On the Importance of Simulation in Enabling Continuous Delivery and Evaluating Deployment Pipeline Performance

Andrea D'Ambrogio
Dept. of Enterprise Engineering
University of Rome
Tor Vergata
Via del Politecnico 1
00133, Rome, Italy
dambro@uniroma2.it

Alberto Falcone, Alfredo Garro
Dept. of Informatics, Modeling,
Electronics and Systems Engineering
University of Calabria
Via P. Bucci 41C
87036, Rende, Italy
{alberto.falcone, alfredo.garro}@dimes.unical.it

Andrea Giglio
Dept. of Innovation and
Information Engineering
Guglielmo Marconi University
Via Plinio 44
00193, Rome, Italy
a.giglio@unimarconi.it

***Abstract*—In modern Software Engineering, Continuous Delivery (CD) is a development approach in which a software is iteratively developed in short cycles ensuring, for each cycle, that the new features are available to end users as soon as they are implemented and tested. CD aims at defining, building and releasing software with greater speed and frequency through the *deployment pipeline* resulting in three major benefits, *visibility, feedback and continuous deployment* respectively, enabling the software functional items to efficiently flow from development to production. In this domain, the need for evaluating the performance of the *deployment pipeline* emerges, since the conventional metrics available in the software engineering discipline are not suited to handle all the involved aspects. In this paper, the metrics suited for supporting CD are introduced and an integration with Modeling and Simulation (M&S) techniques is discussed, based on the Business Process Model and Notation (BPMN) standard, which could represent a valid support offering a graphical notation to easily specify the *deployment pipeline* steps as a standard and repeatable process. The main objective is to identify feasible perspectives in which simulation methods and principles can be exploited, thus evaluating the effectiveness of M&S to support performance analysis of a *deployment pipeline*, seen as a predictable process. Specifically, M&S can be seen as an enabling tool for the evaluation and comparison of different CD choices against requirements through an effective implementation of simulation techniques and virtual testing.**

***Index Terms*—Continuous Delivery, Business Process Model and Notation (BPMN), Modeling and Simulation**

I. INTRODUCTION

Nowadays, in the software engineering and software development domains, many influencing factors play a fundamental role in making software production activities increasingly challenging. Fast-changing customer requirements, as well as unpredictability of market and the need of an ever shorter time to-market are non-negligible aspects that have to be taken into account, since software development has become a demanding area of business [8]. First approaches, strongly based on attractive *lean* principles proposed the adoption of strategies such as *"Decide as late as possible"* [30], configured themselves as early steps towards a *Continuous Delivery (CD)* approach. Furthermore, the dramatic impact of the Agile Manifesto on the way software production is managed, has led in recent years to the widespread adoption of new emerging approaches in software engineering. The first of the twelve fundamental principles defined in the aforementioned manifesto states that *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"* [20].

Despite the fact that continuous delivery was explicitly mentioned in this principle, its widespread adoption is the result of an evolution of different approaches through the years. Starting from *continuous integration (CI)* shifting towards modern *continuous deployment pipelines*, releasing new software versions early and often has become mainstream, and now represents a concrete option also for an ever growing number of companies and practitioners. Let us summarize some baseline definitions:

- *Continuous Integration*. CI is the process of ensuring that a software build is in a correct working state, guaranteeing developers that the release is suitable for production. In this approach, members of a team are pushed to integrate their work frequently, typically each person integrates at least daily leading to several integrations per day. It is worth noting that each integration is verified by an automated build and test procedure with the aim of detecting integration errors as soon as possible.
- *Continuous Delivery*. Deriving from CI, CD ensures that a build is always in a releasable state. Going hand to hand with a *DevOps* approach, where the team is responsible for all aspects of software delivery, after integration and testing activities the software is actually provisioned to a full, production-like stack, and delivered to some set of end users.
- *Continuous Deployment.* When the entire process, from check-in to production, is fully automated, with no human intervention, *continuous deployment* is implemented.
- *Deployment Pipeline.* The foundation of continuous delivery is the *deployment pipeline*, which can be seen the path

Copyright © held by the author

CEUR-WS.org/Vol-2248/paper7.pdf

that a code change takes from check-in (i.e. the commit operation) to production.

