Towards multi-concern software development with Everything-as-Code

Vlad Stirbu
CompliancePal, Tampere, Finland

Mikko Raatikainen
University of Helsinki, Finland

Joel Röntynen
Solita, Helsinki, Finland

Vlas Sokolov
Solita, Munich, Germany

Timo Lehtonen
Solita, Tampere, Finland

Tommi Mikkonen
University of Jyväskylä, Finland

***Abstract*—As software is becoming a central element in our lives, more and more stakeholders have concerns. Unlike today, when developers stop their coding activities to satisfy these stakeholder concerns, we propose dealing with them as part of the coding workflow, the central element of programmers' daily duties. This can be achieved by extending the approach that we call Everything-as-Code (EaC) beyond software engineers and operators.**

SOFTWARE is everywhere and implements many new functionalities in products and services. Therefore, many non-functional or regulatory *concerns* have emerged and must be assured of being appropriately addressed. For example, safety and effectiveness in medical devices are assured by conformance with regulatory processes and requirements, covering specifically the software development lifecycle [4], introduced to protect against errors that the developers might otherwise make. Similarly, the EU's general data protection regulation (GDPR) [14] guards against misused personal information. The need for trustworthy AI [3] and ethical guidelines for its use [8] have recently been proposed to ensure that the technology use is fair and does not cause negative side effects. While these examples are just the tip of an iceberg, the trend is prevalent, and few systems and their developers are – and even should not be – free from similar assurance concerns.

The resulting situation is not straightforward for the software *developers* of these products and services. The developers face multiple concerns for software emerging from domains other than the development of the software solution itself. These new concerns often imply new stakeholders ensuring meeting their particular concerns, which can even block a non-compliant release. For them, dependability and security assurance is

not enough, but a specific audit trail is required, resulting in additional interaction and sidesteps in the development process serving their needs, such as additional documents and tools. Thus, the developers must adapt their ways of working, incorporating yet another new tool or document.

In this paper, we argue for a development approach that places the software and the way software developers work at the core of the system and related product development activities. As the baseline, we use the *Everything-as-Code (EaC)* paradigm [15]. While many software development and operation specialists have already adopted practices of EaC, e.g., as Infrastructure-as-Code, our proposal extends EaC beyond software engineers and operators. We argue that the concept of EaC is extendable and can capture many non-functional or regulatory concerns requiring traceable information for assurance. Towards this end, we describe two different industrial cases to demonstrate capturing information for medical compliance and AI-Explainability as forms of EaC. Then, upon allowing the developers to advance at the pace determined by iterative development with continuous delivery of small increments, we elaborate on how to capture and embed multiple other concerns in this process while respecting the spirit of today's software development.

EaC state-of-the-practice

Today's software engineering practices, especially DevOps [1], [9], focus on short iterations and shortening the time from feature implementation till availability to the end-users with many automated activities. Once developers have implemented a feature, it is integrated into the code baseline after passing verification, delivered and deployed to a production environment, and continuously monitored. Through extensive automation, the integration and delivery processes leverage the capabilities of version control tools, such as GitHub and GitLab, to automatically execute custom pipelines that build the software, run tests, create the needed infrastructure for the execution environment from a versioned configuration, and deploy the software into this environment. Concepts such as *infrastructure-as-code* or *configuration-as-code* are applied because the steps and activities are specified as "*as-code*" and performed efficiently, at a relatively low cost, by code-based scripts familiar to software developers and operations specialists.

This code-based approach has been applied even further. The increased speed pressures organizations that expose their functionality to third parties to deliver the documentation as developer portals. They contain application programming interfaces or software development kits documentation and technical guidelines for using them, deployed at the same pace as the DevOps team delivered the new functionality. The documentation specialists develop *Docs-as-Code* practices for managing and releasing documentation that relies on the same tooling as the development teams. Similarly, as software-intensive systems have become more connected and make use of 3rd party software components, activities related to cybersecurity and software supply chain management have been integrated as *DevSecOps* into the DevOps practices.

