DevOps Performance Engineering: A Quasi-Ethnographical Study

Giuseppe Vergori
Politecnico di Milano
Milan, Italy
giuseppe.vergori@polimi.it

Damian A. Tamburri
Politecnico di Milano
Milan, Italy
damian.tamburri@polimi.it

Diego Perez-Palacin
University of Zaragoza
Zaragoza, Spain
diegop@unizar.es

Raffaela Mirandola
Politecnico di Milano
Milan, Italy
raffaela.mirandola@polimi.it

ABSTRACT

DevOps is a software engineering strategy to reduce software changes' rollout times by adopting any set of tactics that reduce friction in software lifecycles and their organisational variables, for example: coordination, communication, product evolution, deployment, operation, continuous architecting, continuous integration and more. Going DevOps is increasingly demanding that software engineering disciplines which were typically product-oriented such as software performance engineering to rethink their typical comfort zone, enlarging their scope from product, to process or even further to ecosystem and organisational levels of abstraction. This article makes an attempt at understanding what are the dimensions in DevOps organisational scenarios that can be addressed with a performance engineering lens. To do this, we performed a quasi-ethnographical study featuring a real-life industrial DevOps scenario. Discussing our results we conclude that many synergies exist between DevOps and performance engineering each with peculiarities, limitations and challenges - more research is needed to offer a full-spectrum performance-engineering support for DevOps practitioners.

Keywords

Software Performance Engineering, DevOps, The Phoenix Project

1. INTRODUCTION

Quoting from Bass et al.'s definition, DevOps is "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality" [2]. The wide spectrum of software practices entailed in this definition demands typically product-oriented disciplines such as Software Performance Engineering (SPE) [20] to undergo a radical shift of scope from product- to process-, organization- and even software ecosystems perspective. For example, from a DevOps perspective, it is no longer sufficient to model a queuing network [23] for a a software product to evaluate overall software performance. Conversely, that software performance model needs to be evaluated against the times and variables entailing integration and re-deployment of improvements. Therefore, a "classical" queuing network would need enriching with "organizational" rates (e.g., how often are the components in the network integrated, deployed or refactored) or even how long shall a certain refactoring demand (e.g., how many tasks need completion before re-deployment).

A first step in this shift of focus is identifying which variables and dimensions need to be considered along with the familiar notations and variables typically involved in SPE. This paper makes an attempt at identifying these dimensions using a quasi-ethnographical study. Ethnography is, by definition, "the organised study of other groups of people and is commonly associated with anthropological studies of other cultures [...]" [11] - this form of study usually involved direct participation in organizational and socio-technical processes involved [22]. Conversely our case features *quasi-ethnography*, an instance of ethnography where a qualitative dataset is generated using a second-hand report of real-life scenarios. In our case, we analysed *The Phoenix Project* [13], a case of industrial DevOps adoption, i.e., the migration from a classical software engineering approach to a DevOps oriented way of working.

Analysing our dataset we observed that SPE can play a key role in improving performance not only of software products but also on the (software-driven) organizational, ecosystem and lifecycle dimensions behind, for example: (a) planning continuous architecting/refactoring by prioritising on more easily (re-)deployable components; (b) continuously evaluating refactoring costs; (c) carrying out cost vs. performance driven continuous deployment and more. However, at present, a straightforward application of SPE techniques is not possible. Towards this end, we launched a research initiative aiming at understanding how models, parameters and techniques in SPE can be applied to understand and model in the most appropriate way the DevOps process.

The rest of this paper is structured as follows. First, Sections 2 and Section 3 outline the research design behind our

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

DOI: http://dx.doi.org/10.1145/3053600.3053628

quasi-ethnographical study. Second, 4 outlines DevOps as well as "The Phoenix Project", the ethnographical report we considered in the scope of this paper. Further on, Section 5 and 6 provide our attempt at mapping SPE notations with our quasi-ethnographical observations and discuss this preliminary mapping, highlighting threats to validity and outlining a research roadmap. Finally, Section 7 concludes the paper.

