Model-Driven End-to-end DevOps Engineering

Damian A. Tamburri*
Politecnico di Milano
Milan, Italy

Diego Perez-Palacin
Politecnico di Milano
Milan, Italy

Raffaela Mirandola
Politecnico di Milano
Milan, Italy

ABSTRACT

Generally speaking, DevOps is a software engineering strategy that entails deeper, lean, continuous, tool-supported and measurable collaboration between various roles of software engineering, e.g., planners (Product Owners), developers (architects or testers) and operators (infrastructure engineers). Although the benefits of DevOps are clear, there are no proven models and methods to bootstrap and steer software processes in a DevOps fashion. This vision paper illustrates EDEN that is a "End-to-end Devops ENgineering". EDEN is a model-based approach to plan and steer a software process in a DevOps way, inheriting well-established models, concepts and techniques from software engineering research, e.g., UML models, Function-Points, Petri Nets and Queuing Networks. This paper illustrates EDEN and elaborates a sample scenario. We conclude with a preliminary research roadmap.

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]. Even though in its early inception phase, DevOps has gained much attention in industry given its direct practical and tangible benefits (e.g., reduced deployment times, continuous improvement, etc.). However, much "anti-buzz" around DevOps is connected to the relative lack of precise understanding of what it means to "go DevOps", i.e., adopting DevOps as a software engineering strategy part of corporate culture and organisational governance [17]. Also, assuming this understanding may exist, in our direct experience with industrial adoption of DevOps we observed still much practitioner distrust and disillusion concerning DevOps, which is often still seen as "yet another strategy", mostly agnostic of previously existing approaches and methodologies with established best-practices and proven/measurable result.

*damianandrew.tamburri@polimi.it, diego.perez@polimi.it, raffaela.mirandola@polimi.it

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Essentially, industrial problems emerging from a series of industrial focus-groups we held in four big and well-established worldwide companies can be summarised as follows: (i) it is still very difficult to adopt DevOps while keeping "what's good" in the mix, thus avoiding waste; (ii) a need is growing for easing the DevOps transition and speeding up the change by enacting minimalistic changes in small steps; (iii) the trick behind DevOps adoption lies in reducing the pain and offering models/approaches in which the benefits of DevOps become more clear. From these points, we advocate the following: (a) model-driven approaches are agile enough to be used to abstract previously existing assets to speed-up and manage DevOps - for example models can be used to abstract previously existing deployment and configuration scripts; (b) model-driven approaches can be constructed incrementally and their refinement can be automated at will so as to speed the transition from classical to DevOps models - for example a single model such as a reference-architecture can be transformed into its typical deployment expression; (c) models and views [19] can seamlessly harmonise and represent DevOps information and tasks across stakeholders typical of DevOps scenarios [7, 9].

In response to these problems and assumptions, we formulated the following research question: *"What kind of modelling can aid DevOps adoption and engineering?"* This is where EDEN comes in. EDEN stands for "End-to-end Devops ENgineering" and is an approach that reuses previously existing industrial-strength modelling assets for the purpose of planning and managing a software engineering effort with a DevOps approach right from the very beginning.

At a glance, the idea behind the EDEN approach to DevOps engineering (i.e., software engineering in a DevOps fashion) is fairly simple. **First, produce a socio-technical software architecture model** - this model is used as a process view of software architectures, that is, an effort estimation and socio-technical organization (e.g., usable for work-division) product for both software architecture development and operations. For example, an architecture "draft" can be obtained from the typical model that software practitioners use to bootstrap a software development effort, i.e., a Function-Points (FP) model [10], typically used for cost and effort estimation. An FP estimation can be tailored to take also operations into account. Similarly, a socio-technical software architecture model can be inferred from Kanban boards [14] and their implementation in tools that support

Agile Methods such as Trello[1] or Asana[2].

Second, transform the socio-technical architecture model into DevOps-compliant process models - the socio-technical architecture can be transformed into a model that can be used to recommend task-allocations, e.g., using a well-established queuing network [18] or Petri net[8] model with or a series of production servants. For example, the information encoded in the socio-technical architecture models and the planned tasks therein (be it development or operations tasks) can be inserted and evaluated in a petri net to decide which task combination or assignment works better under known project assumptions (e.g., budget, time constraints, etc.). Note that this step can be made: automatic, by means of model-to-model transformations; transparent, by hiding the ad-hoc reasoning logic behind the much more familiar face of tools such as Trello; harmonious and in perfect alignment with Conway's law, that is, "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations" [5]. As a consequence, EDEN can be used as the automated use of Conway's law to enact DevOps *continuous analysis* - i.e., continuous DevOps process improvement by means of models that, on one hand, are used to plan efficient DevOps work-division and task needs (i.e., the socio-technical software architecture model), and on the other hand, to recommend which task-configuration is most consistent with the status.