Towards broadening non-software development concerns in EaC

While EaC as described above is being applied to domains closely related to software developers, such as infrastructure or security, we bring forward that the concept of EaC is extendable beyond technical stakeholders, such as for regulation or governance. The information required these stakeholders is captured in code assets similar as any other information in EaC.

With new stakeholder concerns implemented by broadening the EaC envelope, some constants remain: to keep the development team moving at software speed, the tooling for continuous software engineering revolves around source code and, thus, assumes that GitHub-style support is available for all tasks, leveraging facilities such as versioning and automatic workflows. In other words, with a new stakeholder with novel concerns impacting the software or its development process, the stakeholder should adapt to the software development environment and tooling instead of developers actively providing separate views to any concern that the stakeholders may raise.

In addition to enabling value to the end-users and other stakeholders as soon as possible, this EaC way of working centered around code and

Git respects the developers' and other specialists' workflows. As software development is largely about creative work, interwinding other concerns into the development process allows the developers to maintain their cadence, making them more productive [7]. In contrast, interaction among stakeholders can lead to workflow interruptions, whose resolution can delay development [6].

Consequently, we argue that the next logical step and continuation of EaC is the expansion beyond software engineers and operators. While this may sound radical, we have already made software developers consider the terminology of other fields, up to the point of institutionalizing it as the domain-driven design [2], in which the software adopts the terminology of the field it is serving. With EaC, as the code is the central concept, other fields that relate to it must operate on its terms by borrowing development metaphors. For example, the pull request, the developers' main change management event, can accommodate other specialists during the review phase. However, we do not expect a layperson to master code and Git as a developer, but proper tooling and abstracting interfaces on top of code and Git are needed to make the EaC accessible. Next, we provide two concrete examples from the industry. They cover Compliance-as-Code and AI Explainability-as-Code, and both are commercial endeavors.

Case I: Medical Compliance as Code

Before bringing products to the market, medical device manufacturers must demonstrate compliance with safe and effective use regulations. Traditionally, the regulatory compliance evidence has been collected in a waterfall fashion. However, with the trend of delivering more innovations by software, manufacturers face the challenge of adapting their way of working so that the regulatory activities are aligned – in addition to the product lifecycle – with the iterative and incremental practices of agile development.

Scenario: Handling 3rd party software

Today, no software is built in isolation but with a functionality-reuse mindset so that a manufacturer develops only the essential functionality of the product. At the same time, the rest comes from various third-party components – either commercial or open-source ones – as *software of unknown pedigree (SOUP)*. Regardless of who developed the software components, the manufacturer has the final quality responsibility. Therefore, the manufacturer must ensure that the respective components have been developed with the rigor expected for medical software.

The risk analysis process for SOUP is a source of friction between the software developers and regulatory affairs professionals. The software developers move swiftly but are not accustomed to performing the risk management activities during their daily routine. Likewise, the regulatory affairs specialists are not accustomed to the high velocity of increments at which software developers introduce changes, sometimes incorporating SOUP components. Keeping the software development and risk management aligned using the waterfall-influenced *documentation sprint* is not optimal. The regulatory tasks must be performed on the accumulated change-set as a sum of all increments. As the changeset grows, tracking evolution becomes difficult with artifacts that potentially update several times. A more effective approach, depicted in Figure 1, is to expand the review phase of the pull request to include regulatory affairs professionals in those requests that need their attention. The regulatory affairs professionals can perform the necessary tasks on a small increment and directly interact with the rest of the team that introduced the change. All involved have fresh and accurate information about the nature of the change.

