Toward Multiconcern Software Development With Everything as Code

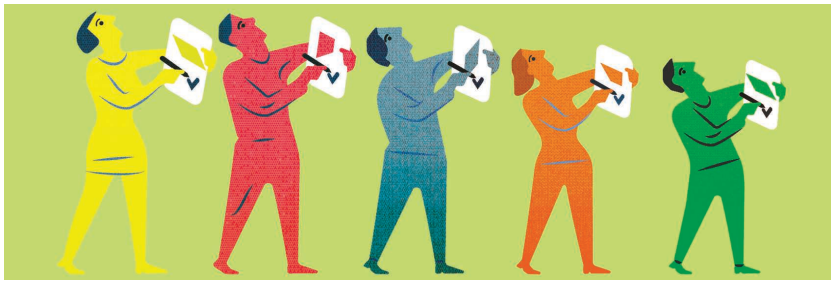
Vlad Stirbu, CompliancePal

Mikko Raatikainen, University of Helsinki

Joel Röntynen, Vlas Sokolov, and Timo Lehtonen, Solita

Tommi Mikkonen, University of Jyväskylä

// As software is becoming a central element in our lives, more stakeholders have concerns. Unlike today, when developers stop their coding activities to satisfy these stakeholder concerns, we propose dealing with them as a part of the coding workflow. //



Digital Object Identifier 10.1109/MS.2022.3167481
Date of current version: 20 June 2022

SOFTWARE IS EVERYWHERE and implements many new functionalities in products and services. Therefore, many nonfunctional 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,[1] introduced to protect against errors that developers might otherwise make. Similarly, the European Union's General Data Protection Regulation[2] guards against misused personal information. The need for trustworthy artificial intelligence (AI)[3] and ethical guidelines for its use[4] have recently been proposed to ensure that technology use is fair and does not cause negative side effects. Although these examples are just the tip of the 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 noncompliant 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, developers must adapt their ways of working, incorporating yet another new tool or document.

In this article, 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.[5] Although many software development and operation specialists have already adopted the practices of EaC, e.g., 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 nonfunctional or regulatory concerns requiring traceable information for assurance. Toward 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 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 development operations (DevOps)[6,7], focus on short iterations and shortening the time from feature implementation until availability to 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, integration and delivery processes leverage the capabilities of version-control tools, such as GitHub and GitLab, to automatically execute custom pipelines that build software, run tests, create needed infrastructure for the execution environment from a versioned configuration, and deploy 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 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 third-party software components, the activities related to cybersecurity and software supply-chain management have been integrated as DevSecOps into DevOps practices.

Toward Broadening Nonsoftware Development Concerns in EaC

Although EaC as described previously 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 by these stakeholders is captured in code assets similar to 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, and 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, a new stakeholder with novel concerns impacting the software or its development process should adapt to the software development environment and tooling instead of developers actively providing separate views of any concern that stakeholders may raise.

In addition to enabling value to end users and other stakeholders as soon as possible, this EaC way of working centered around code, and Git respects developers' and other specialists' workflows. As software development is largely about creative work, interwinding other concerns into the development process allows developers to maintain their cadence, making them more productive.[8] In contrast, interaction among stakeholders can lead to workflow interruptions, whose resolution can delay development.[9]

Consequently, we argue that the next logical step and continuation of EaC is the expansion beyond software engineers and operators. Although this may sound radical, we have already made software developers consider the terminology of other fields, up to the point of institutionalizing it as a domain-driven design,[10] 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 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 1: Medical Compliance as Code

Before bringing products to the market, medical device manufacturers must demonstrate compliance with safe- and effective-use regulations. Traditionally, 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 Third-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 software developers and regulatory affairs professionals. The software developers move swiftly but are not accustomed to performing risk management activities during their daily routine. Likewise, regulatory affairs specialists are not accustomed to the high velocity of increments at which software developers introduce changes, sometimes incorporating SOUP components. Keeping 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 change set grows, tracking evolution becomes difficult with artifacts that potentially update several times. A more effective approach (see 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 whether a pull request introduces new SOUP components or modifies the existing ones.[11] The tool requires a software developer to justify 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 whether the change introduces new risks for which they must plan mitigation. The workflow is completed successfully only when the new SOUP component has been introduced in the software-decomposition documentation.

