A Software Architecture Framework for Quality-Aware DevOps

Elisabetta Di Nitto
Politecnico di Milano
Milan, Italy
elisabetta.dinitto@polimi.it

Pooyan Jamshidi
Imperial College London
London, UK
p.jamshidi@imperial.ac.uk

Michele Guerriero
Politecnico di Milano
Milan, Italy
michele.guerriero@polimi.it

Ilias Spais
ATC
Athens, Greece
i.spais@atc.gr

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

ABSTRACT

DevOps is an emerging software engineering strategy entailing the joined efforts of development and operations people, their concerns and best practices with the purpose of realising a coherent working group for increased software development and operations speed. To allow software architecture practitioners to enrich and properly elaborate their architecture specifications in a manner which is consistent with DevOps, we surveyed a number of DevOps stakeholders. We studied concerns and challenges to be tackled with respect to preparing a software architecture which is DevOps-ready, i.e., described in all details needed to enact DevOps scenarios. Subsequently, we introduce SQUID, that stands for Specification Quality In DevOps. SQUID is a software architecture framework that supports the model-based documentation of software architectures and their quality properties in DevOps scenarios with the goal of providing DevOps-ready software architecture descriptions. We illustrate our framework in a case-study in the Big Data domain.

CCS Concepts

•**Software and its engineering** → **Software notations and tools;** *Designing software;* System description languages;

Keywords

Architecture Frameworks, Model-Driven Design, QoS, QoD

1. INTRODUCTION

Today, most organizations face high market pressure, and their supporting ICT (i.e., Information and Communication Technology) departments are struggling to accelerate the delivery of applications and services while preserving production and operations stability [6]. On one hand, ICT operators lack understanding of the application internals including system architecture and the design decisions behind architectural components [2, 4]. On the other hand, development teams are not aware of operation details including the infrastructure, its limitations and benefits. In response to these challenges, DevOps emerges as a set of software engineering strategies to create a more cohesive working group intermixing development and operations people, concerns and approaches with the aim of increasing the speed to which a design changes reaches operational software within specified quality constraints [2].

In this paper we aim at enriching the state of the art in DevOps with an architecture framework called SQUID, that stands for "Specification Quality In Devops". The SQUID framework shows the following key features: (a) is based on the empirical investigation of software architecture stakeholders, concerns [1] and challenges around specifying and using software architectures in DevOps scenarios; (b) allows the functional and non-functional specification of software architectures in a manner which is most consistent with DevOps scenarios; (c) extends a well known software architecture framework, i.e., Philippe Kruchten's original 4+1 Views [9]; (d) conforms to the standard conceptual overview presented in the ISO / IEC 42010 standard for architecture description [8].

Illustrating SQUID we observed that it does indeed provide a valuable starting point to elaborate strategic software architecture descriptions within DevOps scenarios -these descriptions are in fact, *DevOps-ready*, i.e., they provide full details concerning the architectural aspects and properties needed in DevOps scenarios. Nevertheless, using SQUID in action as part of a Big Data application description featuring DevOps, we also observed that much work still lies ahead around several challenges, such as: (a) using SQUID descriptions as recipes in DevOps-based refactoring; (b) using SQUID descriptions for DevOps-based coordination and division of work; (c) provide appropriate DevOps tooling for the proposed architecture framework.

The rest of this paper is structured as follows. Section 2 elaborates on our investigation of the DevOps architecture

landscape. Sections 3 and 4 outline the original 4+1 framework and how it is augmented towards elaborating SQUID while Section 5 applies SQUID to elaborate a Big-Data application description from a real-life industrial scenario. Finally, Section 6 concludes the paper hinting to future work.

2. DEVOPS PEOPLE AND THEIR ISSUES

By means of market-scanning and Business-Model Canvas modelling [5] within the DICE EU H2020 project consortium's industrial partners we observed several key stakeholders, concerns and challenges in the DevOps context - we argue that understanding and covering these is critical to DevOps success. To understand the DevOps dimensions inherent in our scenario, we carried out over 30+ interviews as well as 3 focus-groups with DICE industrial partners concerning what are the practical requirements to support DevOps-based software development and operations for big data applications. We analysed results through coding and scenario analysis. The results of this exercise are outlined as follows.