As a concrete implementation, a GitHub-integrated tool called CompliancePal (https://compliancepal.eu/) can detect if a pull request introduces new or modifies the existing SOUP components [13]. The tool requires a software developer to justify the use of the SOUP component with functional, performance, and possible hardware requirements. Next, this information is presented to a regulatory specialist using a custom web user interface that emphasizes the changes in SOUP dependencies in a UML-like fashion. Together with the software developer, they determine if the change introduces new risks for which they must plan mitigation. The workflow completes successfully only when the new SOUP component has been introduced in the software decomposition

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MS.2022.3167481, IEEE Software

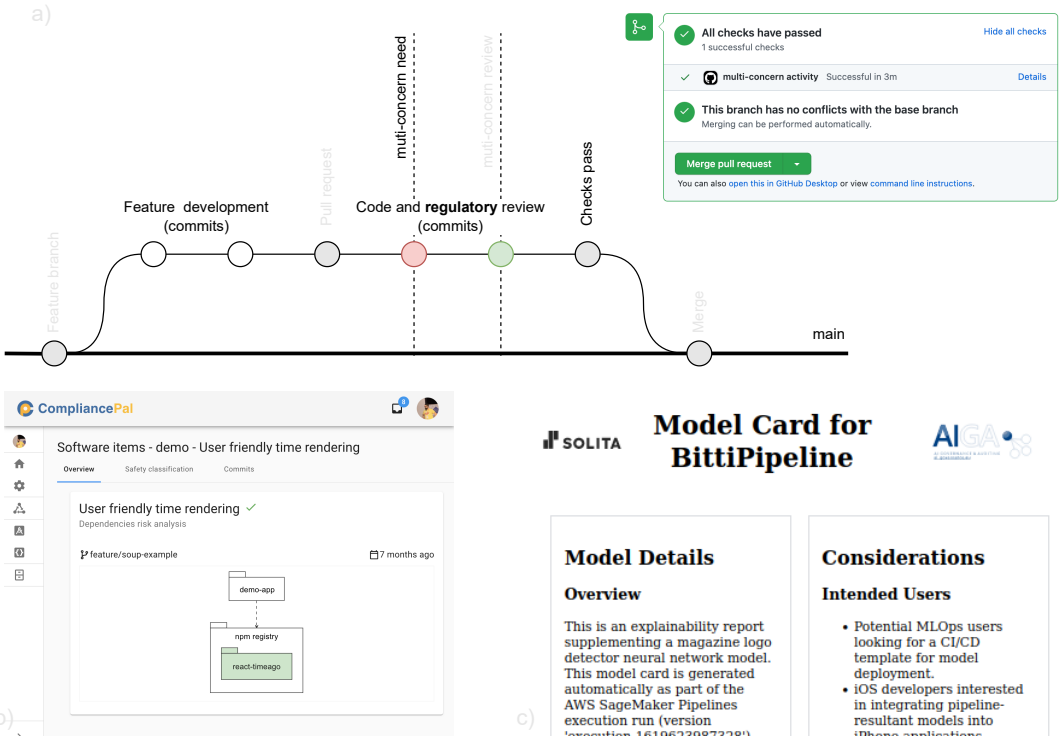

Figure 1: A conceptual synthesis of two cases demonstrating pull request flow enhanced with multi-concern views: a) the identified and reviewed multi-concern activity and the record of the activity being performed recorded using GitHub's pull request check facility, acting as a quality gate; b) the semantic diff view emphasizing new SOUP components introduced in the pull request; c) the visualization of a model card in which information is integrated into the pull request flow..

documentation.

Lessons learned

The compliance workflow integrated into a pull request ensures that the regulatory activities are performed upon changes. The required regulatory documentation and the evidence of performed activities are collected and tracked in a software development environment using the developers' way of working and tooling. The regulatory affairs specialists and software developers work together using *compliance-as-code* practices to achieve common product goals in each increment.

The feedback received from the medical device manufacturers that used CompliancePal encourages expanding the range of regulatory activities handled in this fashion. The initial worry that the regulatory affairs professionals were not at ease with the pull request workflow and the user interface metaphors used by the Git vendor were overstated. They learned the tools fast and, helped by the custom views, worked in sync with the developers.