Lessons Learned

The compliance workflow integrated into a pull request ensures that regulatory activities are performed upon changes. The required regulatory documentation and 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 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 quickly and, helped by customized views, worked in sync with developers.

Case 2: 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. Portions of model cards can be automatically extracted and generated, but other parts require a data scientist to record design decisions and rationales manually. Therefore, model cards can form an additional task for developers to tackle and consider in their daily routines.

Sample Application: Object Detection

To assess feasibility and fitness of the model card approach in the consultancy business, we set up a continuous-delivery pipeline for an object-detection task to identify 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 Amazon Web Services' 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 an object-detection neural network with a TinyYolo[13] architecture.

As the final step of the pipeline, a model card is automatically generated, with its metadata stored in both a JavaScript Object Notation format so that a machine-readable representation can be used to generate different views, and HTML for web-based usage. We focused on creating a view

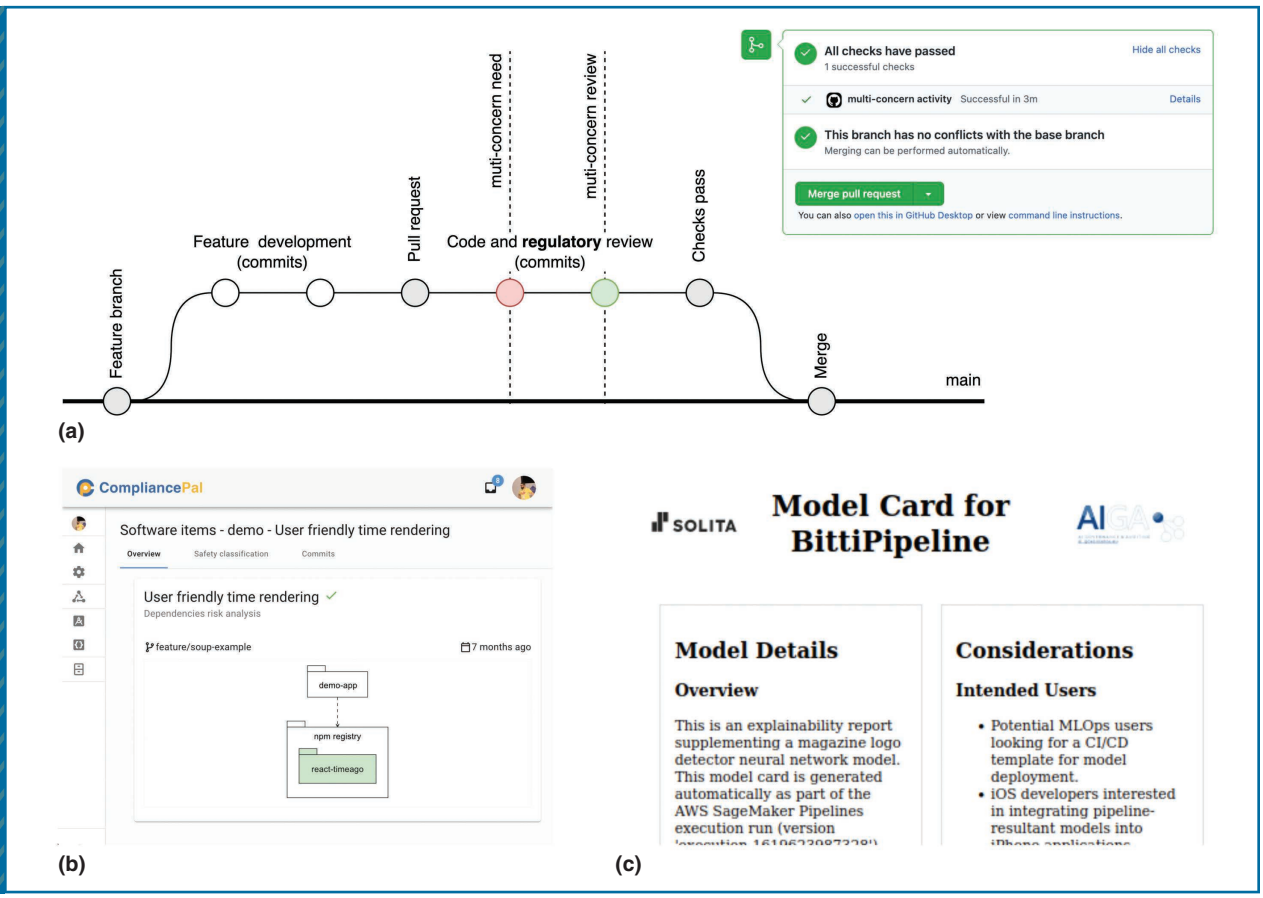


FIGURE 1. A conceptual synthesis of two cases demonstrating a pull-request flow enhanced with multiconcern views. (a) The identified and reviewed multiconcern activity, and the record of the activity being performed recorded using GitHub's pull-request check facility, acting as a quality gate; (b) the semantic-different 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. AWS: Amazon Web Services; CI/CD: continuous integration/continuous delivery; MLOps: machine learning operations.

for data scientists who were looking for other use cases for the trained model. The model card contains sections for an overview and model details, considerations for use of the model (intended use cases, limitations and tradeoffs, and ethical considerations), performance metrics and examples of 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 improves traceability of the development as all changes are tracked. Over time, this enables more people to see and validate decisions and assess 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 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 that generates model cards, and a machine-readable serialization that facilitates the consumption of model cards by scripts in ML pipelines. Although this is