Even in the layman's terms above, EDEN already shows great promise: (a) it would make less painful the industrial DevOps adoption by reusing well-established competences and skills in model-driven engineering research and practice - for example, estimating effort using FP models or drafting preliminary architectures using the same algorithms; (b) it would allow industries to think in a DevOps fashion - for example, by thinking in tasks and intermixing at the same time both Dev- and -Ops tasks right from the very beginning, at the effort estimation phase; (c) it would allow industries to combine agile task-based production methodologies (such as Scrum [15]) with established tools and models from software architecture research, since these models would be embedded directly in the organisational governance structure [17].

This vision paper elaborates further on EDEN concretising it around a reasonable research agenda. We observed that there are many established practices in software architecture research to be used within EDEN to realise a lightweight end-to-end software engineering approach [2].

2. EDEN EXPLAINED

Although DevOps is not only an approach towards collaborative software engineering, With EDEN we assume that DevOps can be summarised as the process in Fig. 1, whereby: (a) operational concerns become predominant at a much earlier stage, e.g., software engineering proposal writing, planning and cost estimation (top-most activity in Fig. 1); (b) the concept of task is the dominant abstraction in a DevOps way of working; (c) the tasks and activities, either connected to development or operational concerns, have a life of their own, i.e., they are opened, assigned, processed, closed, re-opened, etc.; (d) said tasks and activities co-evolve with their development counterparts, for example, if an operational task is re-opened, it is reasonable that a corresponding development task is issued; (e) from a modelling perspective, any key activity in a DevOps process (e.g., cost-estimation or software development) may be further elaborated into EDEN with more concrete modelling, e.g., the pop-out box in Fig. 1 shows a DevOps model elaborated by means of a Scrum lifecycle model. Following these assumptions, providing a DevOps engineering method means to support the activities summarised in the DevOps general process in Fig. 1. The figure shows more concretely the above assumptions in a UML-like activity diagram where activities are typical software lifecycles (e.g., Cost Estimation, Development, etc.) and flows are evolutions (e.g., edge 4A for testing, edge 4B for deployment & operation, etc.).

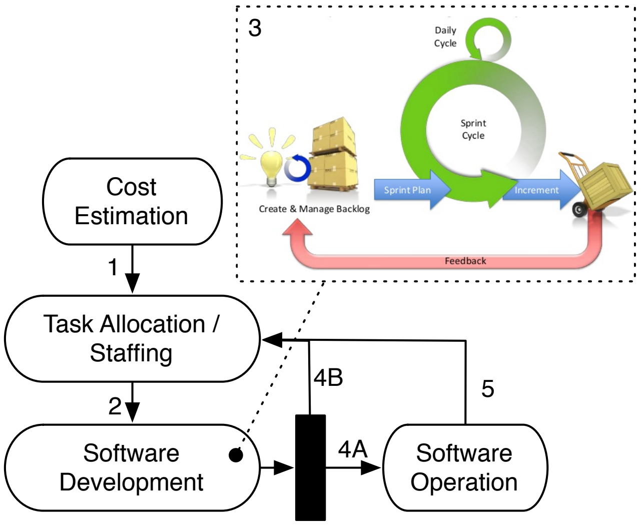


Figure 1: DevOps lifecycle.

Following Fig. 1, EDEN envisions:

1. Supporting **models where operations concerns are evaluated together with development ones** - for example, during the analysis necessary to apply an FP-based effort estimation approach, the complexity evaluation must take into account operational concerns that raise the difficulty class or expense connected to developing and operating a certain product or increase the total FP-count (at least) by a factor of two, i.e., by accounting both development and operation for envisioned software systems;
2. Supporting **dynamic task-allocation and staffing**, i.e., step 2 on Fig. 1, by means of well-established notations and analyses such as Queuing-Networks or Petri Nets - for example, using a combination of initial costs, effort predictions, EDEN can offer a baseline for process status monitoring and time-to-finish, e.g., by means of Petri Nets that illustrate the status of tasks and functions (taking into account the scheduling of task processing decided by project manager) together with an estimation of the expected time required to finish the process and percentage of process already completed; In addition, EDEN may be instrumented to use DevOps live information (see Step 5 on Fig. 1) to make recommendations;
3. Supporting **software development, testing and operation**, i.e., steps 3 (Scrum), 4A and 4B on Fig. 1, by means of additional recommendations, e.g., on which devel-