CD is a software engineering approach that represents a huge step forward in productivity and quality for companies and organizations that adopt it. CD is gaining attention since it provides a well-established process that allows team members to work together to define and create large and complex software with a higher level of control. CI claims to enable companies to release new features, configuration properties, bug fixes and tests, to customers safely and quickly in a sustainable way. This means that it mainly focuses on asserting the correct compilation of the source code and that it passes a chain of test units.

However, measuring the performance of a CD *deployment pipeline* is a challenging task that requires a considerable effort in terms of both time and cost. Many companies including Google, Facebook and IBM are developing their software products by using CD [8] and are trying to find out viable solutions to measure the CD performance. To this purpose, significant benefits can derive from the possibility to perform Modeling and Simulation (M&S) activities on the CD *deployment pipeline* in order to understand, measure and optimize the behavior of systems on which to perform experiments and theoretical analyses.

In this context, the paper introduces metrics that are suited to evaluate the performance of a CD *deployment pipeline* through M&S techniques based on the Business Process Model and Notation (BPMN) standard, in the purpose of defining the pipeline as a standard process and also of analyzing it as a repeatable and predictable process.

The ultimate aim is to provide an exploratory analysis of the available approaches to minimize the *cycle-time* in software production (i.e. the period it takes from an idea to provide actual business value), and also to investigate the different domains in which simulation techniques can be fruitfully exploited to enforce continuous delivery activities through a deployment pipeline optimization.

The rest of this paper is organized as follows. Section II provides an introduction to the essential concepts and background knowledge on the research domain. The *Continuous Delivery (CD)* methodology and the typical architecture of a *CD deployment pipeline* are presented along with some related works available in literature. In Section III, the *CD deployment pipeline* is defined through the concepts provided by the *Business Process Model and Notation (BPMN)* standard with particular focus on how to make available Modeling and Simulation (M&S) practices during the pipeline flow in order to evaluate, predict and optimize its behavior. Finally, Section IV discusses the main objectives of the work and presents some future works.

II. Background and Related work

Continuous Delivery (CD) is a concept which has been taken as a pillar of modern Agile software engineering [11] because its enables higher software development velocity and productivity. The concept of continuous delivery means simplifying and automating the whole delivery process for an application after the source code has been committed into a repository. CD is centered around the collaboration aspects of teams involved in the software production such as, developers, project managers and users, as well as automating the software delivery steps so as to ensure that a software is always ready to be deploy and then delivered to end-users.

Basically, the *deployment pipeline* is an automated process for getting software from the repository into the hands of end-users. CD is preceded by the Continuous Integration (CI) in which the team members integrates the source code [31], [34]. This step leads to a faster feedback cycle and to benefits such as improved productivity and increased communication [31]. Figure 1 shows the *deployment pipeline*.

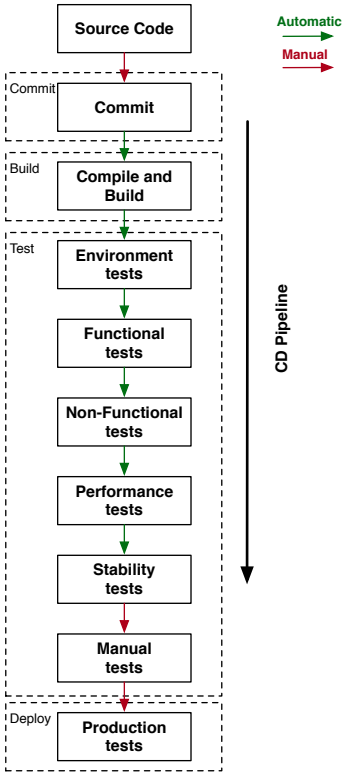

Fig. 1. Overview of the CD deployment pipeline.

The CD *deployment pipeline* starts when developers commit the source code changes into the version control system. At this point, the CI Management System (CI-MS) triggers a new instance of the deployment pipeline and compiles the source code. If the compile procedure successfully completed a transition to the *test* phase is performed; otherwise the procedure terminates with an error. In the *test* phase, the CI-MS runs a set of unit tests, performs code analysis, and builds the installer package. In this step, some manual tests needs to be performed in order to evaluate domain-dependent functional and non-functional requirements. If all the tests pass, the executable code is used to generate the binaries that will be stored in an *artifact* repository [21].

