and adaptable decision support for software performance engineering. In *Proc. Int. Conf. on Perf. Evaluation Methodologies and Tools, VALUETOOLS*, pages 66–73, 2017.
[20] J. Walter, A. van Hoorn, H. Koziolek, D. Okanović, and S. Kounev. Asking what?, automating the how?: The vision of declarative performance engineering. In *Proc. Int. Conf. on Perf Eng.*, ICPE, pages 91–94. ACM, 2016.