[1]https://trello.com/
[2]https://asana.com/

opment task combination is more efficient towards reaching a Potentially-Shippable Product [15] faster; For example, given a certain set of interdependent development tasks (e.g., as reflected by a series of requirements or user-stories) a recommendation system based on the EDEN approach may be constructed to devise which user-story or set of requirements is more "convenient";
4. Supporting **continuous architecting** [16], i.e., step 5 on Fig. 1, by steering further re-factoring decisions giving a precedence to features which are most beneficial (e.g., less time-consuming, cheaper, etc.) according to EDEN's (semi-)automatic recommendations.

Consequently, EDEN is an approach to re-use previously existing modelling notations to drive a software lifecycle in a DevOps fashion by continuous, evidence-based analysis.

The first benefit offered by EDEN for DevOps continuous analysis, is as (a) process status monitoring and (b) recommender system.

As process status monitoring, EDEN illustrates the status of tasks and functions (taking into account the scheduling of task processing decided by project manager) together with an estimation of the expected time required to finish the process and percentage of process already completed. This feature uses both the initial cost and effort predictions and live information: using initial predictions it is created the baseline of the process status, time-to-finish estimations and recommendations, which are further refined during the DevOps process with status information.

As a recommender system, using the remaining set of system functions and the connected development / operations tasks, their function-points "cost" and information of the DevOps team (e.g., skills, average tasks per Scrum cycle, etc.). EDEN uses this information to propose a scheduling of tasks over time and assignment of tasks to developers. This can also be automated as part of tools such as Mylyn[3] or TaskTop[4]. If assignment of tasks to developers is important EDEN may also allow to define different maturity levels among the members of the DevOps team, or different geographical location to account for timing delays.

3. EDEN IN ACTION: A SCENARIO

Our sample scenario stems from a micro-service [13] within the Netflix video streaming web system, more particularly a micro-service presenting recommendations to end-users. Fig. 3 depicts the scenario in its simplest form and phrased using an FP approach [10]. FP essentially envisions representing a very-high overview of the architecture producing its representation following the types of high-level functions depicted in Fig. 2. Essentially, a software system is generally comprised of: Internal Logical File (ILF) - homogeneous set of data used and managed by the application; External Interface File (EIF) - homogeneous set of data used by the application but generated and maintained by other applications; External Input (EI) - Elementary operation to elaborate data coming form the external environment; External Output (EO) - Elementary operation that generates data for the external environment, it usually includes the elaboration of data from logic files; External Inquiry (EI) Elementary operation that involves input and output, without significant elaboration of data from logic files. Based on their internal logic, the instances of each may have different complexity (see Fig. 2).

The development team has identified two functions to implement in the micro-service to deliver: a) a function to retrieve data from the *recommendation database*, concretised as an "Internal Logical File" (ILF) [10] that refers to the recommendation database query and b) the piece of software that collects user preferences by monitoring its interaction with the platform called *preference collector* concretised as a function of type "External Input" (EI) [10]. During the initial estimation (*cost Estimation* in Fig. 1), the development of EI got assigned a quantity fp_{ei} of function-points and the development of the ILF got fp_{ilf} function-points: their sum $fp_{ei} + fp_{ilf}$ gives the total tasks to develop. Using the original FP-counting algorithm and the standard FP complexity Table. Note that, whatever the count for function-points, the FP-counting algorithm was designed to take into account development tasks only. Therefore, to comply with a DevOps scenario, the FP-count needs to be refined to take operations into account.