The related works available in the literature are vast and

extend through empirical and modeling and simulation approaches to studying processes in software development.

In [33] Van Der Hoek et al. identified the problems of releasing component-based software and presented a flexible software release management tool that acts as a bridge to reduces the gap between the development process and deployment process.

Lahtela et al. in [24] presented nine significant challenges regarding the delivery process of software. The authors, also presented solutions to overcome these challenges through well-defined guidelines on how to build an ITIL-based release management process and how it should be performed.

Kajko-Mattson and Fan in [22] outlined a model of the release management process integrating both the vendor and acceptor sides.

Krishnan in [23] presented an economic model to capture and analyze a set of tredoffs involved in the software release decisions and discussed a method to optimize the delivery process that takes into account crucial factors such as the software development cycle and the changes in market needs.

The above presented works do not cover all the aspects engaged in the continuous delivery process and do not provide an adequate view on the issues that can emerge in performing the release management for IT services [3]. Moreover, they are based on empirical results and not on simulation ones that can help to better understand, predict and optimize the delivery process of software.

Through the following related works, an insight into the main issues related to the use of modeling and simulation techniques in delivering software, is given.

Dlugi et al. in [10] highlighted the difficulties to evaluate the performance of a software release, mainly related to the lack of a test environment that is comparable to a production system. The authors introduced a model-based *performance change detection* process that uses modeling and simulation methods for evaluating performance metrics of different software versions. Also, they developed a a plug-in for a CD *deployment pipeline* based on Jenkins.

In [35], Vöst and Wagner investigated the CD pipeline in the automotive domain. They evaluated the effect of various functional and non-functional requirements on the development life cycle. They presented a typical *delivery pipeline in automotive software* that adopts Hardware-in-the-Loop-Simulation approaches to evaluate the software with the operating environment.

Zia et al. in [1] used the Business Process Model and Notation (BPMN) standard to evaluate DevOps alternatives. The effectiveness of BPMN has been demonstrated in other domain [15], [16], [17], and this can represent a viable solution to model and simulate CD pipelines.

This paper has various purposes in common with the presented works but unlike them it introduces metrics to evaluate the performance of a CD *deployment pipeline* through M&S techniques based on BPMN so as to define a standard CD pipeline on which to perform analysis.

III. The Pipeline as a Standard and Predictable Process

A. The Pipeline

Under the assumption that both a business process collaboration and a CD *deployment pipeline* implement an exchange of information between logical processes, in this paper the adoption of BPMN as a notation to define a *deployment pipeline* is proposed. This allows to perform simulation test units in order to evaluate the performance of a *pipeline* through a set of well-known metrics such as, those presented in [25].

The *deployment pipeline* is the workflow, with related activities, that a source code follows from commit to production. Each commit, made by the development team, generates a *release candidate* of the software which flows through the pipeline. If everything goes well, which means that all steps of the pipeline are correctly executed, the software is ready for release. Figure 2 shows a general CD *deployment pipeline* formalized in BPMN notation [17].

With reference to the *BPMN/CD deployment pipeline* depicted in Figure 2, the pipeline starts in the *Development* swing-lane where the source code is ready to be committed into a repository. A transition to the *Continuous Integration* swing-lane happens and during the activity transition a connection to the repository is performed. At this point, the *commit* activity is performed in order to save the changes to both the local and remote repository. In the *Build* activity, the source code is built and the resulting executable file is generated. After that, a transition to the *Simulation* activity is done. Throughout this stage, the new software version is rigorously tested through simulation techniques to evaluate, predict and optimize its behavior. It is important that all the requirements aspects whether functional, non-functional, performance, security and compliance are verified. For each simulation test, some software metrics related to various constructs like class, cohesion and inheritance have been evaluated. To evaluate the complexity of the source code, three standard metrics, which are proposed by various researchers, can considered [36]: (i) *CCM (Cyclomatic Complexity Metric)*, which is a metric based on graph theory that represents the number of linearly independent paths through a program's code; (ii) *HCM (Halstead Complexity Metric)*, which measures the logic volume of the code. It is calculated on the count of the operators and operands; (iii) *(v) NF (number of function)*, which represents the total number of functions. Typically, the lower are the values of these metrics the lower is the complexity of the source code and thus higher should be the code compactness, readability and reliability.