2. DEVOPS IN PILLS

Several references are starting to emerge with the goal of explaining and elaborating DevOps as an organizational software engineering strategy. In essence the core idea behind DevOps is fairly simple: over long years of siloed software development (on one side) and software operations (on the other), a series of distances in socio-technical and organizational practices and tools has been flourishing between Dev- teams and -Ops teams. These distances make the fluent and coherent collaboration between the two silos practically difficult if not impossible. DevOps entails a series of socio-technical and organizational practices and tools that address and tackle these distances one by one until a fast, fluent and resilient organization is distilled.

Several socio-technical practices have been studied to date, e.g., the merging of Dev- and -Ops teams under the same banner and working roof or shifting operational concerns such as infrastructure design to be addressed by software developers themselves or even using software code, versioning, design patterns for infrastructure design itself [2].

Similarly, several organizational practices have been studied to date, e.g., using the same issue-tracking mechanism for both development and operation such that there shall be no difference between developer and operator at all or even reducing the software architecture to a number of modules (or microservices [1]) pretty much equal to the number of developers+operators present and available for that product such that each module can be developed and operated even by the same person [2].

The above are mere examples of the series of practices entailed in DevOps, with more tactics and strategies emerging almost every month. In our context, we argue that SPE is rich with practices that have been used to speed-up software performance; in this vein, the same tactics can be re-thought to address organizational and socio-technical performance issues emerging in software lifecycles.

3. RESEARCH DESIGN

This section outlines the research problem, questions and method behind the contributions in this paper.

First, the problem we address is that a DevOps perspective on software engineering forces several areas of research and practice (including software performance engineering) to undergo a shift of focus between a typical product-based orientation to a wider angle, which includes organizational (e.g., coordination cost, communication, etc.), socio-technical (e.g., skills-based task-allocation costs) as well as lifecycle and ecosystems aspects (e.g., continuous deployment). Although this problem touches much more than SPE, we argue that this discipline is rich with practices, models, methods and tools that may well be used to devise even more refined and advanced DevOps practices. To address this problem, we formulated two research questions:

1. *What software lifecycle variables are synergistic with SPE?*

2. *What SPE models and notations may be used to study these variables?*

With the goal of giving a preliminary answer to these research questions, we adopted qualitative empirical software engineering inquiry featuring quasi-ethnographical research. More in particular, instead of a classical ethnomethodological approach where timely reports are generated by direct observation of socio-technical and organizational phenomena, we chose to adopt indirect observational means, using the contents reported in [13] as ethnographical data (more details on the contents and structure are outlined later in Sec. 4).

Moreover, to analyse the above data we adopted content analysis [12] and software lifecycle modelling [9] techniques resulting in 13 simple and intuitive UML Sequence Diagrams that capture the observed organizational and socio-technical software lifecycle processes featuring DevOps, including the variables and dimensions we were originally looking for.

4. A QUASI-ETHNOGRAPHICAL STUDY IN DEVOPS: THE PHOENIX PROJECT

The Phoenix Project directly reports on the *persona* scenario for Mr. Bill Palmer neo-elected vice-president of company "Parts Unlimited" (PU). PU is developing a new revolutionary project codenamed "Phoenix", fundamental for the future of the company. Bill is essentially responsible for the migration from the previous way of working in PU to a DevOps-based software lifecycle strategy. The book contains detailed reports on how and by means of which practices and what concrete actions did the 150 employees employed by PU actually migrate to DevOps. Coding these reports and analysing the reports with content analysis [12] we were able to distill a series of 13 detailed sequence diagrams[1] that describe typical DevOps workflows in terms of:

- **Actors:** these reflect the typical roles and people involved in DevOps which are needed to enact the software lifecycle (e.g., developers/operators), improve that lifecycle (e.g., reporters, monitors, quality analysts) and coordinate the lifecycle itself (e.g., product owners);

- **Actions:** these reflect the typical tasks to be enacted as part of the typical DevOps way, e.g., allocating tasks, reporting incidents, adding or addressing issues in the reference issue-tracker, etc.;

