DynamicSpotter: Automatic, Experiment-based Diagnostics of Performance Problems

— Invited Demonstration Paper —

Alexander Wert
Karlsruhe Institute of Technology, Am Fasanengarten 5, Karlsruhe, Germany
alexander.wert@kit.edu

ABSTRACT

Performance problems in enterprise software applications can have a significant effect on the customer's satisfaction. Detecting software performance problems and diagnosing their root causes in the testing phase as part of software development is of great importance in order to prevent unexpected performance behaviour of the software during operation. DynamicSpotter is a framework for experiment-based diagnosis of performance problems allowing to detect performance problems and their root causes fully automatically. Providing different kind of extension points, DynamicSpotter allows for utilizing external measurement tools for the execution of performance tests. Building upon an extensible knowledge base, DynamicSpotter provides means to extend the diagnostic capabilities with respect to detection of additional types of performance problems.

1. INTRODUCTION

The performance of enterprise software systems plays a crucial role for the success of software vendors and operators as it directly affects customer satisfaction. In order to prevent end users from facing performance problems during operation of software applications, performance problems and their root causes need to be identified in the testing phase of a software development process. However in practise, diagnostics of performance problems is a highly manual task which requires significant expertise in performance engineering, rendering frequent, thorough analysis of performance problems impractical.

In this demonstration paper, we introduce DynamicSpotter (DS) [2], a novel framework for automatic, experiment-based diagnostics of performance problems in enterprise software systems. DS automates the execution of performance test series, gathering of measurement data, as well as the analysis of measured data in order to scan fully automatically a system under test (SUT) for known types of performance problems and their root causes. Thereby, DS utilizes existing performance measurement tools for instrumentation of the SUT, gathering of measurement data and load generation. Designed for the testing phase of a software development process, DS allows for a frequent, regular execution of diagnostic runs, for instance as part of continuous integration.

DS is an open source tool, which is available on GitHub [2]. DS is designed as an extensible and flexible framework fostering enhancements and extensions by creation of adapters to support the usage of additional measurement tools and detection of additional types of performance problems. In 2014, DS has been reviewed, accepted by the SPEC Research Group and is part of SPEC RG's repository of recommended performance evaluation tools.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.
ACM 978-1-4503-3248-4/15/01.
http://dx.doi.org/10.1145/2668930.2693844.

2. THE DIAGNOSTICS APPROACH

DynamicSpotter (DS) is a framework for experiment-based, automatic diagnostics of performance problems, combining the concepts of software performance anti-patterns [6] with systematic experimentation. As many performance anti-patterns share common characteristics, they can be structured in a hierarchical way, yielding a taxonomy which covers performance problem types from their high level symptoms to their specific root causes [8, 9]. DS utilizes the taxonomy as a decision tree in order to systematically search for performance problems. For each node of the taxonomy, a detection heuristic is applied which is responsible to decide on the existence of the corresponding performance problem in the SUT. Therefore, a detection heuristic specifies a set of experiments to be executed, the data to be gathered and a set of analysis rules to be applied on the data.

While traversing the taxonomy and applying corresponding detection heuristics for each performance problem, DS generates a report. The report states for each node in the taxonomy whether the corresponding performance problem exists in the SUT and points to the location of the corresponding root cause in the SUT.

3. ARCHITECTURE

The architecture of DynamicSpotter (DS) is depicted in Figure 1. DS requires a *taxonomy on performance problems* and corresponding *detection heuristics* as input which are generic artifacts usually provided by performance experts. Hence, the taxonomy and detection heuristics are intended to be reused in different application contexts of DS. *DS Core* is the main component which is responsible for coordinating the instrumentation of the SUT, the measurement process, gathering and pre-processing measurement data, as well as analyzing data. Furthermore, *DS Core* implements the high level process of iterating a taxonomy on performance prob-