In the last activity, named *Test units*, a set of parametrized test units are executed. This allows to measure the progress of the software and detect side effects. A transition to the *Production* swing-lane happens if all the activities have been successfully completed. Otherwise, if at least one activity failed, a state transition to the *source code* activity is performed, since the source code needs to be revised, and the pipelines terminates.

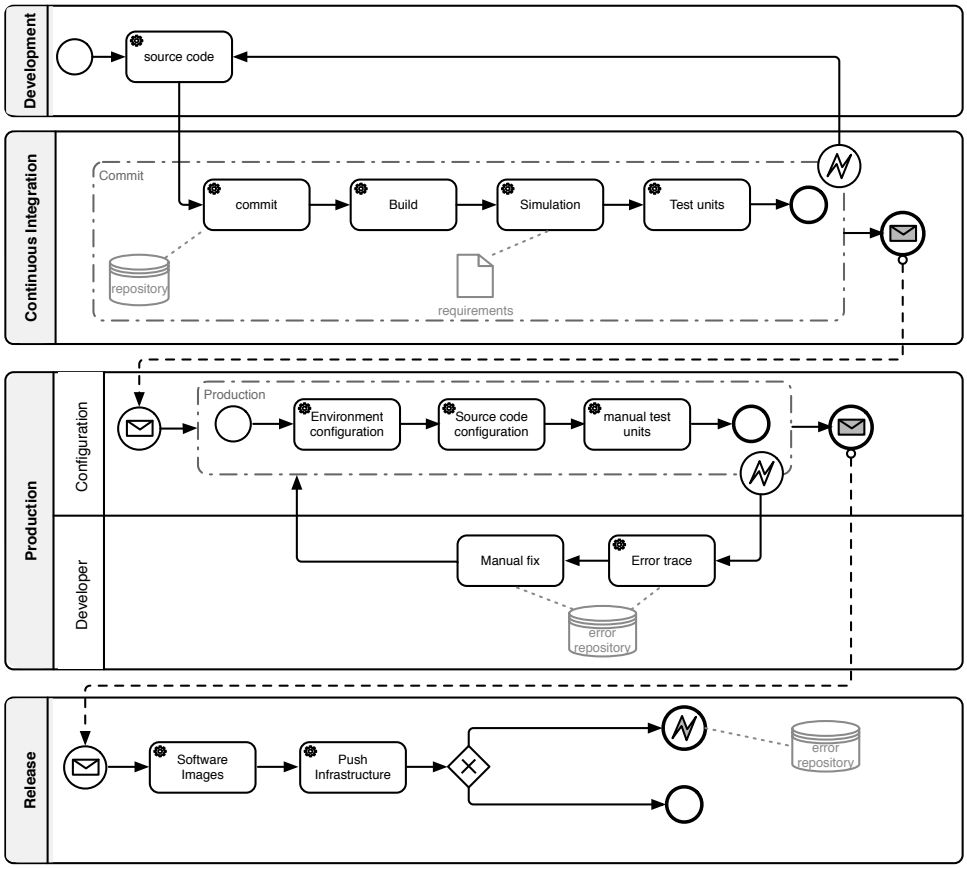

Fig. 2. Overview of the CD deployment pipeline defined in BPMN.

The *Production* swing-lane, the parameters related to the target environment (e.g. AWS Amazon Cloud, Microsoft Azure, Google Cloud, etc.) are configured in the *Environment configuration* activity. Thereafter, the *Source code configuration* activity is performed, in which the software is configured to run into the target environment. Finally, additional manual test units, which take into account the production environment, are executed in the *manual test units* activity.

A transition to the *Release* swing-lane happens if everything goes well. Otherwise, the issues occurred during the *production* activities are collected and stored into the *error repository* in order to be evaluated and fixed by developers. In the *Release* swing-lane, the image of the software release is created and deployed in the environment.

B. Relevant Metrics

Generally speaking, in the study of business processes performance, metrics of particular interest fall under two main categories.

On the one hand we find those metrics that are directly related to time, for example the time in which a user (or job) crosses the entire process starting from the initial state arriving at the final state, executing different tasks in its path. In this case, this metric is defined as *cycle time*.

