

A process for continuous consistency-aware quality management

Martin Armbruster¹, Manar Mazkatli¹ and Anne Koziolk¹

¹KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany

Abstract

In iterative and agile software development, fast development cycles can outdate software models quickly, hindering their application for architecture-based quality predictions. Therefore, in this short paper, we present our envisioned generalized process to continuously manage the consistency between a software project's artifacts, the running software, and software models, enabling quality predictions and round-trip engineering.

Keywords

Consistency Management, Software Architecture, Quality Predictions, Consistency Maintenance

1. Introduction

In iterative and agile software development processes, development teams usually employ DevOps practices, including Continuous Integration (CI) and Continuous Delivery (CD) pipelines, for fast deployments and fast feedback [1]. Architecture-based quality predictions can support the proactive and cost-efficient assessment of design alternatives in this context [2]. There are several approaches for architecture-based quality predictions for different quality attributes [3], such as performance [2], reliability [2], availability [4], confidentiality [5], and aspects of dependability [6, 7]. However, these approaches assume existing non-changing software architecture models.

Due to the fast changes in iterative software development [1], software architecture models can get outdated fast, hindering their application. Furthermore, continuous updates of architecture models are labor-intensive when performed manually. Multiple approaches aim at getting an up-to-date architecture model, either by batch-wise or incremental reverse engineering [8], continuously [9, 10], or with incorporated quality predictions for specific quality attributes [9, 11, 12, 13]. Nevertheless, these approaches only target software architectures without quality predictions or target specific quality attributes, too.

Thus, the question raises: How can architecture-based quality predictions in general be utilized in DevOps environments? In this short paper, we present our envisioned generalized process for the Continuous Consistency-aware Quality Management (CCQM). By maintaining the consistency between different software models, including the software architecture, a software's artifacts, and the running software throughout the development and operation of a software, the CCQM process enables continuous architecture-based quality predictions for quality management.

The reminder of this paper is structured as follows. In section 2, we present our envisioned CCQM process. Then, we conclude the paper in section 3 with a short outlook to a planned evaluation.

ICMM 2025: (In-)Consistency Management in Modeling, co-located with STAF 2025, 10–13 June 2025, Koblenz, Germany

✉ martin.armbruster@kit.edu (M. Armbruster); manar.mazkatli@kit.edu (M. Mazkatli); koziolk@kit.edu (A. Koziolk)

🌐 https://mcse.kastel.kit.edu/staff_martin_armbruster.php (M. Armbruster);

https://mcse.kastel.kit.edu/staff_Mazkatli_Manar.php (M. Mazkatli); https://mcse.kastel.kit.edu/staff_Koziolk_Anne.php (A. Koziolk)

🆔 0000-0002-2554-4501 (M. Armbruster); 0000-0003-4261-8477 (M. Mazkatli); 0000-0002-1593-3394 (A. Koziolk)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