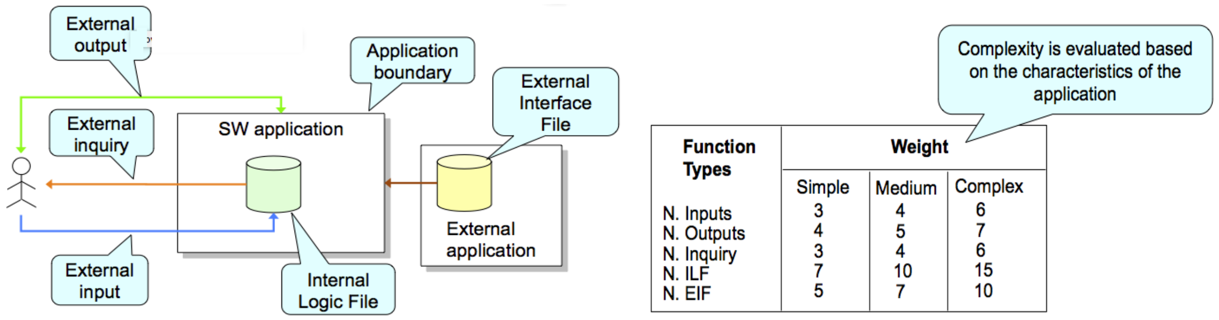
To produce accurate results of process status and valuable recommendations, let's assume Netflix developers want to use EDEN to drive a refactoring exercise. Said developers insert the following information in EDEN models: (1) The set of tasks initially foreseen by *Cost Estimation* in Fig. 1 and the previously stated relationships between these tasks; (2) Development and operations activities following the DevOps approach coded in the EDEN model; (3) The number of team members currently available (i.e., the available development/operations resources). For example, heterogeneous teams can assign quantitative value for their skills and expertise level (e.g., from very experienced to new team members); (4) The acquired-by-experience proportion of tasks that are newly generated during *development* (feedback arrow 4A in Fig. 1); e.g., tasks that failed to be identified in the initial *cost estimation* stage or are still under *development*; (5) The acquired-by-experience proportion of tasks whose necessity is identified during the *software operation* work (feedback arrow 5 in Fig. 1), e.g., due to refactoring/refinement needs; (6) The deadline agreed by the project manager for micro-service delivery, e.g., a *Scrum* cycle [15], a single day, ASAP, etc.

3.1 EDEN Models

EDEN envisions the creation of internal models that represent available software process information. These models are then analysed, e.g., to estimate process performance properties (e.g., expected time to finish or actual progress) or provide task-arrangement recommendations/priorities.

The lifecycle of a task starts when the task is identified (i.e., inserted in EDEN or mined from Software Lifecycle Management repositories such as JIRA[5] or BugZilla[6]). When a task is created, it may be either in a *scheduled* status, if a developer has been assigned to its completion, or in a *unscheduled* status, if it has not any developer associated with its completion. A developer can work both on tasks that have been assigned to him/her or on tasks that are unscheduled. From the moment when a developer starts working on a task until the task is delivered, the developer passes through activities *software development* and *software*

[3]http://www.eclipse.org/mylyn/
[4]https://www.tasktop.com/
[5]https://www.atlassian.com/software/jira
[6]https://www.bugzilla.org/



Function Types	Weight		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. EIF	5	7	10

Figure 2: Function-Points Algorithm: Function-Point Types (right-hand side) and Complexity Table (left-hand side).

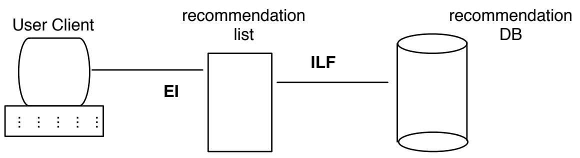


Figure 3: NetFlix: a sample scenario from a toy case-study.

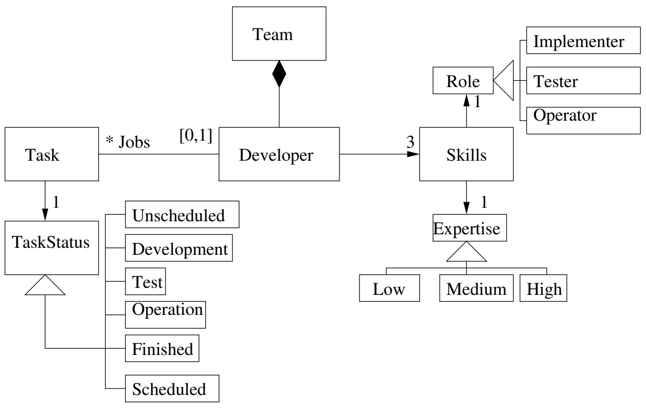


Figure 4: Class diagram of the software process

operation in Fig. 1. EDEN Internal models further refine *software development* activity into *implementation* and *testing* activities. After *testing*, the task goes either to *software operation* activity - in case of successful testing - or again to *implementation* - in case tests fail -. This flow of developer activities entails that each task passes through states *development*, *test*, *operation*, and after the operation state, the task enters a permanent *finished* state.