Since cycle time includes both value-adding and non value-adding activity times (e.g., waiting times), it is a powerful tool to identify process improvement potential.

On the other hand, we find those metrics that refer to the amount of work done in a certain time, for example the number of users (or jobs) that complete the whole process during an observation time. In this case, the abovementioned metrics are related to the concept of *throughput*.

In the context of continuous delivery and deployment, the concepts of *"user"* or *"job"* can be effectively intended as *"features"* flowing throughout the pipeline, from the initial commit to the final deployment in production.

Since from a business analyst perspective, the cycle time is the period it takes from an idea to provide actual business value, it is easy to understand that most profits could be gained from that idea if it can be implemented quickly. If the cycle time is too high, competitors might be first or the idea might not be relevant anymore.

Under this perspective, it becomes clear how, to that optimizing the cycle time requires an efficient process which usually spans multiple departments and involves a lot of people (i.e., a complex process).

In order to optimize a complex process, we need an efficient way to specify and analyze each task that is involved in it. As aforementioned we propose BPMN to describe the deployment pipeline process, and propose the use of simulation techniques to enact analysis activities.

A similar approach enabling performance predictions over a business process has been introduced and discussed in [5], [9]

Benefits of analyzing the process of such a pipeline include finding and removing bottlenecks (e.g., a specific activity that is too expensive in terms of time, due to some inefficiency), shortening the feedback loop (i.e., measuring activities in the middle of the pipeline can be useful to provide an early feedback to end users), automating as much as possible (i.e., an accurate analysis can lead to the identification and the reduction of expensive manual activities and eliminate any error-prone manual tasks). In other words, the ultimate purpose is to optimize and visualize what's going on to improve the flow.

In the continuous delivery and deployment domain, it is easy to understand how the throughput of the pipeline used to deliver new features to production (i.e., to the end user) is a relevant metric.

Referring to the lean world and to kanban practices, a metric that completely captures the concept of the pipeline is certainly *flow efficiency*, which is calculated based on the actual work time (i.e., the actual time spent working on a feature) measured against the total wait time (e.g., data transfer, hardware or software provisioning, environment setting, etc.), presented in [26].

The calculation of flow efficiency is as follows:

$$Flow\ Efficiency\ in\ \% = \frac{Work\ Time}{Work\ Time\ +\ Wait\ Time}$$

Where:

- *Work Time* is the time in which the development of the feature under study can be seen as *in progress*.
- *Wait Time* is the time in which the development of the feature under study can be seen as *blocked* or *waiting*.

Other interesting metrics have been introduced, which can be fruitfully taken into account for analyzing a deployment pipeline, as *Features Per Month (FPM)*, *Releases Per Month (RPM)*, or *Fastest Possible Feature Lead Time* [25].

TABLE I
SUMMARY OF RELEVANT METRICS.

Metric	Unit of Measure
Cycle Time	Time
Throughput	Frequency
Flow Efficiency	%
Features Per Month (FPM)	Features / Time
Releases Per Month (RPM)	Releases / Time
Fastest Possible Feature Lead Time	Time
MTBF	Time
MTTR	Time

- *Features Per Month (FPM)*, identifies the number of new features that have crossed the entire pipeline during a month. The metric is based on *Day-by-the-Hour (DbtH)*, which measures quantity produced over hours worked [29].
- *Releases Per Month (RPM)*, measures the number of releases that have been completed during a single month. It is worth noting how changes in the long term of this metric suggest information on changes in the release cycle. Data for obtaining such a measure can be retrieved from the version control system or from the integration server logs, for instance.
- *Fastest Possible Feature Lead Time*, measures a sort of best case scenario in which, without delays and latencies (e.g., caused by the use of feature branches or by separate build processes), the feature under study spends time only in the build and test phase on the pipeline.

Also monitoring *Mean Time Between Failures (MTBF)* and *Mean Time To Repair (MTTR)* and their balancing can be an effective way to optimize the deployment pipeline.

Mean time between failures reminds the team to to avoid easy failures. However, core point of software development is to provide new value to users, and only looking at MTBF can result in teams becoming overly cautious and never releasing anything new.

In order to avoid such a drawback, a focus on mean time to recover (i.e., a metric that measures the ability to rollback in case of a mistake) can be a key counterbalance.