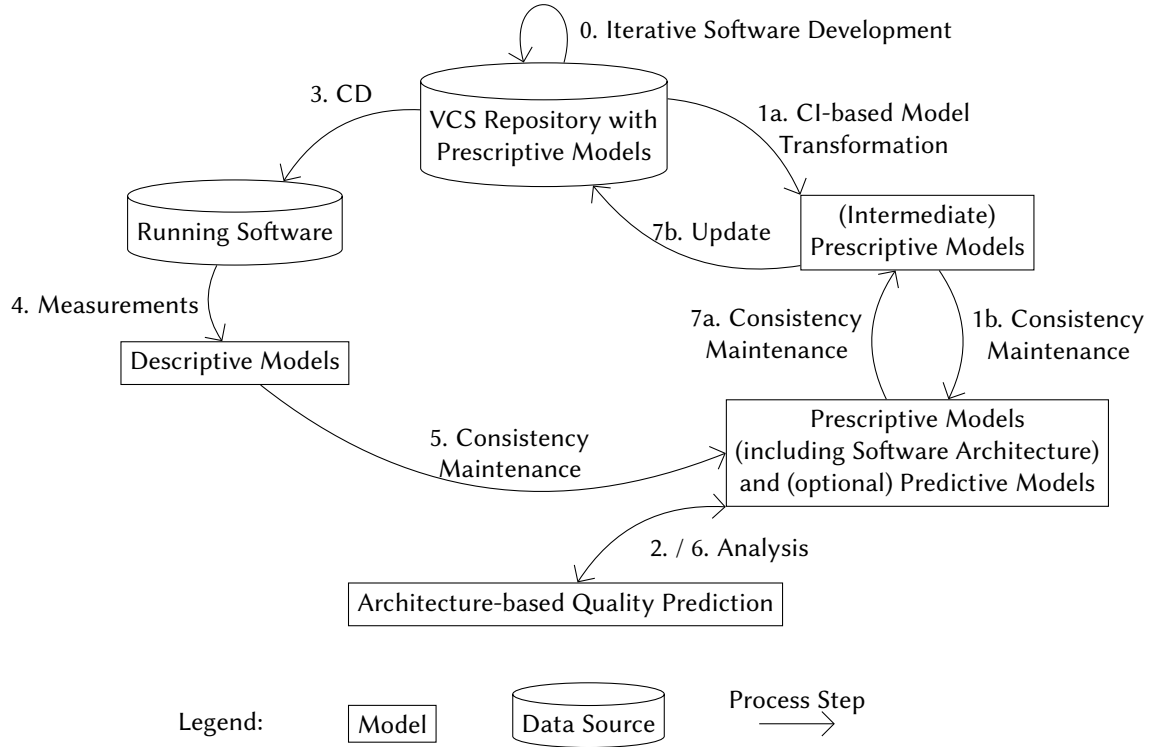


Figure 1: A generalized process for the continuous consistency management to enable architecture-based quality predictions.

2. Continuous Consistency Management for Architecture-based Quality Prediction

In this section, we present our envisioned generalized process for CCQM to continuously manage the consistency between software models, enabling architecture-based quality predictions throughout DevOps practices. An overarching goal is to have incremental and automated process steps as much as possible to reduce the effort and barrier for software developers. We base the process partly on the Continuous Integration of architectural Performance Models (CIPM) approach [9], to which we refer for an example realization of the CCQM process, and the MODA framework [14]. Hence, we differentiate software models according to the three following roles [14]:

- **Prescriptive Models**, which encompass the structure, behavior, and operational context of the software. In our context, these models include source code, deployment descriptors, configuration files, software architecture, and others.
- **Predictive Models**, used in combination with prescriptive models to predict quality attributes in alternative and future states. They can be realized by, for instance, annotations in prescriptive models.
- **Descriptive Models**, which represent a software's runtime and a software at runtime. Here, we consider descriptive models solely as models of measurements (e.g., performance metrics, traces, or logs) taken from the production environment by monitoring the running software.

Figure 1 depicts the CCQM process, its steps, and how the models in their different roles are connected. The numbering of the process steps presents one possible sequence. Nevertheless, in reality, the different steps can be executed in varying orders and concurrently. In the following, we describe the process steps, grouped in four sub-processes for clarity, on a rather abstract level to allow adaptations, potentially needed for concrete realizations.

CI-based Consistency during Software Development CCQM during iterative software development (step 0) mainly concerns the consistency between prescriptive models on different abstraction levels. Thus, in step 1a, a CI pipeline reads the recent changes from a version control system (VCS) repository. Even though the artifacts in the VCS repository (e.g., source code or deployment descriptors) are prescriptive models, we do not directly utilize them for the consistency maintenance. Instead, we automatically parse and transform the artifacts and their changes so that all prescriptive models can conform to the same meta-metamodel (e.g., ecore). The resulting models can be called intermediate, bridging the gap between the artifacts in the VCS repository and other prescriptive models. In addition, this allows easier tracing of related model elements. The execution of step 1a does not necessarily need to be accomplished after every commit in the VCS repository. Software developers can schedule the execution (e.g., after a certain number of commits) or trigger it manually.

The changes of the (intermediate) models trigger the consistency maintenance in step 1b towards the other prescriptive models, mainly the software architecture. This step employs change-based, incremental model transformations to update model parts that are affected by the changes in the VCS repository. These transformations can be defined as rules on the metamodel level, analogous to the CIPM approach [9]. Step 1b should also be fully automated to be able to run in a CI pipeline. However, there are cases, in which input from software developers or architects is required. Thus, open questions remain: How can manual actions be separated from a CI-based execution, to which extent are manual actions required, and how can the manual effort be reduced? As the (intermediate) models can contain information relevant for the predictive models or quality predictions, step 1b can also target predictive models, specific parts of them, or annotations to be able to preserve these relevant information. In this context, an open question is how targeted quality attributes influence the consistency maintenance and preserved information. Steps 1a and 1b represent the reverse engineering of the software architecture [15]. If there is already a software architecture, it can be either incorporated into the prescriptive models [10] or created from scratch by the CCQM process (e.g., for checking the consistency between the existing and newly created software architecture). Based on the consistent software models from steps 1a and 1b, architecture-level quality predictions can be performed (step 2).