EDEN's lifecycle view of tasks is represented in the UML class diagram in Figure 4 while lifecycle evolution is represented in the sequence diagram extended with MARTE [11] annotations in Figure 5. Figure 4 also models the EDEN capability to take in to account developers skills and (in the future) elaborate further on their organisational structure [17]. Each developer has a value for his/her skills when developing, testing and operating. In turn, Figure 5 uses this skills information in its MARTE annotations to represent the expected time of a developer to carry out with develop, test and operation activities. These annotations use <<`ResourceUsage`>> stereotype, being the developer the resource. The internal `loop` in Figure 5 represents the iteration between developing and testing activities until the implementation successfully passes the tests. These models accounts for elements 1), 2) and 3) in the previous list of managed information by EDEN (see Fig. 1). The two internal `alt` operations in Figure 5 account for elements 4) and 5) in the previous list (see Fig. 1) of managed information by modelling the possibility of creation of new tasks - with status unscheduled - both after development and operation.

These UML models are used as input to an automatic model transformation that maps them into analysable models and possibly carries out their analysis.

3.2 Model Transformation and Analytics

This section describes model transformations from the EDEN domain models previously described to analyzable models. This step is carried out transparently, e.g., using tools and techniques inherited from [3] for petri-nets and [6] for queuing networks, and internally by EDEN. Consequently, EDEN industrial practitioners need not worry about any additional knowledge on complex analysis techniques and models.

A wide family of such analyzable models (or discrete event simulator based on their principles) can be considered for the type of information and recommendations envisioned by EDEN. Currently, we are reasoning on adopting stochastic models since they represent the aleatory uncertainty in the quantity of time required by developers to finish a certain task using probability distributions as much as they may represent the aleatory nature of the DevOps scenario in general. In the following we present three of said possible analysis models.

Phase-type distributions: Under the previously presented assumptions with respect to the software process made by EDEN, the expected timing of the evolution of a task once developer starts woking on it is well-represented by phase-type distributions [12] of four states: implementation i, test t and operation o as transient states, and an absorbing state *abs*. The operation state is the only one with a transition to the absorbing state and the initial probability vector $\boldsymbol{\alpha}$ is $\alpha_i = 1$ and $\forall s \in \{t, o\} \alpha_s = 0$. Using the expected times of a developer to complete implementation t_i, testing t_t and operation t_o (which are the *execTime* values in MARTE << `ResourceUsage`>> annotations in Figure 5), and the probability of a developer to develop a task which does not pass the tests *pFailTest* (whose value is in the