2.1 Relevant Stakeholders and Concerns

Table 1 outlines the stakeholders that play a key role in specifying and using software architectures within DevOps scenarios according to our investigation. Column 1 names the stakeholder while Columns 2 and 3 outline a brief description of the stakeholder's role in the context of DevOps and the concerns this role entails.

The table essentially shows how the stakeholders and roles typically involved in software engineering [13] are accompanied by a number of stakeholders (e.g., Middleware communities or standardization bodies) that do not have a direct relation to the product but account for the product's organizational stability, e.g., so that the DevOps cycle may be enacted in a long-lived fashion and without hindrance. Conversely, imagine a scenario where a product is being developed in a DevOps fashion but software architects and product owners do not account for standards or middleware providers / communities. In this same scenario, proceeding without measurement or quantification of the middleware community health may lead to community failure and middleware discontinuation.

2.2 DevOps: Industrial Challenges

The following key challenges were elicited as part of our investigation:

1. **Lack of automated tools for quality-aware DevOps:** industries recognise the lack of development and continuous delivery of products that has been verified in terms of quality assurance. In other words, there exists many different frameworks that software vendors can adopt in order to develop their applications. However, there is no automated mechanism to enable them to verify software quality across such different technological stack in a automated fashion.

2. **Lack of automated tools for DevOps-based architecture improvements:** In order to develop a modern IT application, software vendors require to adopt an architectural style in order to integrate different components of the system. During development, the complexity of such architecture will be increased and there is no automated tool to enable designer to improve the architecture based on performance bottlenecks and reliability issues that have been detected based on monitoring of the system on real platforms.

3. **Lack of tools and methods adaptable to heterogeneous DevOps maturity in industry:** Developing and maintaining modern IT applications (e.g., data-intensive applications) is key to IT market diversity, spanning from big industrial players and small-/medium-enterprises. However, there exists a huge variety of software processes in this diversity. Such diversity is seldom taken into account from a methodological and technological standpoint. For example, very few evaluations show the applicability of certain data-intensive design methods within IT companies consistent with CMMI (Capability Maturity Model Integration) Level 4 (Quantitatively Managed) or level 2 (Managed), yet, the industrial adoption of those design methods may depend greatly on such evaluations.

4. **Lack of Continuous Analytics Frameworks for Business Needs and Technical QoS:** All types of organizations need to move fast, and need to align IT assets to their needs (create new assets or adapt existing ones). Therefore, ICT departments need to assure that their ICT platforms and infrastructures are flexible enough to adopt business changes, regardless of the type of IT infrastructures used to create those data intensive applications, whether they are built using open-source software or data services offered by public cloud providers. In other words, they should avoid technological lock-in as much as possible.

5. **Lack of Explicit Support for Continuous Architecture Refactoring:** The tenets behind DevOps entail producing actionable architectures that can be operated and monitored as soon as possible so that aspects such as efficiency, reliability and safety of applications can be improved incrementally in testing and production environments, with metrics and data that feedback directly and quickly to development teams for faster testing, improvement and adjustment to meet service level goals.

6. **Continuous Fine-Grained Application Monitoring and Anomaly Detection:** Access to live data gathered by monitoring engines should also provide performance and reliability data for observed applications in production to gauge the need for identifying outliers and detecting any anomaly that may harm continuous business processes that are dependent to such data intensive applications. Moreover, such monitoring data should assist application architects and developers in building toward an optimized target infrastructure.

7. **Continuous Configuration Optimization:** modern day applications typically spend expensive resources that are offered to companies via public clouds. Therefore, it is of utmost importance to optimize such applications in order to demand less resources and produced more output. Therefore, organizations require to have automated tools to optimally configure such application without the need to hire experts to optimize their

Table 1: Critical DevOps Stakeholders and Concerns, An Overview.