a step in the right direction, the proposed information is still limited to technical stakeholders, such as data scientists or ML engineers. To fulfill the model card's potential, the information should accommodate the needs of other stakeholders. The model card's 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 constructed from a single software model. A model card becomes a must-have artifact that conveys 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.

The core element that contributes to the ongoing success of DevOps practices is the collaborative culture between the software development and operations organizations.[14] Sharing information and expanding the skill set of team members, as well as performing activities together, induces a sense of shared responsibility.

The aforementioned cases demonstrate how the EaC approach can be extended to safety-critical systems and AI/ML applications for concerns of ensuring that systems are safe and effective, and that ML models do not introduce unintended side effects, respectively. As software engineers or data scientists already possess much of the knowledge needed by 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 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 collaboration of regulatory affairs specialists—generally understood as performing quality assurance tasks—with that of software engineers and data scientists, a new multidisciplinary team improves the quality outcome in compliance and AI explainability. The results, which are in line with expectations of DevOps adoption,[15] provide practical examples of how EaC is indeed able to handle the concerns of noncoder stakeholders while at the same time do not interfere significantly with developers, who are able to maintain their high delivery pace and way of working.

References

1. *Medical Device Software–Software Life-Cycle Processes*, , International Electrotechnical Commission, London, U.K., 2015.
2. P. Voigt and A. V. d. Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*, vol. 10, 1st ed. Cham: Springer International Publishing, 2017, p. 3,152,676.
3. L. Floridi, "Establishing the rules for building trustworthy AI," *Nature Mach. Intell.*, vol. 1, no. 6, pp. 261–262, 2019, doi: 10.1038/s42256-019-0055-y.
4. S. Larsson, "AI in the EU: Ethical guidelines as a governance tool,"

ABOUT THE AUTHORS

VLAD STIRBU is the founder of CompliancePal, Tampere, 33820, Finland. His research interests include continuous software engineering practices in the context of regulated industries. Stirbu received his D.Sc. (Tech.) in software engineering from Tampere University of Technology. Contact him at vlad.stirbu@compliancepal.eu.

