Quality Assessment in DevOps: Automated Analysis of a Tax Fraud Detection System

Diego Perez-Palacin
diegop@unizar.es
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain

Youssef Ridene
y.ridene@bluage.com
Netfective Technology
32 Avenue Leonard de Vinci
Pessac, France, 33600

José Merseguer
jmerse@unizar.es
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain

ABSTRACT

The paper presents an industrial application of a DevOps process for a Tax fraud detection system. In particular, we report the influence of the quality assessment during development iterations, with special focus on the fulfillment of performance requirements. We investigated how to guarantee quality requirements in a process iteration while new functionalities are added. The experience has been carried out by practitioners and academics in the context of a project for improving quality of data intensive applications.

Keywords

DevOps; Model-based QoS; Tax Fraud detection; Data Intensive Applications; Unified Modeling Language (UML)

1. INTRODUCTION

Building a service-time effective and reliable Data-Intensive Application (DIA) consists in finding a suitable combination between a varieties of frameworks that meet requirements. Besides, without a careful design by developers on the one hand, and an optimal configuration of frameworks by operators on the other hand, the quality of the DIA cannot be guaranteed. In fact, in a DevOps process developers build and test the application in an isolated, so-called, development environment, while operators are in control of a targeted run-time environment. Conceptually, the latter comprises all entities that will interact with the program: infrastructures, operating systems, Web servers, software agents and individuals. It is the responsibility of operators to guarantee that every framework needed by the DIA is installed in the production environment and properly configured for an optimized performance.

Within the DICE European project [3], Netfective Technology started building a solution (Big Blu) to demonstrate the capabilities of Big Data in e-government applications, especially for Tax fraud detection. Because of the complexity of Big Blu, we have a plethora of requirements to consider while building our solution. One of this most primordial requirement is that the whole system must be performance-effective, especially in terms of service throughput and resource utilization, because we are targeting data stores behind applications that manage country wide invoice records, tax payments or refunds (so billions of records of heterogeneous information), just to mention a few of the tax-involved categories. Moreover, many questions have to be addressed at early stage: How to design a reliable Big Data architecture? How could all the quality requirements be satisfied?. For example, given a set of requirements for a tax agency, we can identify and create optimal architecture or evaluate alternatives and measure the impact of business logic changes. What architecture to adopt keeping in mind the future evolution of the system?

DICE project researches towards building a quality-driven framework for development, deployment, monitoring and continuous improvement of DIA. DICE accommodates DevOps practices for simplifying the Devs and Ops activities in DIA through this quality-driven framework. The idea of quality assessment in DICE is to reduce the number of DevOps iterations, by assessing quality properties of software designs before construction. This paper investigates such an idea through its application in the Big Blu system. Our investigation uses a simulation tool (SimTool), developed within DICE, for quality analysis.

The rest of the paper is organized as follows. Section 2 presents SimTool. Section 3 presents Big Blu. Section 4 reports a couple of experiences carried out using SimTool for developing Big Blu. Section 5 concludes the work and revises some related work.

ICPE '17 Companion, April 22–26, 2017, L'Aquila, Italy.

ISBN 978-1-4503-4404-3/17/04...$15.00

DOI: http://dx.doi.org/10.1145/3053600.3053632

2. AUTOMATING QUALITY PRACTICES

Within the DevOps practices, we allocate the quality assessment actions in between the design and code implementation stages. SimTool contributes towards automatizing these actions. Concretely, it computes performance metrics (throughput, utilization and service response time) and a reliability metric, the service time to failure. SimTool fits within the DICE IDE toolchain, which defines the complete DevOps workflow for a DIA, to effectively accomplish the quality assessment in a Dev iteration.

2.1 Typical Usage Scenario

SimTool targets to enhance the quality of the code after each DevOps iteration. The objective is to reduce the

number of iterations needed to fulfill the required quality of the product. Hence, the quality in each Dev iteration is more likely to be accepted in its respective Ops iteration, i.e., it is reduced the number of quality issues found during monitoring in the real production environment.