Stakeholder	Description	Concerns
Systems Designer / Architect	In the Context of DevOps, the software and systems designer or software architect becomes the reference figure with the responsibility of setting up the general conceptual overview (i.e., the architecture) of the product to be as well as its platform and infrastructure conterparts. Finally, the DevOps software architect[2] is the reference figure adapting the software architecture specification to welcome feedback streams from systems' operation.	Using Architectures as Work-overview and division across the DevOps team; Architecture quality by means of -Ops feedback; Architecture profiling to Dev and Ops sides of teams; Architecture monitoring;
Systems Developer / Programmer	In the context of DevOps the programmer becomes a technical person capable of developing a system while taking into account its operational aspects as well. The DevOps programmer is capable of assessing the impact of architectural changes both in software development and operations.	Meeting operations' expectations; mirroring the infrastructure-as-code counterpart [2];
Infrastructure provider	In the context of DevOps a IaaS provider is any person or organization technical and operationally responsible for making the service architecture usable by cloud service users.	Standardization; Interoperability; Monitoring; Accountability[10, 11];
Quality Assurance Expert / System Manager	The QA expert or systems manager is any person capable in evaluating the non-functional aspects and properties of the architecture (e.g., performance, privacy, etc.).	Architecture Iterative Enhancement; Architecture Modifyability [3];
System Tester	The tester transforms architectural and development information into viable tests that can be (re-)executed in an automated fashion;	Test automation; Test serialization; Test remote executability;
Middleware Provider	DevOps solutions are nestled around many open- and closed-source products that are assembled together and choreographed by adeguate middleware. Middleware providers are organizations, communities or people responsible for said middleware and their concerns are focused on the reliability of their product as much as the organizational health [12] of the surrounding community or organization since the middleware reliability rests heavily, e.g., on the engagement of the surrounding community in improving it in a DevOps fashion.	Middleware reliability; Community health;
Standardization Body	In the context of DevOps, standardization bodies cover the role of offering ways to speed-up by on-the-fly composition by standardization. For example, the use of TOSCA can ensure the interoperability between products and services compatible with and operating according to the TOSCA standard.	Standard adoptibility; Standard breadth (the ability to encompass multiple technologies over multiple domains); Standard depth (the ability to support multiple granularity and abstraction levels)

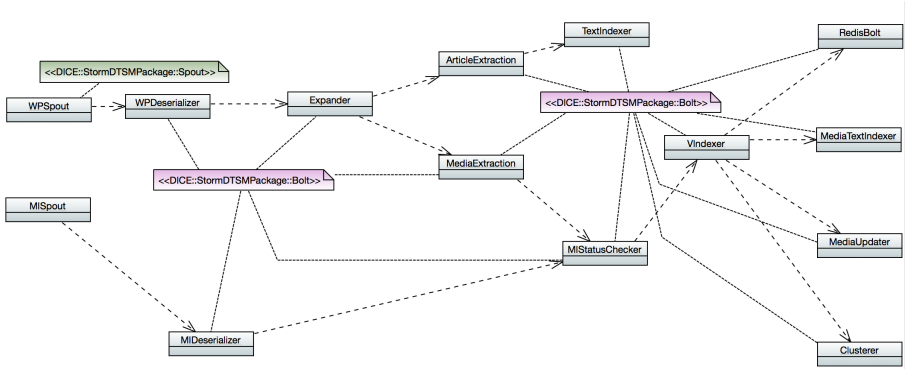

Figure 1: the Social-Sensor App, representing the SQUID development view on the StormFocusedCrawler component from Fig.3.

applications. This becomes even more critical for applications that are deployed over large clusters and harness the potential of big data, e.g., for the purpose of business intelligence and analytics.

From the stakeholders, concerns and challenges outlined above we observed that: (a) software architecture views conforming to the ISO/IEC 42010 Standard must be augmented with more fine-grained architecture descriptions specific to various framework and middleware involved in DevOps applications development - this is critical to have an overall understanding of software architecture traits that are affected by operational concerns; (b) the software architecture needs to be augmented with deployable images, i.e., there needs to be an architectural representation that allows inferring a concrete blueprint containing infrastructure, platform as well as application topology; (c) finally, there needs to be model-based synchronization of said augmentations and views, to fully support the DevOps feedback loop.

3. THE 4+1 ARCHITECTURE FRAMEWORK

In response to the DevOps players and challenges previously outlined in Sec. 2 we observed a number of required views and notations missing from the state of the art but curiously reminiscent of Philippe Kructhen's [9] classical 4+1 software architecture views framework. Although similar, the 4+1 views model required specialisation and additional re-elaboration. As a result, the DevOps architecture framework we advocate is rather simple and can be interpreted as a specialisation or refinement of the 4+1 software architecture framework. The original 4+1 framework offered notations, views and architecture concerns for:

- **Logical Software Architecture Views** - that is, representations of architecture elements from a functional perspective, where logical communication, sequences of actions and class specifications are typical concerns;
- **Development Software Architecture Views** - that is, representations of architecture elements from the typical "blueprint" perspective, allowing for authority management, division-of-work, architecture reasoning;
- **Process Software Architecture Views** - that is, representations of architecture elements from a process perspective where concurrency, distribution and other non-functional aspects are discussed, verified and acted upon;
- **Physical Software Architecture Views** - that is, representations of architecture elements as mapped to the physical nodes or clusters where architecture elements will exist and function;
- **Transversal Scenario Views** - these capture the essential scenarios to be supported by the software architecture in question or its general operation;

4. THE SQUID FRAMEWORK

Within the scope of the DICE EU H2020 Project, we are developing conceptualisations and notations to support a DevOps way of working in the scope of Big Data applications, where DevOps becomes a key enabler since it allows to deliver quality results with speed and continuous improvement/innovation. SQUID is our extension to the views originally intended in the 4+1 model (see Sec. 3) and stems from five key refinements to Philippe Kruchten's original notation for software architecture representation - these refinements emerged from our industrial study and reflect DevOps-ready software architectures:

1. the logical architecture view needs to be synchronised (e.g., by means of model-to-model transformation) with: (a) a quality verification view for each quality property relevant to application domain; (b) for each quality property in (a), a monitoring view stemming from the operations view; (c) a process view specific to the quality verification and monitoring views. This is because DevOps advocates continuous improvement and continuous architecting by direct interaction of developers, designers, infrastructure engineers and middleware operators - as a consequence, software architecture elements in the logical view need to be quickly transformed into analysable notations with different levels of granularity (e.g., entire application vs. single middleware) and the results need to be quickly and continuously fed back into the architecture.

2. the development software architecture view needs to be synchronised (e.g., by means of model-to-model transformation) with an operations software architecture view, i.e., a view that reflects the operational image for software architectures at hand, e.g., imagine a Hadoop cluster and the various nodes involved. At the same time, the operations software architecture view is a specialisation of the Physical software architecture view where all elements are (a) mapped to their physical nodes and (b) able to feed runtime behaviour information back to the development software architecture view. This refinement tackles the lack of explicit support for continuous architecture refactoring as a result of direct interaction of design and development artefacts with what can be learned from their operation. As a result of modelling these views, stakeholders involved in design, development, operations and infrastructure can join forces in tuning said shared views with operational and monitoring insights.

3. there needs to be a specific process architecture view per every software component or framework involved or used in the architecture since both the single and aggregated performances of these components may need to be explored incrementally, e.g., think of refining a data intensive application which features a single Apache Spark component but with the need for several optional components, e.g., YARN or Kafka. These additions may actually hamper performances rather than boosting them - as highlighted in Sec. 2 we found that incremental evaluation is a must-have in DevOps scenarios. The separate process views need to be shared across development and operations people so that, on the one hand, they can be used to analyse and develop better applications, on the other hand, they can be used in combination with logical and development views to infer deployable software architecture descriptions.