- **Tools:** these reflect the typical tools that automate or accelerate the DevOps way - e.g., harmonised issue-trackers and version-control systems, continuous integration pipeliners, continuous deployment, infrastructure-as-code mechanisms, etc.;

- **Variables/Risks/Decision-Points:** these reflect the decision points or loops we distilled from the scenario

[1]a complete technical report (only in italian) on the case-study at hand and the tentative mapping proposed in this thesis is available online: http://tinyurl.com/h3xoalc

at hand - for example, "the Phoenix Project" also contains a detailed series of incident reports which can be considered typical in DevOps software product chains - the incident management procedures enacted in such instances need to be further investigated to isolate ways in which incident management can be sped up;

- **Measurable Quantities:** these reflect the quantities that actors in PU used to evaluate, weigh and eventually consent on a common decision - these quantities can be further investigated to understand trade-offs or typical barriers in decision processes as well as decision outcomes;

Figure 1, for example, outlines the process used to handle software outage. The figure outlines the procedure as an alternative of two sub-scenarios corresponding to *"Known Problems"* or *"Unknown Problems"*; in the former (top of the sequence diagram), employees emit corresponding help requests outlining the problem in an appropriate form and with corresponding problem information. Similarly, in the sub-scenario wherefore outages and connected problems are unknown, appropriate problem investigation ensues and solutions plans are devised with consequent task assignments.

The key idea we advocate in this paper is that this and similar lifecycle processes bear key variables and recurrences with respect to typical software engineering processes - these variables and recurrences along with the process behind them may be further studied and instrumented with SPE models, approaches and notations for improved DevOps speed.

5. SUPPORTING DEVOPS SCENARIOS WITH PERFORMANCE ENGINEERING

In order to evaluate the performance of the software development process following DevOps, we investigate whether we can inspire by already proposed Performance Engineering techniques. Concretely, due to our previous experiences in software quality evaluation, we focus on the reutilization techniques coming from the model-based SPE research field.

Therefore, we apply techniques of performance evaluation of a product to carry out the performance evaluation of a process. Fortunately, many times the performance of the Software product has been evaluated as the performance of its workflow; while the software development process can be also seen as a workflow of actions.

The successful application of a concrete model-based SPE technique to study the performance of DevOps processes is achieved if both the results offered by the SPE technique are meaningful in the DevOps processes domain and the inputs required by the SPE technique are obtainable in such DevOps domain. In this section we elaborate on this matching between usefulness and obtainability of the outputs and inputs, respectively, proposed SPE in the software development process following DevOps practices. In the scope of this mapping, our two main research questions can be rephrased as follows:

1. *Are the outputs offered by model-based SPE useful in the context of performance evaluation of the software development process?* This research question is shortened to *performance metrics matching*

2. *Are the inputs required by model-based SPE achievable in the context of performance evaluation of the software development process?* This research question is shortened to *input information matching*

We address these research questions in the following, using our analysis of the information available from *The Phoenix Project* [13].

5.1 Performance metrics matching

Metrics of interest when evaluating DevOps software development process are much close to software process improvement research, e.g., the work of [15] or [7]. Similarly, we report:

1. Expected task completion rate for a given assignment of developers to tasks. In this way, the best assignment of developers to tasks can be chosen;

2. Expected finishing time of the tasks associated to the current cycle. In this way, the deadlines for the current DevOps chain iteration can be established or renegotiated;

3. Probability of having finished all the tasks associated to the current cycle by its deadline. In this way, it can be done a risk assessment of the progress of the iteration;

4. Expected percentage of tasks finished by the end of the established iteration deadline. In this way, the percentage of completed tasks by the deadline or the deadline for all tasks can be renegotiated;

5. Expected finishing time of the whole development. In this way, the expected finishing time of the project can be devised. If the completion of the project is expected to happen before final delivery, some developers can be released and allocated to new projects. Otherwise, if the project is delayed, the presence of some functionalities or the final release date can be renegotiated. Also an attempt to satisfy the final delivery date by adding more developers to the project can be discussed [6];