Consistency during Software Operation During software operation, starting with the CD pipeline (step 3), the CCQM process aims to automatically maintain the consistency between the running software and its models. Therefore, measurements from the running software are incorporated into the descriptive models (step 4). An open question is how to connect existing monitoring tools and infrastructure with this step. Since we consider here only models of measurements as descriptive models for their explicit modeling, we assume that the measurements are only appended to the descriptive models, which do not change otherwise. Afterward, the actual consistency maintenance can be automatically executed to update corresponding prescriptive and (optionally) predictive models (step 5), for example, performance parameters in CIPM [9]. It is configurable when step 5 is triggered, analogous to step 1a. Beside manual executions, automatic triggers encompass specific changes and thresholds for certain change characteristics, for instance, when a predefined time frame or number of changes has passed. As a consequence of the consistency maintenance, the software architecture can reflect the production environment accurately, and quality predictions based on an up-to-date state of the production environment can be performed (step 6). While it seems that there could be conflicts or collisions when maintaining the consistency between software models with changes from development and operation, the software models, the software architecture, in particular, or the CCQM process should be able to handle and reflect the different aspects by, for instance, providing multiple versions and variants of the models. This enables analyses for different development and operation states.

Forward Engineering during Software Development As last step, the CCQM process allows changing the prescriptive models, especially the software architecture, in response to prediction results (automatically) or software evolution (manually). With these changes, the consistency maintenance in step 7a can semi-automatically update the prescriptive models (e.g., source code models), resulting in

forward engineering [15]. To close the process, the artifacts in the VCS repository are changed according to the updated intermediate prescriptive models (step 7b) by automatic model transformations. Although the CCQM process contains both forward and reverse engineering, effectively enabling round-trip engineering [16], it is also possible to only realize one, either reverse or forward engineering.

Architecture-based Quality Predictions The most important part of the CCQM process is the architecture-based quality predictions in step 2 and step 6. All previous steps ensure that the required software architecture models and optionally relevant prescriptive, predictive, or descriptive model (parts) are consistent with the running software or development state. Using these consistent models, existing analyses and prediction approaches can be applied. As their concrete application varies, we do not further specify steps 2 and step 6. As an example, the performance predictions in the CIPM approach directly operate on the software architecture, enriched with performance parameters [9]. In contrast, an approach by Cortellessa et al. [4] for availability defines a round-trip process: the software architecture is transformed into a specific analysis model, from which the analysis results can be transferred back to the architecture.

3. Conclusion and Outlook

In this short paper, we presented a generalized process for the continuous consistency management of software models, enabling architecture-based quality predictions throughout DevOps practices. During software development, the CCQM process ensures the consistency between artifacts in a VCS repository and mainly the software architecture. During software operation, the CCQM process maintains the consistency between the running software and its models. Based on the consistent models, architecture-based quality predictions can be performed.

In future work, we plan to evaluate our envisioned CCQM process regarding its (1) suitability and (2) applicability for different architecture-based quality predictions. For the (1) suitability, we want to conduct interviews with researchers in the domain of quality predictions to find out how well analyses can be embedded into the CCQM process and which (potential) additional features need to be considered. While we have a concrete realization of the CCQM process within the CIPM approach for performance [9], we plan an additional application of the CCQM process for a different quality attribute.

Acknowledgments

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263, and by KASTEL Security Research Labs.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles, A survey of devops concepts and challenges, *ACM Comput. Surv.* 52 (2019). URL: <https://doi.org/10.1145/3359981>. doi:10.1145/3359981.
- [2] A. Martens, H. Koziol, S. Becker, R. Reussner, Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, in: *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering, WOSP/SIPEW '10*, Association for Computing Machinery, New York, NY, USA, 2010, p. 105–116. URL: <https://doi.org/10.1145/1712605.1712624>. doi:10.1145/1712605.1712624.

