Runtime Metric Meets Developer

Augmenting Source Code with Runtime Information to Make Programs Live (Demonstration)

Jürgen Cito
University of Zurich
Zurich, Switzerland
cito@ifi.uzh.ch

ABSTRACT

For software developers, source code is the static blueprint that is transformed into running behavior. In modern applications, this behavior is often facilitated through distributed systems. Thus, behavior developers observe on their machine is usually not the same as it is perceived by end users. We propose a system where source code is mapped with information gathered at runtime to make programs feel "alive". We implemented a system that maps runtime performance to source code artifacts. In this demo, we show how performance problems can be prevented in different scenarios.

KEYWORDS

Programming experience; Software Analytics; Performance engineering

ACM Reference format:
Jürgen Cito. 2017. Runtime Metric Meets Developer. In *Proceedings of Programming Experience Workshop, Brussels, Belgium, April 4, 2017 (PX/17),* 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 AUGMENTING SOURCE CODE WITH RUNTIME INFORMATION

As software developers, we build functionality based on abstractions that manifest themselves as structural text we call source code of programs. In this demo, we want to entertain a conservative idea of live programming: Mapping the structural text we write with their execution footprint gives developers a feel of how their programs are perceived "in the wild". It makes program code, that is usually static blueprints, feel alive. In the following, we describe the conceptual model of that idea and an implementation that focuses of software performance.

1.1 Conceptual Model

We propose a conceptual framework (Figure 1) that models runtime information together with source code to achieve tighter integration of runtime aspects in the software development workflow.

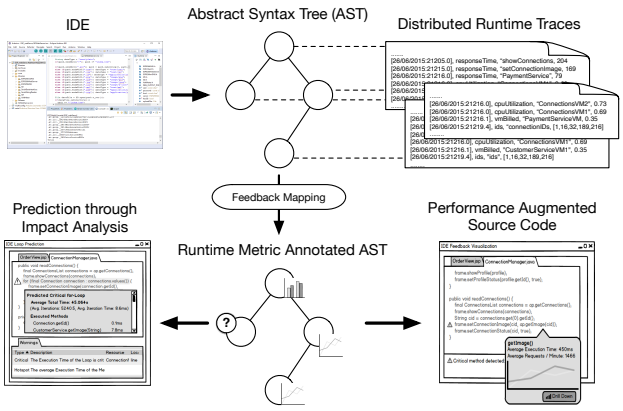


Figure 1: Conceptual Framework for the proposed approach. Nodes of source code, represented as a simplified Abstract Syntax Tree (AST), are annotated with runtime performance metrics.

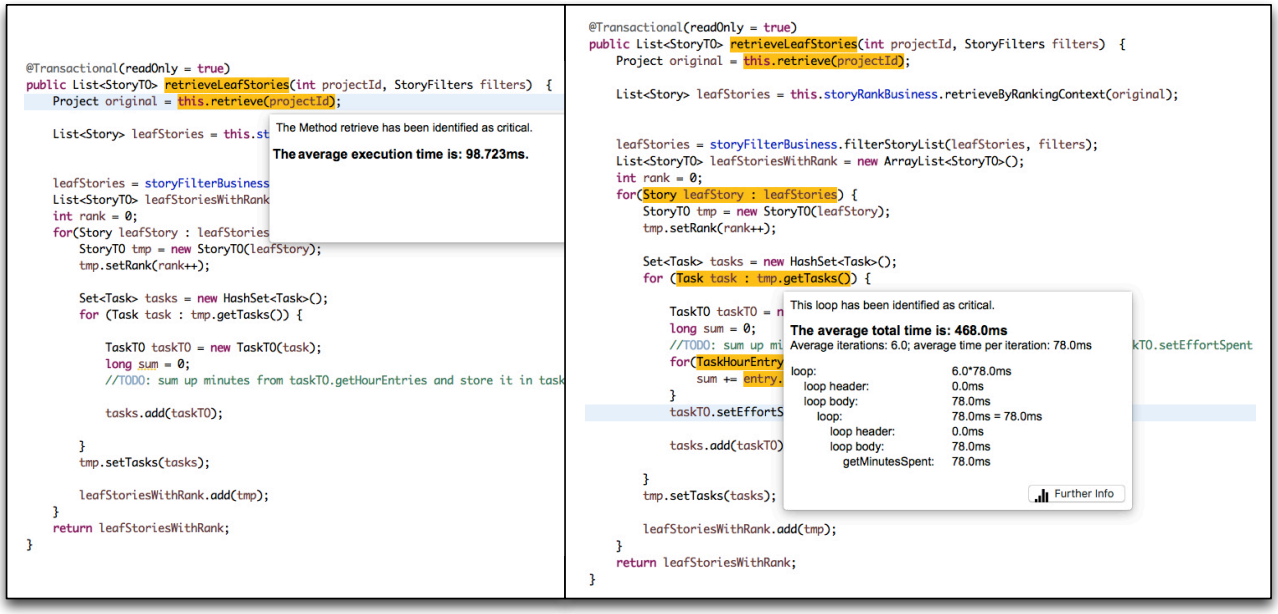
Theory. The basic theory underlying the conceptual framework is that static information available while working on maintenance tasks in software development does not suffice to properly diagnose problems that occur at runtime. The conjecture then is: Augmenting source code with dynamic information from the runtime yields enough information to the software developer to (1) make informed, data-driven decisions on how to perform code changes, (2) perceive code as it is experienced by its users.