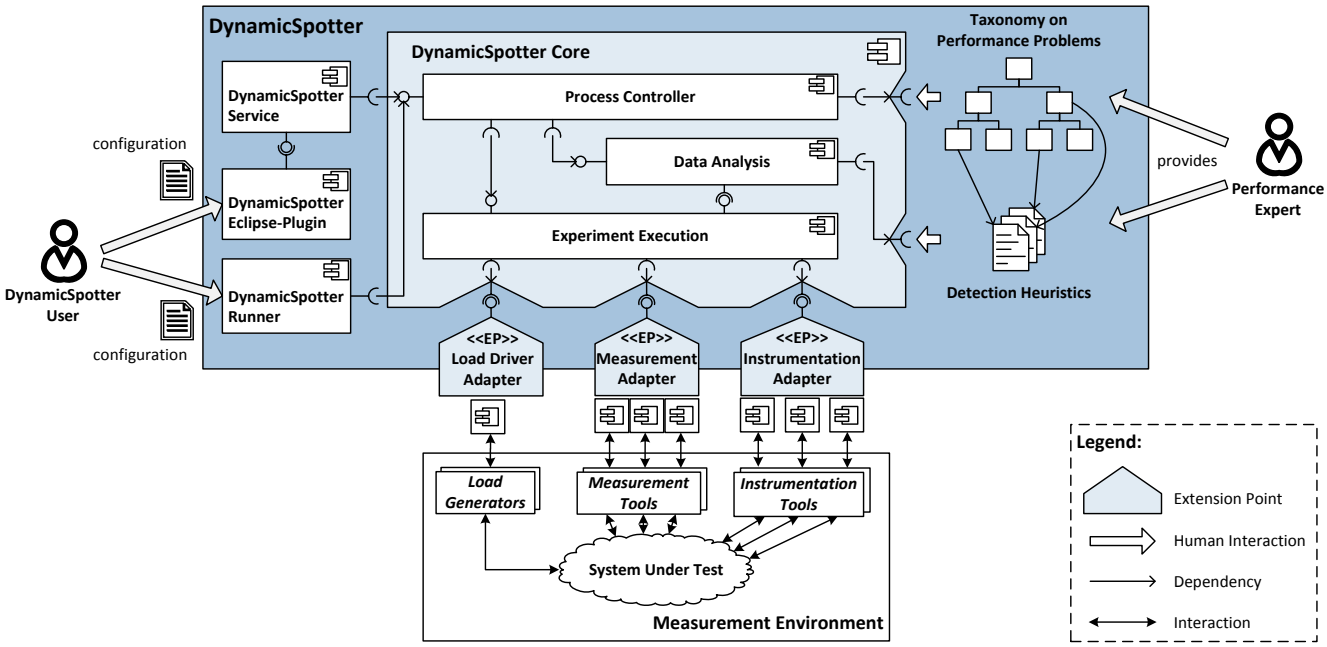

Figure 1: Architecture of DynamicSpotter

lems. The *Experiment Execution* component is responsible for automating experiment execution, while the *Data Analysis* component encapsulates the analysis of measurement data for individual performance problems according to the detection rules defined in the detection heuristics. For each sub-task, DS provides extension points (cf. ≪EP≫ in Figure 1) allowing to provide adapters for specific tools used for instrumentation, monitoring and workload generation:

Load Generator Adapter: This extension point provides means to use existing load generation tools like Apache JMeter [4], Faban [3], etc. for workload generation.

Instrumentation Adapter: This extension point allows to provide adapters for instrumentation tools like DiSL [5], Kieker [7], our own instrumentation tool AIM (Adaptable Instrumentation and Monitoring) [1], etc. Thereby, DS uses a generic instrumentation description model (IDM) to decouple the instrumentation description from the tool realizing it.

Measurement Adapter: A measurement adapter is used to control data collection, as well as to transform and transfer data from the monitoring tools to DS in a common data representation format. Though a specific instrumentation and measurement adapter are often realized within one external tool, they represent conceptually different tasks.

DS may use several instrumentation, measurement, and load generation adapters, whereby the selected set of adapters to be used in a specific application context of DS determines the *Measurement Environment*. In order to run DS, a *DS User* has to describe the *Measurement Environment* in a configuration which is either passed to a headless DS process (*DS Runner*), or the user may prefer an interactive way of creating a configuration for DS using the *DS Eclipse-Plugin*.

4. CONCLUSION & FUTURE WORK

DynamicSpotter (DS) is a framework for experiment-based, fully automatic diagnostics of performance problems in enterprise software systems. DS has been applied in multiple case studies [8, 9] showing promising results with respect to the automation of performance problems diagnostics. As DS is a framework, the diagnostics capabilities highly depend on the detection heuristics available for DS. Currently, heuristics for the detection of software bottlenecks and communication performance anti-patterns are available. Extending the existing set of supported detection heuristics is an important task for future work. Furthermore, we are working on providing additional adapters for common measurement tools (e.g. Kieker [7]).

5. REFERENCES

[1] Aim homepage. http://sopeco.github.io/AIM/.

[2] DynamicSpotter homepage. http://sopeco.github.io/DynamicSpotter/.

[3] Faban homepage. http://faban.org/.

[4] Apache Software Foundation. Apache JMeter homepage. http://jmeter.apache.org.

[5] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi. Disl: a domain-specific language for bytecode instrumentation. In *AOSD'11*. ACM, 2012.

[6] C. Smith and L. Williams. Software performance antipatterns; common performance problems and their solutions. In *CMG-CONFERENCE-*, 2002.

[7] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *ICPE'12*. ACM, 2012.

[8] A. Wert, J. Happe, and L. Happe. Supporting swift reaction: automatically uncovering performance problems by systematic experiments. In *ICSE'13*. IEEE Press, 2013.

[9] A. Wert, M. Oehler, C. Heger, and R. Farahbod. Automatic Detection of Performance Anti-patterns in Inter-component Communications. In *QoSA'14*. ACM, 2014.