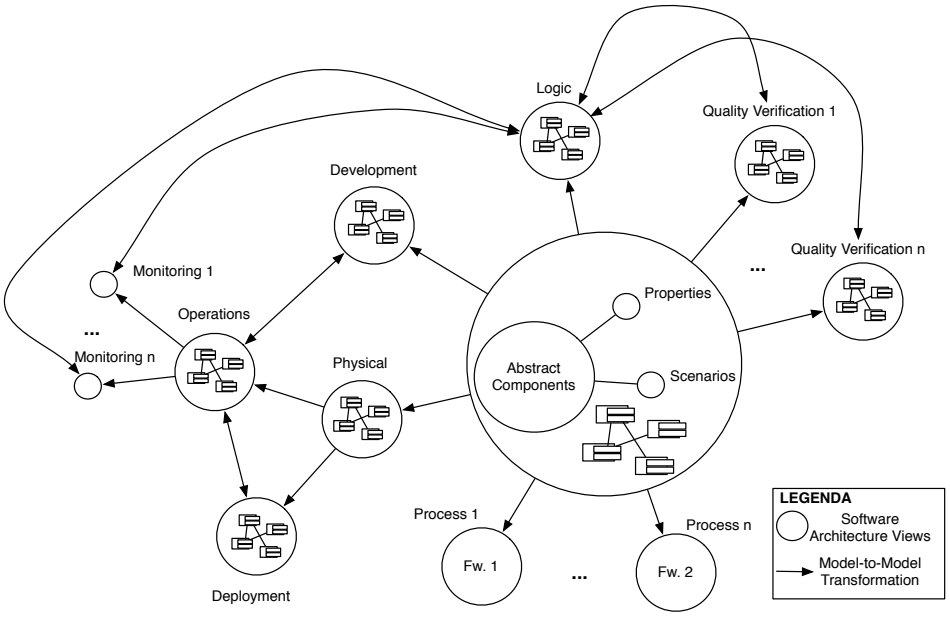

Figure 2: the SQUID Architecture Framework, an overview.

4. there needs to be an actionable (i.e., runnable or deployable) version of the physical software architecture view - this version needs to be consistent with the infrastructure-as-code paradigm [2], i.e., it needs to be possible to generate deployable infrastructure code from the actionable physical view. This is necessary to support the DevOps feedback loop in possibly co-evolving the software architecture jointly with the underlying infrastructure. Also, this view is "fed" continuously to the operations view, e.g., to elicit and act upon runtime feedback. Finally, this view is inferred directly from other architectural representations that contain process, logical structure and properties of architecture specifications.

5. Finally, much like the transversal scenario view from the 4+1 framework, a single unifying view shall enable the specification of an abstract component-based software architecture representation. However, in the context of DevOps this representation is scenario- and property-driven since: (a) it shall allow the specification of the scenarios and use-cases that the architecture is supposed to support; (b) it shall allow the mapping of said scenarios to software architecture elements, for example, a word-counting scenario must be mapped to a batch-processing component; (c) it shall allow the mapping of desired quality of service (QoS) and quality of data (QoD) properties with specific architecture elements, for example, lossless word-counting shall be mapped to a fault-tolerant batch-processing component. The scenario and property driven specification is also consistent with the intended use of descriptions such as TOSCA, i.e., the "Topology and Orchestration Specification for Cloud Applications".

A high-level overview of the SQUID architecture framework is outlined in Fig. 2. At the center of the framework, a single abstract view specifying abstract components, their properties and the scenarios they support. Around the center, a number of ancillary views that cover logic, quality verification, development, operations/monitoring, process and physical representations of software architectures.

5. SQUID IN ACTION: A CASE FROM INDUSTRY

To illustrate and evaluate the workings of SQUID we applied the principles coded in it to describe the application architecture owned by one of the industry partners part of the DICE project consortium. More in particular, our partners in ATC Inc. own a social-sensing application that harnesses Storm-based complex-event processing to mine data from online sources and elicit essential topic-based information abstractions called DySCOs.

In the context of the social-sensing application in question, we provided our partners with several diagrams including: (a) a software architecture view that describes the high-level overview of the application using abstract components (corresponding to the SQUID logical architecture view, see Fig. 3); (b) a software architecture view that describes the inner topological decomposition of the Storm framework instance running within the social sensing app (Fig. 1, corresponding to the SQUID process architecture view focusing on the StormFocusedCrawler component from Fig. 3).

To evaluate the resulting architecture description and its value in refactoring the software architecture in a DevOps fashion, we inspected the architecture description in a joint workshop with our ATC partners.

We observed that the architecture description does in fact offer the right level of detail to enact architecture decision-making and design refactoring. However, the value of SQUID representations is still limited, given the lack of actionable model-2-model transformations that would offer the automation layer intended behind SQUID. In the Sec. 6 we discuss further this limitation and our finding, with a plan to tackle it in the rest of the DICE EU H2020 project agenda.