- *Mean Time Between Failures (MTTF)*, is the predicted elapsed time between inherent failures of a system, during normal system operation. In the deployment pipeline can be a measure of the mean time between a feature not passing some test, for example. MTBF can be calculated as the arithmetic mean (average) time between failures of a system (i.e., a feature scoring a KO in some test). Achieving a good MTBF in a pipeline involves getting feedback early on and making sure that thorough validation occurs in testing environments. Moreover, such validations should be run on environments that are similar to production, and with realistic data.
- *Mean Time To Repair (MTTR)*, represents the average time required to repair a failed component or device (i.e., the failure of a feature in the pipeline). Assuming failure is inevitable, it's important to ensure that mean time to recover is as fast as possible. In other words, it measures the time that is needed to *"rollback"* to a previous build after a release failure. In this scenario it is clear how robust monitoring of production is essential. In order to make this approach more effective, teams should learn about failures through monitoring and alerts, not through customer complaints.

Table I summarizes relevant metrics of interest and their units of measure.

C. Simulation Benefits

The ever-growing complexity of modern software, which also involves physical and/or virtual platforms, requires the adoption of effective analysis techniques to support the design and operation. Using M&S techniques is it possible to generate the real-world scenarios that would be hard to get in the

real world, especially in distributed environments [13], [14], [18], [27], [12], [28]. Moreover, moving from monolith to Microservice architecture, the involved parts have their own lifecycle [32].

The solution introduced in section III (see Figure 2) regards the availability of a *Simulation* activity that combines the static test units with simulation ones. The benefit of the *Simulation* activity is that it continuously urges the software services, creating a constant and uniform load on them. This allows to monitor the *CD development pipeline* and notify developers if any activity failed.

Moreover, through the use of the *Simulation* activity is it possible to evaluate the source code, which is managed by a *CD development pipeline*, on specific embedded platforms and then overcome typical issues involved in the hardware-based setups, such as:

- *Hardware Checking*. Managing simulation tests is much easier respect to use hardware, especially when complete and actual tests are too expensive to be performed in terms of cost, time and other resources. Though simulation is also easier to handle multiple hardware configurations, since this means to change configuration parameters in the tests.
- *Test Software Reflecting the Hardware*. Generally, Hardware platforms have limited amounts of computational power and memory space. Through simulation is it possible to evaluate the maximum workload.
- *Configuration Reuse*. With a simulation is very easy to create, save and reuse hardware/software configurations.
- *Error Handling*. The part of the source code that is responsible for handling faults and errors conditions can be difficult to test on hardware. Through a simulator, inject errors/faults is easy since any part of the simulator code can be accessed and modified.
- *Environmental aspects*. By using simulation is it possible to imitate environmental conditions in order to evaluate how the system responds to environmental stresses. Such tests are very hard to perform using the real system.

IV. Conclusions

In the last decades software production activities have become increasingly challenging and software development has become a demanding area of business. In order to reduce the time to market of new product features, following the *Agile* and *DevOps* principles, new approaches as *continuous delivery* and *continuous integration* are now widely adopted, enabling the software functional items to efficiently flow from development to production.

Such an approach leverages on methods and tools as the *deployment pipeline*, which represents a series of tasks to be executed in order to deliver a software product or feature to end users, starting from a commit stage.

From this perspective, the need for analyzing and optimizing such activities emerges.

In this paper, we have discussed the adoption of a well-known standard to represent the deployment pipeline as a process (i.e., BPMN), in order to make it repeatable and clearly specified.

Moreover, we have discussed a set of candidate metrics which can be suitable to analyze the performance of such process, most of them referable to the concepts of *throughput* and *cycle-time*.

Finally, we have discussed the benefits of simulation approaches and techniques to carry on performance predictions over a deployment pipeline.

More specifically, the support provided from simulation can be twofold. On the one hand, end-to-end performance of the whole pipeline can be analyzed over different configurations, and, on the other hand, particular hardware or software components can be simulated (e.g., using contracts in case of microservices or using a virtual hardware component).

As a further step, work is ongoing to implement a full stack framework to enact modeling and simulation of a deployment pipeline, leveraging on previous experiences in a cloud-based environment [4], [6], [7], [2], [19].