6. Probability of having finished all the tasks of the project by its deadline. In this way, it can be done a risk assessment of the overall finishing date of the project and, if necessary, renegotiate the same topics as in point 5);

7. Proportion of time doing Dev activities and proportion of time doing Ops activities. In this way, some sort of "load-balancing" activity can take place between Dev- and -Ops sides of the lifecycle, working to optimise such times and reduce waste, if possible;

8. Frequency with which developers have to leave their task unfinished and move to another either due to incidents in the task under development that block it because the have to be solved by project management, or by interruptions coming from project managers to solve incidents caused by previous tasks;

Familiar metrics in SPE are the system response time, activity response time, system throughput, percentage of utilization of resources, residence times in resources. Moreover, in the study of response times, it is commonly necessary more information beyond its average or expected value. In such cases, they are provided percentiles or partial cumulative distribution functions of the response time. These

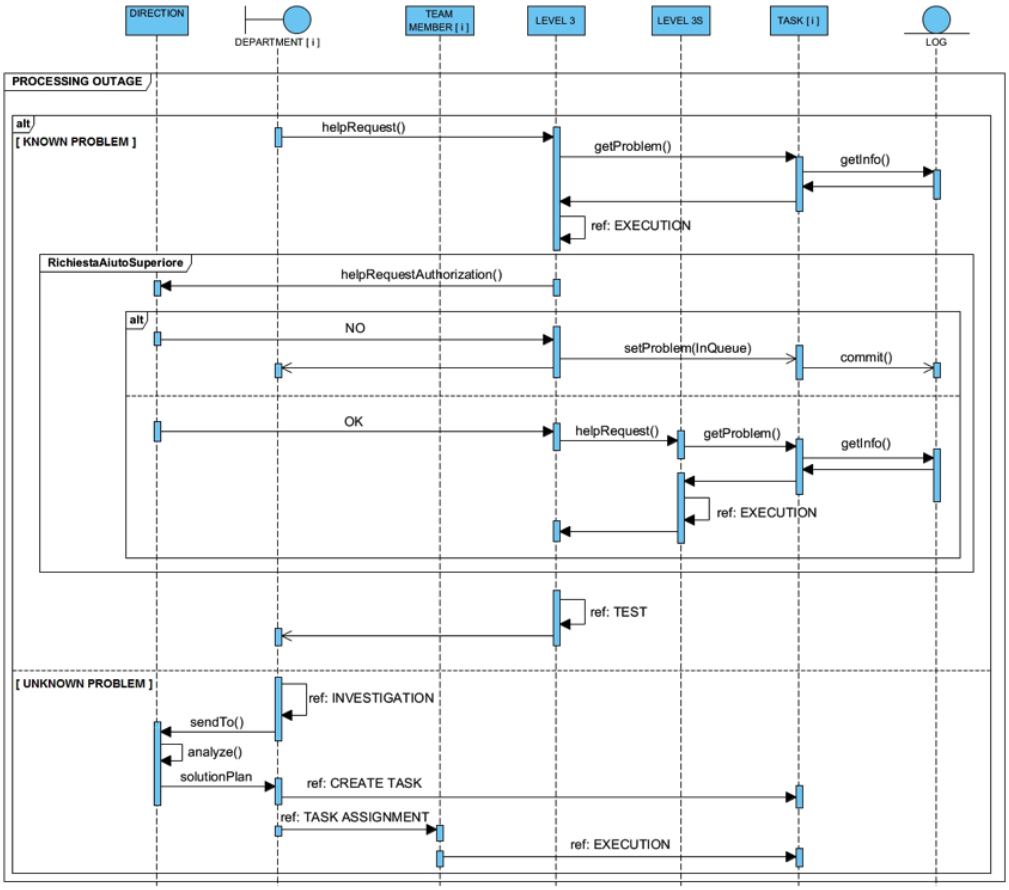

Figure 1: Example of DevOps process - handling software outages from our own experience report.

are expressed as $P(t < T)$, and their values help to answer questions in the software requirements domain such as *the system response time shall be lower than 2 seconds the 95% of times*.