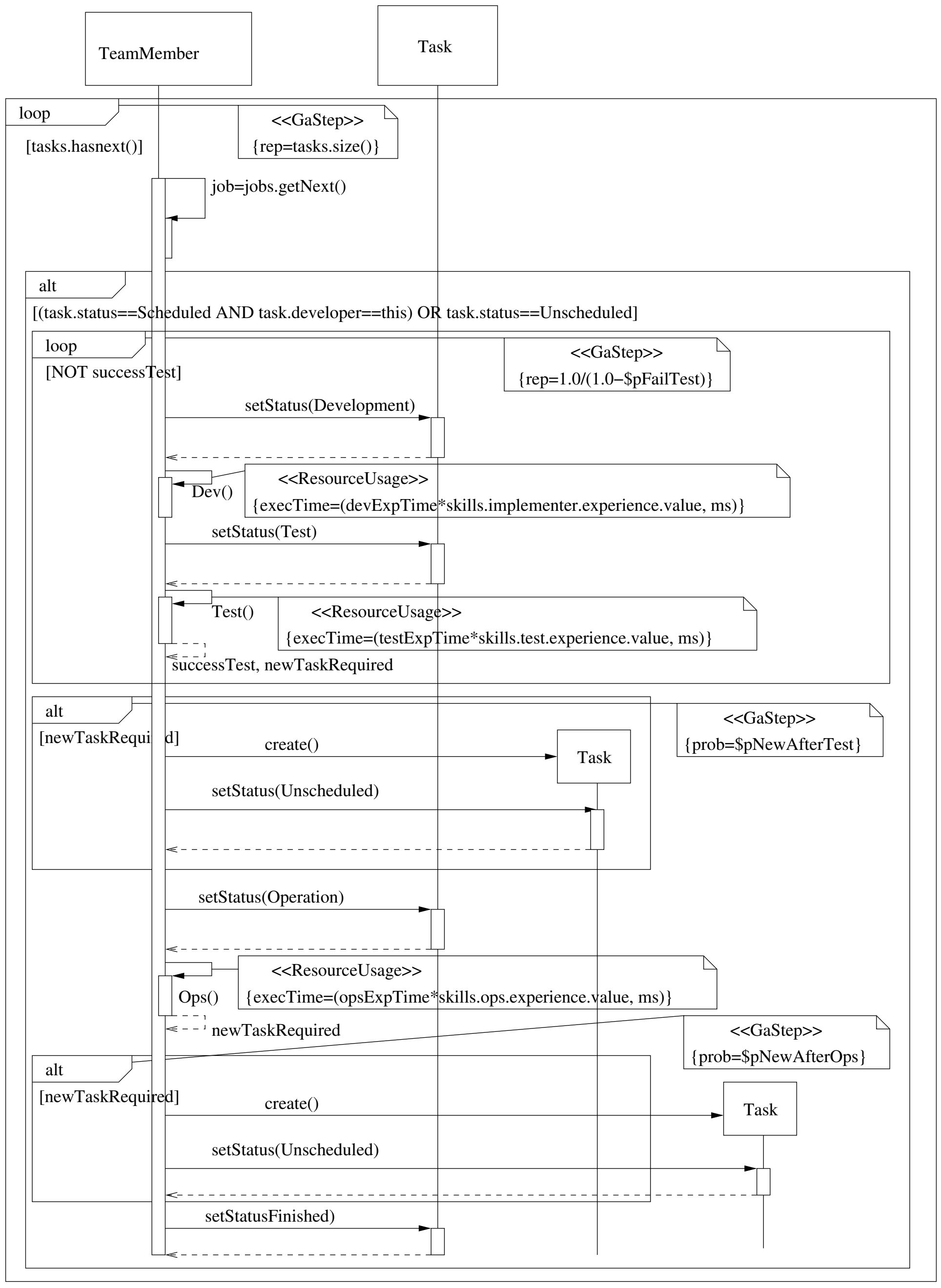


Figure 5: Sequence Diagram TeamMember centric activities

MARTE annotation attached to the `loop` in Figure 5), the values $q_{s,s'}$ of the infinitesimal generator matrix $\boldsymbol{Q}$ of the underlying continuous time Markov chain of the phase phase-type distribution are: $q_{i,t} = t_i^{-1}$, $q_{i,i} = -t_i^{-1}$, $q_{t,t} = -t_t^{-1}$, $q_{t,i} = t_t^{-1} pFailTest$, $q_{t,o} = t_t^{-1}(1 - pFailTest)$, $q_{o,o} = -t_o^{-1}$ and $q_{o,abs} = t_o^{-1}$. The rest of $q_{s,s'}$ are equal to 0. Fig. 6(b) depicts the underlying continuous time Markov chain of this distribution. This model takes into account elements 2) and 3) in the previous list of managed information and enables analyzability. Element 3) is taken into account by assigning a $\boldsymbol{Q}$ to each developer including their personal capabilities.

A more accurate analysis of the timing and evolution of the software process should also consider the tasks that are generated during development, i.e., those ones that were not anticipated in the initial estimation. According to Figures 1 and 5, the realization of the necessity of new tasks can happen both during the *development* and during the *operation* of other tasks. The introduction of these concepts in the analyzable models enables the consideration in the analysis of elements 1), 4) and 5) in the list of managed information. As introduced in the list, we assume that the DevOps team has followed some internal evaluation operations, e.g., during SCRUM cycles, and they have some metrics like proportion of foreseen and unforeseen tasks at the Cost Estimation stage. Using these proportions, EDEN models may feature additional parameters: the probability of unveiling the necessity of a new task during the integration tests in the Development activity (called *pNewAfterTest* in annotations in Figure 5), and the probability of discovering a new task the Operation activity of another task(called *pNewAfterOps* in annotations in Figure 5). For analyzing these possible behaviors, we show the output of the model transformation to the stochastic models of Extended Queueing Networks (EQN) [4] and Generalized and Stochastic Petri Net (GSPN) [1] (see Fig. 6(a) and 6(c)). For the sake of simplicity in the illustration of the following models, we consider a development process where all tasks are initially in a unscheduled state and all developers have the same skills.

EQN: EQN in Fig. 6(a) uses, beyond the classical service nodes, allocation elements to acquire and release the resources (i.e., the developer in the current case), and split nodes to create new tasks according to a given probability after both the development and operation stages. All service nodes are infinite server. Service rates and probabilistic routing decisions have been assigned with the appropriate values to model the evolution of the status of a task with the same accuracy as the Phase-type distribution in the part (a) of the Figure. As it is an automatic model transformation from the domain models, all required values for the model transformation are also present in Figure 5. **GSPN:** In turn, part (c) depicts a GSPN with the same behavior as (a). Again, routing decisions are decided probabilistically by the probability associated to each of the transitions in conflict in each marking and new tasks can be created after the development stage and after the operations stage by the immediate transitions with the associated probabilities $(1 - pFailTest)pNewAfterTest$ and $pNewAfterOps$, respectively.