Feedback Mapping. In an initial step, the abstract syntax tree (AST) of the source code is combined with the dynamic view of runtime information in a process called feedback mapping [2]. In the IDE, this results in a performance augmented source code view, that allows developers to examine their code artifacts annotated with performance data from runtime traces from production. Examples for such a mapping go from method calls with execution times, usage statistics for features or collections with size distribution.

Impact Analysis. When working on maintenance tasks, software developers often add or delete code. Nodes in the AST of newly added code does not yet have runtime traces that can be mapped. An impact analysis model designated for a specific purpose (e.g., performance prediction) is given the delta between the annotated AST, with all its attached operational data, and the current AST that include the new changes as parameters to infer new information about the unknown nodes in the current AST. A prediction through impact analysis gives early feedback and gives developers

PX/17, Brussels, Belgium

978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn



(a) Displaying Operational Footprint **(b) Inferring the Performance of Newly-Written Code**

Figure 2: Eclipse plugin *PerformanceHat*, an implementation of the conceptual framework of augmenting runtime information to source code

an outlook to the future of their code changes prior to running the application in production.

2 GOALS

Our approach follows three high-level goals:

(1) Operational Awareness: By integrating runtime aspects into source code and, thus, into the development workflow, software developers become more aware of the operational footprint of their source code.

(2) Contextualization: Software developers struggle with incorporating runtime aspects, because information about operational aspects (i.e., runtime metrics) is usually presented on the wrong abstraction level (i.e., visualized in dashboards) [1]. Our approach brings runtime metrics into the context of software development tasks by being integrated into the change process.

(3) Prevention: By achieving operational awareness and contextualization alone, we envision to already prevent introducing certain classes of runtime problems. Through impact analysis where we leverage lightweight, analytical models to provide inference for code changes, we want to avoid additional problems from reaching production.

3 PROOF-OF-CONCEPT IMPLEMENTATION

Based on the conceptual framework, we implemented a proof-of-concept as an Eclipse IDE plugin for Java programs, that implements the idea for software performance in cloud runtimes. Performance information (i.e., execution times) is attached to method definition and method calls. Currently, adding method calls and loops within a method body are supported for impact analysis and prediction. Figure 2 provides a screenshot of the implementation, called *PerformanceHat*[1].

4 DEMONSTRATION

In this demo, we explored the ways runtime information can aid the software development process. We demonstrated the capabilities of our implementation *PerformanceHat* on a case study application (AgileFant[2]). We show how software developers can become more aware of the performance of existing methods and can infer information about new methods through impact analysis. Figure 2 illustrates this process: (a) shows a method that is augmented with runtime performance information, providing software developers with an immediate sense of the method's operational footprint, (b) depicts how impact analysis propagates the information of newly written code and infers the impact of these code changes.

REFERENCES

[1] Jürgen Cito, Philipp Leitner, Thomas Fritz, and Harald C. Gall. 2015. The Making of Cloud Applications: An Empirical Study on Software Development for the Cloud. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, NY, USA, 11.

[2] Jürgen Cito, Philipp Leitner, Harald C Gall, Aryan Dadashi, Anne Keller, and Andreas Roth. 2015. Runtime Metric meets Developer - Building better Cloud Applications using Feedback. In *Proceedings of the 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2015)*.

[1] http://sealuzh.github.io/PerformanceHat/
[2] https://www.agilefant.com/