Case II: Explainable AI by Model Cards

Our second example focuses on the work of data scientists and challenges in communication, particularly how to document and communicate pipeline architecture and Machine Learning (ML) model decisions to foster transparency and support ML model validation. Model card [12] is an approach that aims to clarify ML models' intended use cases and avoid their misuse. Intended as documentation for a trained ML model, a model card contains information about the intended use case, benchmark evaluation in relevant conditions, such as demographics or geographic location, or any other information that the creators consider necessary for the proper use. Parts of model cards can be automatically extracted and generated, but other parts require a data scientist

to record the design decisions and rationales manually. Therefore, the model cards can form an additional task for developers to tackle and consider in their daily routines.

Sample application: Object detection

To assess the feasibility and fitness of the model cards approach in the consultancy business, we set up a continuous delivery pipeline for an object detection task to detect magazines based on their logos in pictures of magazine racks. The pipeline is an open-source, version-controlled, reproducible solution hosted in a GitHub repository (https://github.com/solita/mlops-pipeline-sagemaker). The pipeline runs on AWS SageMaker Pipelines framework and orchestrates a series of standard data-related steps, such as data augmentation and processing, as well as the automated training, testing, and deployment of the object detection neural network with a TinyYolo [5] architecture.

As the final step of the pipeline, a model card is automatically generated, with its metadata stored both in JSON format so that a machine-readable format can be used to generate different views and an HTML for web-based usage. We focused on creating a view for data scientists looking for other use cases for the trained model. The model card contains sections for overview and model details, considerations for the use of the model (intended use cases, limitations and tradeoffs, and ethical considerations), performance metrics and examples of the model prediction in the evaluation set, and training parameters. The data scientist needs to fill in the overview and the considerations of use for new versions of the model, while the other sections are automatically updated on every run of the pipeline.

Lessons learned

Using the EaC approach, where all aspects of the pipeline will be defined as code or templates in a repository under version control, enables a shared understanding in a cross-functional team. Having everything under version control makes the development better traceable, as all changes are tracked. Over time, this allows more people to see and validate the decisions and assess the rationales. Furthermore, the systematic way of developing models using the pipeline will establish best practices within the company, allowing code and template reuse in a scalable way.

The data scientists working with the model card found it beneficial to have to think about and record the limitations of the model already during the development process, as these considerations affected the decisions about how to train the model. For example, for better accuracy in a typical phone use scenario, they started using tilted photos during training.

Although the original proposal presented the model card as a visual representation, recent developments (https://github.com/tensorflow/model-card-toolkit) propose a programmatic mechanism to generate the model cards and a machine-readable serialization that facilitates the consumption of model cards by scripts in ML pipelines. While this is a step in the right direction, the proposed information is still limited to technical stakeholders, such as data scientists or machine learning engineers. To fulfill the model card's potential, the information should accommodate the needs of other stakeholders. The model card metadata becomes a model from which various views intended for different stakeholders can be derived, in a similar fashion as various software architectural views can be derived from a single software model. A model card becomes a must-have artifact that conveys the information about the model's intended use or performance and other activities conducted with the model (e.g., regulatory risk management) or metadata about data sets that facilitate downstream testing.

Discussion and Key Takeaways

The core element that contributes to the ongoing success of DevOps practices is the collaboration culture between the software development and operations organizations [11]. Sharing information and expanding the skill-set of the team members, as well as performing activities together, induces a sense of shared responsibility.

The above cases demonstrate how the EaC approach can be extended to safety-critical systems and AI/ML applications for concerns of ensuring that the systems are safe and effective and that ML models do not introduce unintended side effects, respectively. As the software engineers or data scientists already possess much knowledge

needed by the other stakeholders, they are active information producers. In contrast, other stakeholders typically use or assess the information without actively modifying it. Therefore, it is natural that the workflows that facilitate information sharing and collaboration are built around the tools used by the software engineers, especially Git for version control and pull requests for change management. Additional tools can generate the custom information views required by other stakeholders, making Git the ledger system for EaC.