References

[1] Zia Babar, Alexei Lapouchnian, and Eric Yu. Modeling devops deployment choices using process architecture design dimensions. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 322–337. Springer, 2015.

[2] P. Bocciarelli, A. D'Ambrogio, E. Paglia, T. Panetti, and A. Giglio. A cloud-based service-oriented architecture for business process modeling and simulation. In *CEUR Workshop Proceedings – INCOSE Italia Conference on Systems Engineering*. CEUR-WS, CEUR-WS, 2017.

[3] Paolo Bocciarelli, Andrea D'Ambrogio, Alberto Falcone, Alfredo Garro, and Andrea Giglio. A model-driven approach to enable the distributed simulation of complex systems. In *Complex Systems Design & Management, Proceedings of the Sixth International Conference on Complex Systems Design & Management, CSD&M 2015, Paris, France, November 23-25, 2015*, pages 171–183. Springer-Verlag, 2015.

[4] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio, and Antonio Mastromattei. Automated development of web-based modeling services for msaas platforms. In *International Workshop on Model-Driven Approaches for Simulation Engineering (Mod4Sim 2017), part of the 2017 Spring Simulation Multiconference (SpringSim 2017)*, Virginia Beach, VA, USA, 2017.

[5] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio, and Emiliano Paglia. A bpmn extension to enable the explicit modeling of complex task resources. In *Proceedings of the INCOSE Italy Conference on Systems Engineering*, 2016.

[6] Paolo Bocciarelli, Andrea D'Ambrogio, Antonio Mastromattei, Emiliano Paglia, and Andrea Giglio. Business process modeling and simulation: State of the art and msaas opportunities. In *Proceedings of the Summer Simulation Multi-Conference*, SummerSim '17, pages 1–12, San Diego, CA, USA, 2017. Society for Computer Simulation International.

[7] Paolo Bocciarelli, Andrea D'Ambrogio, Emiliano Paglia, and Andrea Giglio. A service-in-the-loop approach for business process simulation based on microservices. In *Proceedings of the 50th Computer Simulation Conference, SummerSim 2018, Bordeaux, France, July 09-12, 2018*, pages 24:1–24:12, 2018.

[8] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybke Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57(Complete):21–31, 2015.

[9] Andrea D'Ambrogio, Emiliano Paglia, Paolo Bocciarelli, and Andrea Giglio. Towards performance-oriented perfective evolution of bpmn models. In *6th International Workshop on Model-Driven Approaches for Simulation Engineering*, 2016.

[10] Markus Dlugi, Andreas Brunnert, and Helmut Krcmar. Model-based performance evaluations in continuous delivery pipelines. In *Proceedings of the 1st International Workshop on Quality-Aware DevOps, QUDOS 2015, Bergamo, Italy, September 1, 2015*, pages 25–26, 2015.

[11] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016.

[12] Alberto Falcone and Alfredo Garro. The SEE HLA starter kit: enabling the rapid prototyping of hla-based simulations for space exploration. In *Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems 2016, MSCIAAS 2016, and Space Simulation for Planetary Space Exploration, SPACE 2016, part of the 2016 Spring Simulation Multiconference, SpringSim 2016, Pasadena, CA, USA, April 3-6, 2016*, page 1. The Society for Modeling and Simulation International Inc., 2016.

[13] Alberto Falcone and Alfredo Garro. A java library for easing the distributed simulation of space systems. In *16th International Conference on Modeling and Applied Simulation, MAS 2017, Held at the International Multidisciplinary Modeling and Simulation Multiconference, I3M 2017, Barcelona, Spain, September 18-20, 2017*, pages 6–13. CAL-TEK S.r.l., 2017.

[14] Alberto Falcone, Alfredo Garro, Anastasia Anagnostou, and Simon J. E. Taylor. An introduction to developing federations with the high level architecture (HLA). In *2017 Winter Simulation Conference, WSC 2017, Las Vegas, NV, USA, December 3-6, 2017*, pages 617–631. Institute of Electrical and Electronics Engineers Inc., 2017.