A typical usage scenario of the SimTool happens when a new Dev cycle starts, the addition of a *new functionality* is planned, and monitored information from the Ops is received. The overall system, including the new functionality, needs to satisfy quality requirements, which were already defined or updated in the current iteration. Using the monitored information, the quality parameters in the design models, e.g, the actual host demands, are automatically updated. Moreover, the developers update the educated guesses used in the previous iteration.

However, it is unclear for developers the required system quality considering the single new functionality. In fact, the requirements refer to the overall system quality, therefore the developer has no clue for synthesizing quality requirements based on the new functionality. Using SimTool, the developers carry out a *sensitivity analysis*[1] to discover the expected overall system quality based on assumptions about the quality that the new functionality can offer. For instance, developers will predict the system response time considering different execution times for the new functionality, but according to the actual host demands for the already monitored software components. Without SimTool, the developer would remain clueless about the quality expectations introduced by the new functionality.

Consequently, SimTool helps developers to produce a piece of software that contributes to satisfy the overall system quality when it executes in the production environment. The developer has now an asset to assess the quality of the developed piece of functionality before notifying the next stage of the toolchain. Quality properties can now be checked through tests performed over the design of the single new functionality, instead of needing to wait until acceptance or system tests, or in a worst case to wait to the production environment to realize that the overall system quality does not satisfy the requirements.

2.2 SimTool Architecture

SimTool implements, within the DICE toolchain, the usage scenario above described. Therefore, the tool takes as input, for the current Dev iteration, a DICE profiled[2] design model. Then, it computes the aforementioned quality metrics for an assessment of the new DIA functionality.

A high-level architectural view of SimTool is depicted in Figure 1. The following paragraphs describe the responsibility of each component, inside SimTool, for carrying out the metrics computation.

The *Simulator GUI* component, firstly searches in the UML [15] model for the input variables (e.g., routing rates or execution host characteristics) and the quality metrics defined. Secondly, the user is allowed to *configure* the model, i.e., to introduce the actual values for the variables and to select the subset of metrics to compute. Thirdly, the component launches a *simulation* of the configured model, i.e., a computation of the quality metrics. Finally, when the *simulation* finishes, it displays the results for the metrics (`view-results` operation).

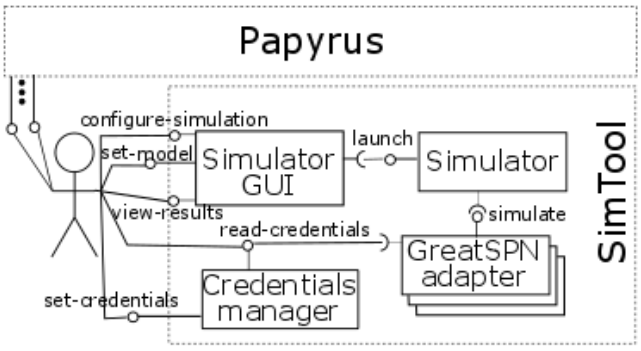

Figure 1: Component and connector view of the SimTool

The *Simulator* component offers a `launch` operation to the *Simulator GUI* for the actual simulation of a configured model. The *Simulator* selects the simulation engine based on the characteristics of the model and the chosen metrics, e.g., a Petri net simulator for performance or a fault tree analysis engine for reliability. It implements M2M transformations from the UML design to the target formalism, e.g., to the PNML[3] [10] format for Petri nets.

The *GreatSPN adapter* component follows the *Adapter* design pattern [7] to implement a `simulate` interface, which is the expected one for the *Simulator* component. It transforms the PNML [10] into the GreatSPN [5] Petri net file format. The GreatSPN engine simulates the model and returns the results.

The *Credentials manager* component stores credentials for connection to different simulation engines allocated through different servers. We decided this separate component for managing credentials, rather than setting them in the *Simulator*, because credentials are used by several components implementing the `simulate` interface, and the used component is chosen at runtime.

3. A FRAUD DETECTION DIA

Tax frauds represent a huge problem for governments, causing them a big loss of money each year. The European Union has estimated the fiscal loss lost due to tax evasion to be of the order of 1 trillion euros [4]. Governments are increasingly using Big Data in multiple sectors to help their agencies manage their operations, and to improve the services they provide to citizens and businesses. In this case, Big Data has the potential to make tax agencies faster and more efficient. However, detecting fraud is actually a difficult task because of the high number of tax operations performed each year and the differences inherent in the way the taxes are calculated.