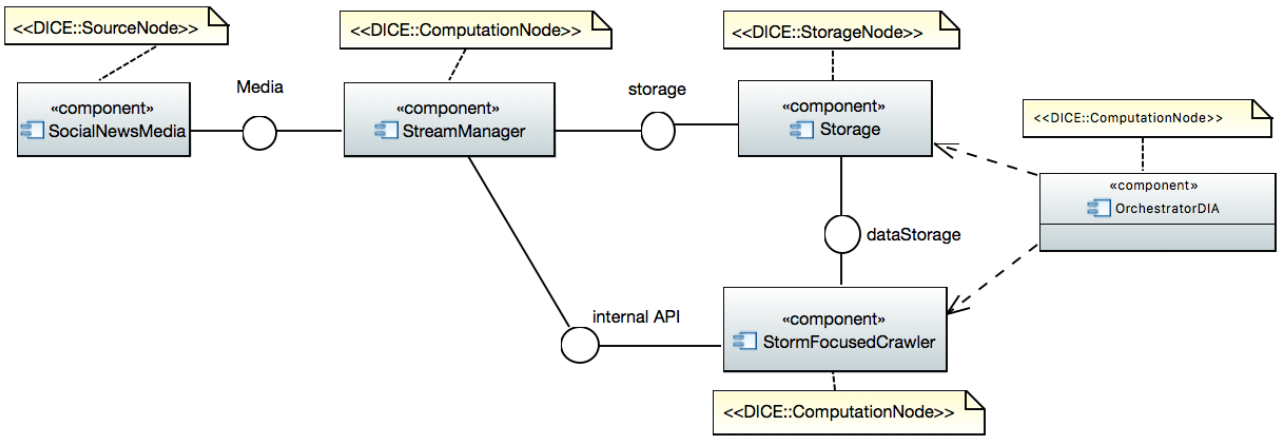

Figure 3: the Social-Sensor App, representing the SQUID Logic View.

6. CONCLUSIONS AND FUTURE WORK

DevOps is industrial software engineering's response to the increasing "need for speed" emerging within current IT markets. In layman's terms, DevOps entails a series of strategies that are aimed at reducing the friction and timing in delivering and operating an architectural change, feature or improvement [2]. In so doing, DevOps promises to deliver software speed in a shorter time-span. However, we observed that specifying software architectures such that they can work in DevOps scenarios is no easy task.

First, this paper describes the DevOps architecture landscape by outlining the key stakeholders, concerns and challenges in DevOps scenarios.

Second, we proposed SQUID, a software architecture framework [1] as a refinement of the widely known and used 4+1 model originally introduced by Kruchten [9].

Finally, by applying SQUID on a real-life industrial scenario, we savoured SQUID's pros and limitations. We found that SQUID offers a valuable basis for describing and (tentatively) achieving quality in DevOps scenarios but much work stil lies ahead with using software architecture frameworks to achieve Quality-aware DevOps especially in terms of tooling, applicability and user-friendliness. In the future we plan to apply SQUID in action within the DICE H2020 EU project for the purpose of fully describing DevOps-ready software architecture descriptions in the context of Big Data applications. In so doing, we plan to support SQUID from a methodological and tooling perspective by refining the (soon to be released) DICE IDE to support the SQUID architecture framework and the transformations intended within it, for which we already began a preliminary specification [7].

7. ACKNOWLEDGEMENTS

The authors' work is partially supported by the European Commission grant no. 610531 (FP7 ICT Call 10), SeaClouds and by the European Commission grant no. 644869 (H2020 - Call 1), DICE.

8. REFERENCES

[1] Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*. Addison Wesley, 1998.

[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] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software 69*, 1–2 (Jan. 2004), 129–147.

[4] Bersani, M. M., Marconi, F., Tamburri, D. A., Jamshidi, P., and Nodari, A. "continuous architecting of stream-based systems". 115–121.

[5] Fritscher, B., and Pigneur, Y. Visualizing business model evolution with the business model canvas: Concept and tool. In *CBI (1)* (2014), IEEE, pp. 151–158.

[6] Gebert, S. Devops und continuous delivery, 7 2014.

[7] Guerriero, M., Tajfar, S., Tamburri, D. A., and Nitto, E. D. "towards a model-driven design tool for big data architectures". In *proceedings of the 2nd International Workshop on Big Data Software Engineering - BIGDSE* (2016), IEEE.

[8] ISO/IEC/(IEEE). ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems, 07 2007.

[9] Kruchten, P. The 4+1 View Model of Architecture. *IEEE Software 12*, 6 (1995), 45–50.

[10] Tamburri, D. A., and Lago, P. Satisfying cloud computing requirements through agile service networks. *Proceedings of SERVICES 2011* (2011).

[11] Tamburri, D. A., Lago, P., Dorn, C., and Hilliard, R. Architecting in networked organizations. In *WICSA* (2014), IEEE Computer Society, pp. 247–250.

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

[13] van Vliet, J. C. *Software engineering - principles and practice (3. ed.)*. Wiley, 2008.