These SPE metrics, for which there exists theories to calculate their values from system models, match the presented performance metrics of interest in DevOps software development. Concretely, we can see that: metric 1) can be calculated by computing the system throughput, 2) is analogous to expected response time of the development process, 3) can be calculated by computing the probability that the response time of the system is less than the remaining time until the deadline, 4) can be calculated by using the throughput (rate of tasks completion) and the deadline, 5) and 6) are calculated as 2) and 3) but using as input the information for the whole development instead of the information for the current iteration, 7) is similar to calculation of utilization of resources, 8) can be computed by using the relation between the sum of rates with which developers change the task in which they are working (throughput of start developments) and the system throughput.

Therefore, for the *performance metrics matching* research question, we have seen that SPE offer solutions for computing the software development performance metrics we have been able to elicit.

5.2 Input information matching

The inputs for model-based SPE are software models that represent the behavior of the system in one or more execution scenarios. These software models are built by software engineers and expressed in languages that they master as BPEL-WS [5] or the even more richer domain of UML [3, 10, 8, 18, 16] by applying profiles as MARTE[14]. Palladio [19] follows the Meta-Object Facility (MOF) standard and is a domain specific modeling language devised to model software architectures and ease their performance evaluation.

We are not in a position to solve this research question because we lack a comprehensive view of all the particularities that affect the good "health" of a software development process in a organisation. However, we hypothesize that, at least the fundamental activities and interactions of a DevOps software development process can be modeled with general purpose languages like UML. To keep it as plausible hypothesis or to refute it, we have done the experiment of modeling the software development process and interactions described in the *The Phoenix Project* case study with one of the input languages accepted by SPE, concretely with UML. Figure 1 shows one of the organisational scenarios, modeled with a UML Sequence diagram. As described in Section 4, the diagram represents the scenario of incidence management of a development task with respect to the involved teams and the task lifecycle. It is detailed the involvement of incidence management levels, as "Level 3" and "Level 3S" to help in *"known problems"*, and how they queue the diverse incidences that can exists in a given moment. It is also detailed how, when the team faces an *"unknown prob-*

lem", it is required a time consuming investigation whose output will produce new tasks that will solve the incidence. In turn, the implementation of these newly generated tasks will consume time of developers, their completion will have to satisfy delivery dates, and they will have to be scheduled and addressed in one of the DevOps cycles. The conclusion of this experiment has been that the generation of models in the input languages used by SPE such as profiled UML diagrams is a feasible work in the context of performance evaluation of a DevOps software development process, although we do not provide the full description of the study and their modeling to the obtained 13 UML Sequence Diagrams due to space restrictions,

6. DISCUSSION

This section discusses the main benefits of the approach and ideas contained in this paper, later touching on threats to validity and outlining a tentative research agenda around the ideas we proposed.

6.1 Discussions and Lessons Learned

First, we observed that the proposed idea leads to several forms of evidence-based insight curiously reminiscent of Joel Spolsky's Evidence-Based Scheduling[2]. In so doing, the idea of combining DevOps with SPE may lead to offering insights that fit equally well in projects, teams and organisations that already underwent a shift to agile methods or in their non-agile counterparts. In the former scenario, the discipline of DevOps performance engineering would help planning cycles and iterations or during retrospectives while in the latter scenario our approach would help proceeding the software lifecycle in a flexible way, i.e., choosing the next development/operation step in a way which is best-fitting with the evidence at had. Additional research needs to be invested on our approach first to understand the implications of this observation and second to understand if and how our approach (with its supporting tools and notations) may extend agile tools such as Trello or other tools typical in more classical approaches to software engineering.

Second, the idea reported in this paper suggests that SPE and DevOps are definitely synergistic when it comes to deciding the best-fit task allocation considering the various organisational forms in the software development resource-pool. For example, consider a scenario where several partner organisations share in a software development project (e.g., through outsourcing or using a mix of open- and closed-source software communities). An approach of DevOps performance engineering may take said organisational scenario in consideration, e.g., offering the possibility to adopt ad-hoc scheduling schemes or rules for the scenario in question.