During the development, these models can be updated, and jobs or tokens in the EQN or GSPN, respectively, can be set to the actual stage in which they are. In such way, continuous analysis of the models along the software process will provide results better aligned to the future reality.

4. RESEARCH ROADMAP

This section discusses the main EDEN benefits and outlines a tentative research roadmap that covers most of the key research venues we found during this initial investigation of our idea.

First, we observed that EDEN offers a form of evidence-based insight curiously reminiscent of Joel Spolsky' Evidence-Based Scheduling[7]. In so doing, EDEN offers insights that fit equally well either in projects, teams and organisations that already underwent a shift to agile methods or in their non-agile counterparts. In the former scenario, EDEN would help planning cycles and iterations or during retrospectives while in the latter scenario EDEN 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. Additional research needs to be invested on EDEN first to understand the implications of this observation and second to understand if and how EDEN (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, EDEN may work well in 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). EDEN 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. From the sketch we provided in this paper, we could grasp that EDEN: (a) offers the possibility to combine multiple well-established notations at the same time and for the key purpose of steering and governing software lifecycles in a DevOps fashion; (b) said multiple notations have non-trivial but intriguing interactions and may yield valuable insights in combination (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) EDEN may have the potential to lower considerably the organisational and technical barriers [17] in DevOps. From these premises there are several research directions for future work: First, More research shall be invested in **EDEN automation**. We observed that much of the information that EDEN tools and methods need, 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 repos such as MyLyn. EDEN may be rigged to work in a proactive way by retrieving this necessary data automatically. Moreover, several Application Lifecycle Management (ALM) solutions exist that may work well from interfacing with EDEN. For example, tools such as TaskTop[8] that offer harmonisation layers across software development pipelines may feed task-based information into EDEN and receive feedback in return.

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

[7]http://www.joelonsoftware.com/items/2007/10/26.html
[8]http://tinyurl.com/j7mxxtg