Netfective Technology has started building a Tax fraud DIA, called Big Blu. As stated in the Introduction of the paper, Big Blu requirements cannot be meet without the joint contribution of developers and operators. Indeed, even if it is well programmed, the DIA may fail to reach its

[1]The sensitivity analysis allows to obtain multiple results for the metric by considering multiple input values.

[2]The DICE profile [8] is based on the standard MARTE profile [14] for performance and on the DAM profile [2] for reliability.

[3]Petri Net Markup Language.

functional or non-functional goals due to failures, problems, or incorrect behaviors occurring in the production environment. The core idea of DevOps is to foster a close cooperation between the Dev and Ops teams. In our context and through the adoption of the SimTool, we built this required cooperation thanks to shared UML models. They serve as contracts where every expected aspects of the production environment are clarified and specified in terms of SLA (service level agreements), time requirements or other quality requirements. We have designed UML DICE-profiled diagrams, see Figures 2 and 3, for this purpose.

Following a lean approach [12], Netfective built a simple but realistic plan for the development of the solution (initiated 12 months ago). First, we built a rapid prototype to validate the whole approach around focus groups and internal requirements gathering. Second, we built an MVP (Minimum Viable Product) respecting the 80/20 rule which means 80 percent of the expected results must come from 20 percent of efforts. This MVP has been developed, deployed and tested on a private Cloud relying on an agile DevOps approach. The third and last step of our plan is to move from an MVP to a first release with a stable and scalable architecture which can be deployed on a production environment. This step will be carried out during the next months. The current version of the MVP is based on Java and Scala as programming languages above a Cassandra [9] and Spark [16] clusters respectively for data management and data processing. The global solution is made of 3 main parts:

- the graphical user interface which is a web application developed using AngularJS 2
- the DIA itself
- And a restful webservice which makes the glue between the user-interface and the DIA

This application is deployed on a private Cloud and tested on a data set containing about 10 million taxpayers with all related details (the data were generated in order to avoid any privacy and confidentiality issues). The application is continuously running above the Cassandra databases which are filled with taxpayers detail, historical tax declarations, etc. The application performs computation on all data including new generated inputs (simulation of user inputs and details coming from various sources). These data have to be processed using Fraud Indicators (FI) which are a set of rules described by a domain expert and are known fraud behaviors. In the case a recently added FI, we have to proceed to a new batch processing phase on all the data. But we need to be able to answer any real time query using a merge between old batch results and new real-time computations. This architecture is a concrete application of the Lambda Architecture [13]. The user will be notified on the graphical user interface with the taxpayers who may be fraudulent. It is also possible to gather statistics on fraud results. The three layers of the solution are presented by the profiled activity diagram in Figure 2. In the first row, we can identify the various possible user interactions such as the selection of the fraud indicator to be applied for the detection, the exploration of the results or the monitoring of the status of a launched detection. These interactions are managed at the layer of the middle which sends the "orders" to the DIA to execute the underlying Spark jobs.

The UML models in Figures 2 and 3 were annotated with the input performance information needed to carry out the metric computation. However, since these values are not visible in the figures, we report them in Table 1. Moreover, the performance requirements are: the path that executes `Launch fraud detection` activity in Figure 2 should have a response time lower than 10 minutes; the rest of paths should have a response time lower than 10 seconds.

Probabilities. *Prob. of...*		**Workload**	
Fraud indicator creation	0.11	1 request every 600s	
Starting fraud detection	0.5	**Resources.** *Num. of...*	
Stopping fraud detection	0.02	Web Server	1
Getting fraud detection status	0.12	Big Data Management	1
Getting fraud detection results	0.25	Big Data Processing	1

Activities Service Times	
Request fraud indicator creation	2s
Select fraud indicators	3s
Request fraud detection stop	3s
Request fraud detection status	1s
Request fraud detection results	2s
Create fraud indicator	3s
Submit fraud detection	2s
Stop fraud detection	1s
Get fraud detection's status	0.1s
Get fraud detection's results	3s
Store analysis results	20s
Launch fraud detection	150s

Table 1: System properties annotated in the UML models

4. EXPERIMENTATION

As indicated, the MVP product was developed, deployed and tested by Netfective following DevOps. Next subsections report a couple of experiences conducted for assessing quality during two iterations.