VLAS SOKOLOV is a data scientist working for Solita Munich, 81247, Germany. His research interests include object detection and software automation. Sokolov received his Ph.D. in astrophysics from Ludwig-Maximilians-Universität München. Contact him at vlas.sokolov@solita.fi.

MIKKO RAATIKAINEN is a researcher focusing on software engineering and business at University of Helsinki, Helsinki, 00014, Finland. Raatikainen received his D.Sc. (Tech.) in software engineering from Aalto University. Contact him at mikko.raatikainen@helsinki.fi.

TIMO LEHTONEN is a senior software designer at Solita Tampere, 33800, Finland. His research interests include lean software development and data analytics. Lehtonen received his Ph.D. from Tampere University of Technology. Contact him at timo.lehtonen@solita.fi.

JOEL RÖNTYNEN is a data scientist and quantum software engineer for Solita, Helsinki, 00170, Finland. His research interests include machine learning and topological phases of matter. Röntynen received his Ph.D. in theoretical physics from Aalto University School of Science and Technology. Contact him at joel.rontynen@solita.fi.

TOMMI MIKKONEN is a professor of software engineering at the University of Jyväskylä, Jyväskylä, 40100, Finland. His research interests include software engineering for artificial intelligence systems and continuous delivery of software systems. Mikkonen received his doctoral degree in software engineering from Tampere University of Technology. Contact him at tommi.j.mikkonen@jyu.fi.

in *The European Union and the Technology Shift*, A. B. Engelbrekt, K. Leijon, A. Michalski, and L. Oxelheim Eds. Cham: Palgrave Macmillan, 2021, pp. 85–111.

5. A. Ahmad Zeeshan, "Automating everything as code," in *DevSecOps for .NET Core*, New York, NY, USA: Springer-Verlag, 2020, pp. 109–162.
6. C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, 2016, doi: 10.1109/MS.2016.68.
7. L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devOps concepts and challenges," *ACM Comput. Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019, doi: 10.1145/3359981.
8. K. Kuusinen, H. Petrie, F. Fagerholm, and T. Mikkonen, "Flow, intrinsic motivation, and developer experience in software engineering," in *Agile Processes in Software Engineering and Extreme Programming*, H. Sharp

and T. Hall, Eds. Cham: Springer International Publishing, 2016, pp. 104–117.
9. K. Kati, "Value creation and delivery in agile software development: Overcoming stakeholder conflicts," in *Proc. IFIP Conf. Human-Comput. Interaction*, 2017, pp. 123–129, doi: 10.1007/978-3-319-92081-8_12.
10. E. Evans and E. J. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Reading, MA, USA: Addison-Wesley, 2004.
11. V. Stirbu and T. Mikkonen, "CompliancePal: A tool for supporting practical agile and regulatory-compliant development of medical software," in *Proc. 2020 IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, pp. 151–158, doi: 10.1109/ICSA-C50368.2020.00035.
12. M. Mitchell *et al.*, "Model cards for model reporting," in *Proc. Conf. Fairness, Accountability, Transparency*, 2019, pp. 220–229, doi: 10.1145/3287560.3287596.
13. I. Khokhlov *et al.*, "Tiny-yolo object detection supplemented with geometrical data," in *Proc. 2020 IEEE 91st Veh. Technol. Conf. (VTC2020-Spring)*, pp. 1–5, doi: 10.1109/VTC2020-Spring 48590.2020.9128749.
14. L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of DevOps," in *Agile Processes in Software Engineering and Extreme Programming*, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds. Cham: Springer International Publishing, 2015, pp. 212–217.
15. W. P. Luz, G. Pinto, and R. Bonifácio, "Adopting devOps in the real world: A theory, a model, and a case study," *J. Syst. Softw.*, vol. 157, p. 110,384, Nov. 2019, doi: 10.1016/j.jss.2019.07.083.

Digital Object Identifier 10.1109/MS.2022.3177876