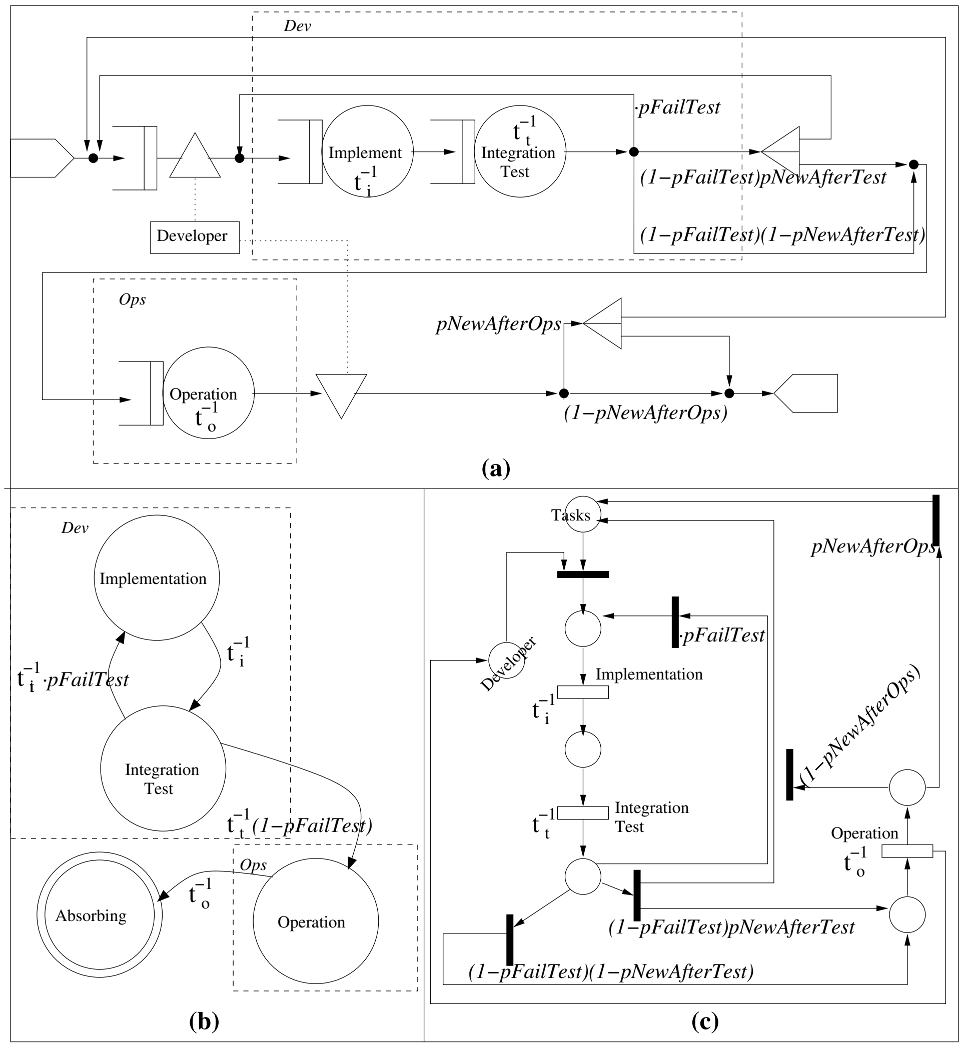


Figure 6: Analyzable models modelling the DevOps lifecycle

For example, Petri nets and queuing networks models with the emphasys 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;

Third, more research shall be invested in **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. Specifically, we can distinguish two main research directions: (1) **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;(2) **model analysis:** more research on EDEN 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;

5. CONCLUSION

DevOps is recently gaining momentum as a corporate software engineering culture and strategy that promises increased efficiency and reduce time to market. Analysing industrial practice, we observed several key limitations to DevOps adoption due to lack of reuse of existing architecture-centric practices in continuum with DevOps tenets and challenges. In response to this, this paper illustrates EDEN, a first-of-its-kind DevOps engineering approach featuring the (re-)use of available modelling notations and technologies which are currently very common in industrial practice (e.g., Function-Points Modelling and Analysis or Petri-Nets). EDEN is a mechanism to aid the industrial adoption of DevOps approaches by allowing intelligent steering of new software projects using a strategy closer to DevOps. Even in its primordial stages, EDEN shows great promise and several key interesting research paths. For example, EDEN may function as a DevOps governance mechanism rotating around previously existing modelling notations. More research should be invested in further elaborating on how and by means of which combination of modelling notations and tools may best support DevOps[9].

[9]Some of the authors' work is partially supported by the

6. REFERENCES

[1] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. *Modelling with Generalized Stochastic Petri Nets.* John Wiley Series in Parallel Computing - Chichester, 1995.

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

[3] Bernardi, S., Donatelli, S., and Merseguer, J. From uml sequence diagrams and statecharts to analysable petri net models. In *WOSP '02: Proceedings of the 3rd international workshop on Software and performance* (New York, NY, USA, 2002), ACM Press, pp. 35–45.

[4] C. Sauer, E. M., and Kurose, J. The research queueing package: Past, present and future. In *National Computer Conf* (1982), pp. 273–280.

[5] Conway, M. How do committees invent? *Datamation Journal* (April 1968), 28–31.

[6] Cortellessa, V., and Mirandola, R. Prima-uml: a performance validation incremental methodology on early uml diagrams. *Sci. Comput. Program. 44*, 1 (2002), 101–129.

[7] Guerriero, M., Tajfar, S., Tamburri, D. A., and Nitto, E. D. "towards a model-driven design tool for big data architectures".

[8] Jensen, K. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Springer-Verlag, 1997. 2nd corrected printing.

[9] Marcello M. Bersani, Francesco Marconi, D. A. T. P. J., and Nodari, A. "continuous architecting of stream-based systems". 115–121.

[10] Matson, J. E., Barrett, B. E., and Mellichamp, J. M. Software development cost estimation using function points. *IEEE Trans. Software Eng. 20*, 4 (1994), 275–287.

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

[12] Neuts, M. *Matrix-geometric Solutions in Stochastic Models: An Algorithmic Approach.* Algorithmic Approach. Dover Publications, 1981.

[13] Newman, S. *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, February 2015.

[14] Polk, R. Agile and kanban in coordination. In *AGILE* (2011), IEEE Computer Society, pp. 263–268.

[15] Schwaber, K., and Beedle, M. *Agile Software Development with Scrum.* Prentice Hall, 2001.

[16] Shahin, M. Architecting for devops and continuous deployment. In *ASWEC (2)* (2015), F.-C. D. Kuo, S. Marshall, H. Shen, M. Stumptner, and M. A. Babar, Eds., ACM, pp. 147–148.

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

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

[19] Woods, E. Harnessing uml for architectural description-the context view. *IEEE Software 31*, 6 (2014), 30–33.

European Commission grant no. 610531 (FP7 ICT Call 10), SeaClouds and by the European Commission grant no. 644869 (H2020 - Call 1), DICE.