[15] Alberto Falcone, Alfredo Garro, Andrea D'Ambrogio, and Andrea Giglio. Engineering systems by combining bpmn and hla-based distributed simulation. In *2017 IEEE International Conference on Systems Engineering Symposium, ISSE 2017, Vienna, Austria, October 11-13, 2017*, pages 1–6. Institute of Electrical and Electronics Engineers Inc., 2017.

[16] Alberto Falcone, Alfredo Garro, Andrea D'Ambrogio, and Andrea Giglio. A model-driven method to allow the distributed simulation of bpmn models. In *27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2018, Paris, France, June 27-29, 2018*, pages 0–0. Institute of Electrical and Electronics Engineers Inc., 2018.

[17] Alberto Falcone, Alfredo Garro, Andrea D'Ambrogio, and Andrea Giglio. Using bpmn and hla for engineering sos : lessons learned and future directions. In *2018 IEEE International Conference on Systems Engineering Symposium, ISSE 2018, Rome, Italy, October 1-3, 2018*. Institute of Electrical and Electronics Engineers Inc., 2018.

[18] Alberto Falcone, Alfredo Garro, Simon J. E. Taylor, Anastasia Anagnostou, Nauman R. Chaudhry, and Omar-Alfred Salah. Experiences in simplifying distributed simulation: The HLA development kit framework. *Journal of Simulation*, 11(3):208–227, 2017.

[19] Angelo Furfaro, Teresa Gallo, Alfredo Garro, Domenico Saccà, and Andrea Tundis. Requirements specification of a cloud service for cyber security compliance analysis. In *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*, pages 205–212. IEEE, 2016.

[20] Jim Highsmith and Martin Fowler. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001.

[21] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley Boston, 2011.

[22] Mira Kajko-Mattsson and YuLong Fan. Outlining a model of a release management process. *Transactions of the SDPS*, 9(4):13–25, 2005.

[23] Mayuram S. Krishnan. Software release management: a business perspective. In *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative Research, October 31 - November 3, 1994, Toronto, Ontario, Canada*, page 36, 1994.

[24] A. Lahtela and M. Jntti. Challenges and problems in release management process: A case study. In *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, pages 10–13, July 2011.

[25] Timo Lehtonen, Sampo Suonsyrjä, Terhi Kilamo, and Tommi Mikkonen. Defining metrics for continuous delivery and deployment pipeline. In *Proceedings of the 14th Symposium on Programming Languages and Software Tools (SPLST'15), Tampere, Finland, October 9-10, 2015.*, pages 16–30, 2015.

[26] N. Modig. *This is Lean: Resolving the Efficiency Paradox*. Rheologica, 2012.

[27] Björn Möller, Alfredo Garro, Alberto Falcone, Edwin Z. Crues, and Daniel E. Dexter. Promoting a-priori interoperability of hla-based simulations in the space domain: The SISO space reference FOM initiative. In *20th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2016, London, United Kingdom, September 21-23, 2016*, pages 100–107. Institute of Electrical and Electronics Engineers Inc., 2016.

[28] Björn Möller, Alfredo Garro, Alberto Falcone, Edwin Z. Crues, and Daniel E. Dexter. On the execution control of HLA federations using the SISO space reference FOM. In *21st IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017, Rome, Italy, October 18-20, 2017*, pages 75–82. Institute of Electrical and Electronics Engineers Inc., 2017.

[29] Kai Petersen and Claes Wohlin. Measuring the flow in lean software development. *Softw., Pract. Exper.*, 41(9):975–996, 2011.

[30] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[31] Daniel Ståhl and Jan Bosch. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87:48–59, 2014.

[32] Guido Steinacker. Scaling with microservices and vertical decomposition. *dev. otto. de*, 2014.

[33] André van der Hoek and Alexander L. Wolf. Software release management for component-based software. *Softw., Pract. Exper.*, 33(1):77–98, 2003.

[34] Manish Virmani. Understanding devops & bridging the gap from continuous integration to continuous delivery. In *Innovative Computing Technology (INTECH), 2015 Fifth International Conference on*, pages 78–82. IEEE, 2015.

[35] Sebastian Vöst and Stefan Wagner. Towards continuous integration and continuous delivery in the automotive industry. *arXiv preprint arXiv:1612.04139*, 2016.

[36] Sheng Yu and Shijie Zhou. A survey on metric of software complexity. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 352–356. IEEE, 2010.