4.1 Quality Issues

In a given iteration, the Ops team reported saturation of the `Big Data processing` computational node, see Figure 3. In particular, response time for the computational branch initiated by `Starting fraud detection`, see Figure 2, was required to be completed in 10 minutes, however this was largely surpassed.

The Dev team realized, by inspecting monitored information, that the issue could be due to some events:

- The growth in the number of requests to the system.
- The growth of the database.

Any of these situations -and, thus, also their combination- would cause that requests to `Launch fraud detection` activity arrived more frequently than they could be served by the `Big Data processing` node. Therefore, the requests would be waiting in an increasing size queue.

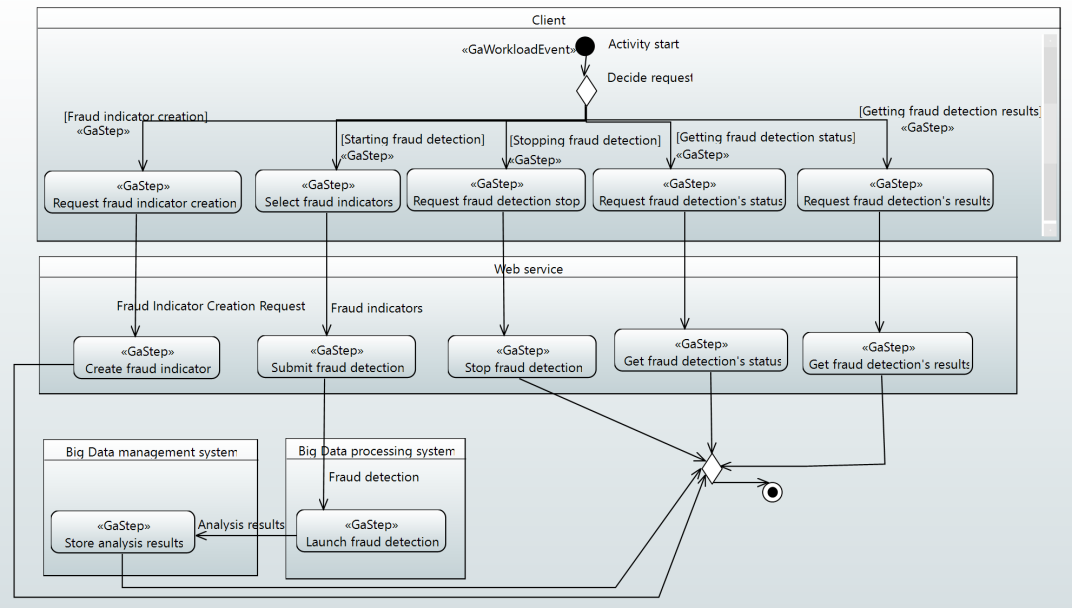

Figure 2: Execution scenario of the fraud detection DIA

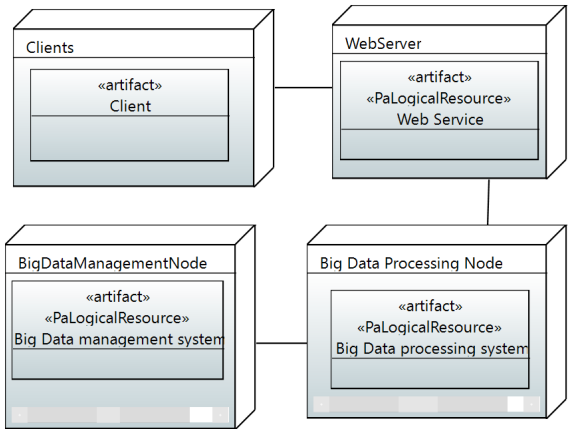

Figure 3: Deployment diagram of the fraud detection DIA

Using SimTool, we tried to confirm our guess by simulating increments in the arrival rate of requests to the processing node, and also increments in the execution time required by `Launch fraud detection`. Concretely, we simulated the arrival rate from 0.0016 req/s until reaching system saturation and the execution time from 2.5 minutes to 5 minutes.