In summary, from the sketch we provided in this paper, we could grasp that: (a) DevOps performance engineering shows the promise of possibly combining multiple well-established notations at the same time and for the key purpose of steering and governing software lifecycles in a DevOps fashion; (b) SPE notations may have non-trivial but intriguing interactions and may yield valuable insights when combined together (e.g., multi-view modelling and layered queuing networks that reflect multiple community types, multiple process models in the same DevOps product development effort); (c) DevOps performance engineering may even have the potential to lower considerably the organisational and socio-technical barriers [21] in DevOps.

6.2 Threats to Validity

Although offering a valuable lens of insight into DevOps and the synergies it bears with SPE, the work reported in this pages bears the threat to validity of any single-observation case-study. Also, the quasi-ethnographical nature of our study does not reflect a direct observation of the case-study object but rather a second-hand report which is also represented in a novelised form and enriched with many circumstantial and factual aspects reflecting business, organizational, social and socio-technical connotation. Although this nature allowed us to observe the very processes that we were originally looking for, this same nature cannot allow us to generalise the contents and contributions reported in this paper beyond the illustrative purpose we intended. That being said, the ideas and concepts we report serve the purpose of laying foundations to more work and further investigation around the research roadmap we defined. In providing these foundations, we hope that more researchers around the world will pick up the study such as we left it, possibly addressing some of the shortcomings we pointed out in this section.

6.3 Research Roadmap

From the premises and ideas outlined in the previous pages, there are several research directions for future work:

- First, more research shall be invested in the steps of **performance metrics matching** and **input information matching** described in Section 5. To this end, besides using existing data on DevOps processes, we plan to exploit our expertise both on SPE and on its application to software processes [4].

- To facilitate the SPE adoption, another research line we intend to investigate is the possibility of developing a **complete (semi-)automatic approach** encompassing the previous steps. For example, we observed that much of the required information concerning DevOps process practices is already existing in various task-based repo's widely used in industrial practice, e.g., bug-trackers such as JIRA databases, boarding tools such as Trello, or task-/issue-based repos such JIRA or MyLyn. Our idea is that this approach may be rigged to work in a proactive way by retrieving this necessary data automatically. Moreover, several Application Lifecycle Management solutions exist that may work well from interfacing with our approach. For example, tools such as TaskTop[3] that offer harmonisation layers across software development pipelines may feed task-based information into our approach and receive feedback in return.

- More research shall also be invested in **multi-view modelling.** We plan to investigate the use of different types of models according to (i) the global requirements, (ii) the development phase and (ii) the data availability. These models can be used in a stand-alone or in a combined way. For example, Petri nets

[2]http://www.joelonsoftware.com/items/2007/10/26.html

[3]http://tinyurl.com/j7mxxtg

and Queuing Networks models with the emphasis on the competition for the resource usage can be used for staffing purpose as well as for organizational issues. Bayesian networks can be helpful as a decision making tool helping the prioritization of tasks or module in the software architecture;

- Another direction in which more research shall be invested is **uncertainty analysis.** Several tasks at the initial development phases cannot be precisely defined so it is important including uncertainty management analysis techniques to take these aspects into account[17]. Specifically, we can distinguish two main research directions:
  - **model incompleteness:** additional research will focus on defining incremental approaches able to include pieces of information in the available models as soon as they are available;
  - **model analysis:** more research on our approach will focus on defining techniques able to infer undefined properties about module/tasks based on the overall requirements and the estimations obtained by the analysis of the available models;

7. CONCLUSION

DevOps is recently gaining momentum as a corporate software engineering culture and strategy that promises increased efficiency and reduce time to market. In our industrial practice and experience we observed several key synergies between DevOps and software performance engineering tenets and current challenges. In response to these synergies, this paper explores a second-hand ethnomethodological report over an industrial DevOps adoption campaign.

With this preliminary qualitative investigation in mind, we made an attempt at meshing DevOps and Software Performance Engineering featuring the (re-)use of available modelling notations and technologies which are currently very common in industrial practice (e.g., Petri-Nets).