Our contribution shows that by facilitating the collaboration of regulatory affairs specialists, generally understood as performing quality assurance tasks, with software engineers and data scientists, the new multi-disciplinary team improves the quality outcome in compliance and AI-explainability. The results, in line with the expectations of DevOps adoption [10], provide practical examples of how EaC is indeed able to handle the concerns of non-coder stakeholders while at the same time not interfere significantly with developers, which are able to maintain their high delivery pace, and way of working.

REFERENCES

1. Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Software*, 33(3):94–100, 2016.
2. Eric Evans and Eric J Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
3. Luciano Floridi. Establishing the rules for building trustworthy AI. *Nature Machine Intelligence*, 1(6):261–262, 2019.
4. International Electrotechnical Commission. IEC 62304:2006/A1:2015. Medical device software - Software life-cycle processes, 2015.
5. Ivan Khokhlov, Egor Davydenko, Ilya Osokin, Ilya Ryakin, Azer Babaev, Vladimir Litvinenko, and Roman Gorbachev. Tiny-yolo object detection supplemented with geometrical data. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.
6. Kati Kuusinen. Value creation and delivery in agile software development: Overcoming stakeholder conflicts. In *IFIP Conference on Human-Computer Interaction*, pages 123–129. Springer, 2017.
7. Kati Kuusinen, Helen Petrie, Fabian Fagerholm, and Tommi Mikkonen. Flow, intrinsic motivation, and developer experience in software engineering. In Helen Sharp and Tracy Hall, editors, *Agile Processes, in Software Engineering, and Extreme Programming*, pages 104–117, Cham, 2016. Springer International Publishing.
8. Stefan Larsson. AI in the EU: Ethical guidelines as a governance tool. *The European Union and the Technology Shift*, pages 85–111, 2021.
9. Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.
10. Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. Adopting devops in the real world: A theory, a model, and a case study. *Journal of Systems and Software*, 157:110384, 2019.
11. Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of devops. In Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 212–217, Cham, 2015. Springer International Publishing.
12. Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.
13. Vlad Stirbu and Tommi Mikkonen. CompliancePal: A tool for supporting practical agile and regulatory-compliant development of medical software. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 151–158. IEEE, 2020.
14. Paul Voigt and Axel Von dem Bussche. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.
15. Afzaal Ahmad Zeeshan. Automating everything as code. In *DevSecOps for. NET Core*, pages 109–162. Springer, 2020.

Vlad Stirbu is the founder of CompliancePal. He received his D.Sc (Tech) in software engineering from Tampere University of Technology. His research interests include continuous software engineering practices in the context of regulated industries. Contact him at vlad.stirbu@compliancepal.eu

Mikko Raatikainen is a researcher focusing on software engineering and business at University of Helsinki. He holds D.Sc. (Tech.) in software engineering. Contact him at mikko.raatikainen@helsinki.fi

Joel Röntynen is a data scientist and quantum software engineer working for Solita, based in Helsinki, Finland. He holds a PhD in theoretical physics. Contact him at joel.rontynen@solita.fi

Vlas Sokolov is a data scientist working for Solita, based in Munich, Germany. He holds a PhD in astrophysics. Contact him at vlas.sokolov@solita.fi

Timo Lehtonen is a senior software designer at Solita, a consultancy company for software intensive data solutions. He defended his doctoral degree in the field of software engineering in 2017. Contact him at timo.lehtonen@solita.fi

Tommi Mikkonen is a professor of software engineering at University of Jyväskylä. He received his doctoral degree in 1999 from Tampere University of Technology. Contact him at tommi.j.mikkonen@jyu.fi