Figure 4(a1) depicts the simulation results. Firstly, note that we have depicted a yellow plane at response time value of 600 seconds, for the reader to easily check whether the requirement is fulfilled or not, i.e., if it is below to 10 minutes or not. For the initial configuration (2.5 minutes of execution time and 0.0016 req/s), the system was stable and the response time was 3 minutes and 46 seconds. If only the arrival rate is increased, up to 3 requests every 10 minutes, then the response time is 8 minutes and 45 seconds. If only the execution time increased, up to 5 minutes, then the response time is 10 minutes and 25 seconds. If both variables increase (5 minutes of execution time and 3 requests every 10 minutes), then the response time increases unlimitedly[4].

Fig. 4(a2) studies the utilization of the `Big Data Processing` node with respect to our two variables. We observe that, for an arrival rate of 3 requests every 10 minutes and an execution time of 3.5 minutes, the node is almost saturated, with a 97.8% of utilization. When execution times are above 3.5 minutes, then the node is saturated, at 100% of utilization.

In the subsequent Plan stage of the new Dev cycle, developers faced two alternatives:

- Acquire new nodes in the private Cloud devoted to the `Big Data Processing`, so to parallelize the requests to the `Launch fraud detection` activity.
- The reengineering of the `Launch fraud detection` activity to make it faster, they had already observed that some parts of the code could be refactored.

The second alternative could easily reduce the service time by 40%, although going further this reduction would require much work. We used again SimTool to deeply analyze the first alternative and a combination of both.

Figure 4 (b1) depicts the results of the first alternative. It shows that using 3 nodes, the expected response time is between 6 and 7 minutes. In turn, with 4 nodes, the system will be able to satisfy more than 6 request every 10 minutes. Part (c1) also considers the second alternative by refactoring of the activity down to 3 minutes, i.e., an improvement of 40%. Then, using 2 nodes, we could offer an expected response time between 4 and 5 minutes. In turn, using 3 nodes, more than 8 requests every 10 minutes could be served. Parts (b2) and (c2) depict the utilization of the nodes for the same alternative solutions as (b1) and (c1), respectively.

[4]Figure 4(a1) does not show graphically this tendency to infinite values as it is cut at the response time value of 1000.