The ideas proposed in this paper may lead to mechanisms that aid the industrial adoption of DevOps approaches by allowing intelligent steering of new software projects using a strategy ever closer to DevOps. Even in a primordial stage, the ideas reported in this paper show the great promise and several key interesting research paths existing behind combining software performance engineering with DevOps processes and organizational scenarios. For example, our approach may function as a DevOps governance mechanism rotating around previously existing modelling notations.

In the future, we plan to conduct more research in the general directions pointed out in our tentative research roadmap, with the key goal of further elaborating how and by means of which combination of modelling notations and tools from SPE may best support DevOps.

Acknowledgment

Some of the authors' work is partially supported by the European Commission no. 644869 (H2020 - Call 1), DICE.

8. REFERENCES

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, 2016.

[2] L. J. Bass, I. M. Weber, and L. Zhu. *DevOps - A Software Architect's Perspective*. SEI series in software engineering. Addison-Wesley, 2015.

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

[4] A. Bertolino, E. Marchetti, and R. Mirandola. Performance measures for supporting project manager decisions. *Software Process: Improvement and Practice*, 12(2):141–164, 2007.

[5] P. Bocciarelli and A. D'Ambrogio. Model-driven performability analysis of composite web services. In *SIPEW*, volume 5119 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2008.

[6] F. P. Brooks. *The Mythical Man-Month*. Addison-Wesley, Boston, MA, anniversary edition, 1995.

[7] B. M. Feeley. *IDEAL: A User's Guide for Software Process Improvement*. Software Engineering Institute (SEI), 1996.

[8] M. Fleck, L. Berardinelli, P. Langer, T. Mayerhofer, and V. Cortellessa. Resource contention analysis of cloud-based system through fuml-driven model execution. In *NiM-ALP@MoDELS*, volume 1074 of *CEUR Workshop Proceedings*, pages 6–15. CEUR-WS.org, 2013.

[9] R. B. France and B. Rumpe. Model-based lifecycle management of software-intensive systems, applications, and services. *Software and System Modeling*, 12(3):439–440, 2013.

[10] V. Grassi and R. Mirandola. Uml modelling and performance analysis of mobile software architectures. In M. Gogolla and C. Kobryn, editors, *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2001.

[11] M. Hammersley and P. Atkinson. *Ethnography*. Routledge, London, 2003.

[12] H.-F. Hsieh and S. E. Shannon. Three approaches to qualitative content analysis. *Qualitative health research*, 15(9):1277–1288, 2005.

[13] G. Kim, K. Behr, and G. Spafford. *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. IT Revolution Press, 1st edition, 2013.

[14] MG. Uml profile for marte: Modeling and analysis of real-time embedded systems, 2009.

[15] M. Niazi, D. Wilson, and D. Zowghi. Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice*, 11(2):193–211, 2006.

[16] D. Perez-Palacin and J. Merseguer. Performance sensitive self-adaptive service-oriented software using hidden markov models. In *ICPE*, pages 201–206. ACM, 2011.

[17] D. Perez-Palacin and R. Mirandola. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In *ACM/SPEC International Conference on Performance Engineering, ICPE'14*, pages 3–14, 2014.

[18] D. B. Petriu and C. M. Woodside. Software performance models from system scenarios. *Perform. Eval.*, 61(1):65–89, 2005.

[19] R. Reussner, S. Becker, J. Happe, H. Koziolek, K. Krogmann, and M. Kuperberg. *The Palladio component model*. Karlsruhe, 2007.

[20] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, Reading, MA, USA, 1990.

[21] D. A. Tamburri, R. Kazman, and H. Fahimi. The architect's role in community shepherding. *IEEE Software*, 33(6):70–79, 2016.

[22] D. A. Tamburri, P. Lago, and H. van Vliet. Organizational social structures for software engineering. *ACM Comput. Surv.*, 46(1):3, 2013.

[23] K. S. Trivedi and R. A. Wagner. A decision model for closed queuing networks. *IEEE Trans. Software Eng.*, 5(4):328–332, 1979.