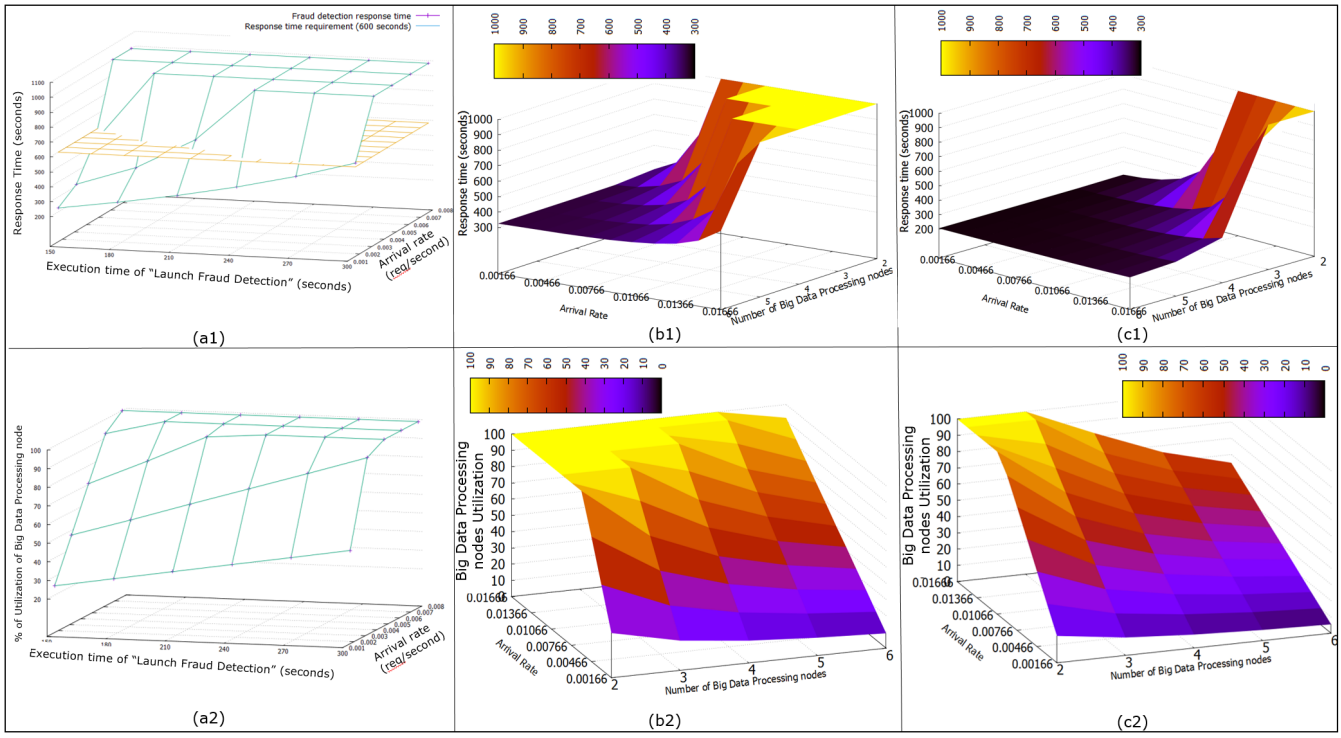

Figure 4: SimTool results for the "Quality Issues" DevOps experience

The decision was to use 3 processing nodes, which should satisfy the response time requirement.

4.2 Adding New Functionality

The experience here reported corresponds to the scenario described in Subsection 2.1. In particular, the new functionality is an API, that will be configured to be automatically and periodically called by clients. The API provides volatile information of interest for all clients. Expectations are that each client will make an automatic request to the API every minute, and currently there are 200 clients. Moreover, an API method is expected to execute, exclusively in the Web Server node, at a very fast speed, between 10 and 20 milliseconds. However, such interval is not granted yet, then the developers want to be assessed with an upper limit for it. Hence, they could themselves assess whether the functionality performance is acceptable already in the unit tests of the functionality.

First, engineers extended the UML design to consider the new functionality in the activity diagram. Later, they used the SimTool to predict the activity response time within the whole system.

Figure 5(a) ensures that the current deployment can satisfy the system response time requirement, even when executing the new functionality. Even having a 20 ms of execution time for the new functionality and doubling the expected arrival rate of requests, the expected response time is below 60 ms.

Figure 5(b) shows results to understand the limits of the current deployment depending on the achieved execution time of the new functionality. From this simulation, we obtained that, for the current workload expectation of 200 requests per minute, the new functionality could take up to 260ms of execution time without violating the response time requirement. This obtained value is given to developers, to be checked in their unit tests. Looking at both the response time and utilization charts in Figure 5(b), engineers also use the value of 160ms of execution time of the new functionality to set an alert in the monitoring of the application. Under the current assumptions, if the new functionality goes over such value, the expected response time would be 0.4s and the `Web Sever` utilization would go over 50%. So, when the alert goes on, although the requirement should still is satisfied, the engineers may want to study how the system is behaving and might take some actions for the next DevOps cycle, such as creating a cluster of `Web Server` nodes.

5. CONCLUSION AND RELATED WORK

This work has reported an experience on the usage of a software quality evaluation tool during a DevOps-oriented software development. The application under development is Big Blu, a Tax Fraud Detection system. We presented two common scenarios during development. In particular, during the DevOps planning and software creation stages. The purpose was to reduce the number of DevOps iterations until satisfactorily completing a modification of the system. This goal was achieved by reporting during the Dev stages the expected quality of the application.

Currently, SimTool is incorporating specific characteristics of Big Data technologies, which will help on the evaluation of alternative designs for DIA. In fact, Big Blu foresees the use of technologies of this type, such as Cassandra or Spark.

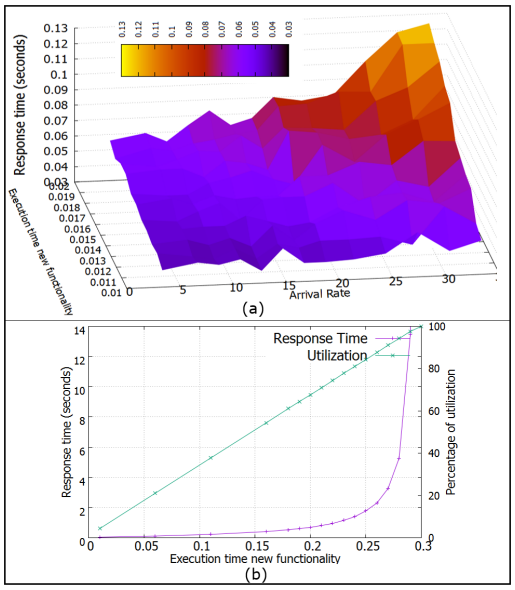

Figure 5: SimTool results for the "Adding New Functionality" DevOps experience

Related Work.

The Software Performance Engineering community has traditionally brought theories and tools to facilitate the quality evaluation of software systems. Recently, this community has started paying attention to DevOps approaches, and to how software products developed under DevOps practices could benefit from diverse quality evaluations along their development life cycle. A research agenda towards the application of performance engineering approaches in DevOps developments has been defined [1]. Work in [6] proposes a model-based evaluation of software with Continuous Delivery in every build right after their acceptance tests. However, SimTool is intended to be used in earlier stages of the development cycle, and therefore it can guide the coding phase, while it does not give feedback to the developer after the code commit is done, as [6] is able to do. The approach in [11] also aims at reducing DevOps cycles, by proposing a continuous performance evaluation of the application, including monitored performance information.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation framework programme under grant agreement No. 644869 (DICE). José Merseguer has been supported by the Spanish Government (*Ministerio de Economía y Competitividad*) under project No. TIN2013-46238-C4-1-R and The Aragonese Goverment Ref. T27 – DIStributed COmputation (DISCO) research group.

6. REFERENCES

[1] A. Brunnert et al. Performance-oriented DevOps: A research agenda. Technical Report SPEC-RG-2015-01, SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), Aug. 2015.

[2] S. Bernardi, J. Merseguer, and D. Petriu. *Model-driven Dependability Assessment of Software Systems*. Springer, 2013.

[3] Casale, G. et al. DICE: Quality-driven Development of Data-intensive Cloud Applications. In *Proceedings of the Seventh International Workshop on Modeling in Software Engineering*, pages 78–83, NJ, USA, 2015. IEEE Press.

[4] E. Commission, 2017. http://ec.europa.eu/taxation_customs/fight-against-tax-fraud-tax-evasion/a-huge-problem_en.

[5] Dipartamento di informatica, Universita di Torino. GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets, Dec., 2015. URL: `www.di.unito.it/~greatspn/index.html`.

[6] M. Dlugi, A. Brunnert, and H. Krcmar. Model-based performance evaluations in continuous delivery pipelines. In *Proceedings of QUDOS'15*, pages 25–26, New York, NY, USA, 2015. ACM.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, Nov. 1994.

[8] A. Gómez, J. Merseguer, E. Di Nitto, and D. A. Tamburri. Towards a uml profile for data intensive applications. In *Proceedings of QUDOS'16*, pages 18–23, New York, NY, USA, 2016. ACM.

[9] E. Hewitt. *Cassandra: The Definitive Guide*. O'Reilly Media, 2010.

[10] ISO. Systems and software engineering – High-level Petri nets – Part 2: Transfer format. ISO/IEC 15909-2:2011, Geneva, Switzerland, 2008.

[11] J. Kroß, F. Willnecker, T. Zwickl, and H. Krcmar. Pet: Continuous performance evaluation tool. In *Proceedings of QUDOS'16*, pages 42–43, New York, NY, USA, 2016. ACM.

[12] D. Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[13] N. Marz and J. Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[14] OMG. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1, Juny 2011. URL: `http://www.omg.org/spec/MARTE/1.1/`.

[15] OMG. *Unified Modeling Language: Infrastructure and Superstructure*, May 2011. Version 2.4, formal/11-08-05.

[16] J. G. Shanahan and L. Dai. Large Scale Distributed Data Science using Apache Spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15*, pages 2323–2324. ACM, 2015.