- [3] L. Grunske, Early quality prediction of component-based systems – a generic framework, *Journal of Systems and Software* 80 (2007) 678–686. URL: <https://www.sciencedirect.com/science/article/pii/S0164121206002238>. doi:<https://doi.org/10.1016/j.jss.2006.08.014>, component-Based Software Engineering of Trustworthy Embedded Systems.
- [4] V. Cortellessa, R. Eramo, M. Tucci, From software architecture to analysis models and back: Model-driven refactoring aimed at availability improvement, *Information and Software Technology* 127 (2020) 106362. URL: <https://www.sciencedirect.com/science/article/pii/S0950584920301300>. doi:<https://doi.org/10.1016/j.infsof.2020.106362>.
- [5] S. Seifermann, R. Heinrich, D. Werle, R. Reussner, Detecting violations of access control and information flow policies in data flow diagrams, *Journal of Systems and Software* 184 (2022) 111138. URL: <https://www.sciencedirect.com/science/article/pii/S0164121221002351>. doi:<https://doi.org/10.1016/j.jss.2021.111138>.
- [6] S. Bernardi, J. Merseguer, D. C. Petriu, *Model-Driven Dependability Assessment of Software Systems*, Springer Berlin, Heidelberg, Heidelberg, 2013. URL: <https://doi.org/10.1007/978-3-642-39512-3>.
- [7] S. Mustafiz, X. Sun, J. Kienzle, H. Vangheluwe, Model-driven assessment of system dependability, *Software & Systems Modeling* 7 (2008) 487–502. URL: <https://doi.org/10.1007/s10270-008-0084-1>. doi:<https://doi.org/10.1007/s10270-008-0084-1>.
- [8] A. Qayum, M. Zhang, S. Colreavy, M. Chochlov, J. Buckley, D. Lin, A. R. Sai, A framework and taxonomy for characterizing the applicability of software architecture recovery approaches: A tertiary-mapping study, *Software: Practice and Experience* 55 (2025) 100–132. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3364>. doi:<https://doi.org/10.1002/spe.3364>. arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3364>.
- [9] M. Mazkatli, D. Monschein, M. Armbruster, R. Heinrich, A. Koziolk, Continuous integration of architectural performance models with parametric dependencies – the cipm approach, *Automated Software Engineering Journal* (2025). doi:10.1007/s10515-025-00521-9, a preprint is also available in KITopen: <https://doi.org/10.5445/IR/1000151086/v3>.
- [10] M. Langhammer, *Automated Coevolution of Source Code and Software Architecture Models*, Ph.D. thesis, Karlsruher Institut für Technologie (KIT), 2017. doi:10.5445/IR/1000069366.
- [11] V. Cortellessa, D. Di Pompeo, R. Eramo, M. Tucci, A model-driven approach for continuous performance engineering in microservice-based systems, *Journal of Systems and Software* 183 (2022) 111084. URL: <https://www.sciencedirect.com/science/article/pii/S0164121221001813>. doi:<https://doi.org/10.1016/j.jss.2021.111084>.
- [12] S. Spinner, J. Grohmann, S. Eismann, S. Kounev, Online model learning for self-aware computing infrastructures, *Journal of Systems and Software* 147 (2019) 1–16. URL: <https://www.sciencedirect.com/science/article/pii/S0164121218302188>. doi:<https://doi.org/10.1016/j.jss.2018.09.089>.
- [13] G. Ganea, I. Verebi, R. Marinescu, Continuous quality assessment with incode, *Science of Computer Programming* 134 (2017) 19–36. URL: <https://www.sciencedirect.com/science/article/pii/S0167642315000520>. doi:<https://doi.org/10.1016/j.scico.2015.02.007>, 6th issue of Experimental Software and Toolkits (EST-6).
- [14] B. Combemale, J. Kienzle, G. Mussbacher, H. Ali, D. Amyot, M. Bagherzadeh, E. Batot, N. Bencomo, B. Benni, J.-M. Bruel, J. Cabot, B. H. Cheng, P. Collet, G. Engels, R. Heinrich, J.-M. Jezequel, A. Koziolk, S. Mosser, R. Reussner, H. Sahraoui, R. Saini, J. Sallou, S. Stinckwich, E. Syriani, M. Wimmer, A hitchhiker’s guide to model-driven engineering for data-centric systems, *IEEE Software* 38 (2021) 71–84. doi:10.1109/MS.2020.2995125.
- [15] E. Chikofsky, J. Cross, Reverse engineering and design recovery: a taxonomy, *IEEE Software* 7 (1990) 13–17. doi:10.1109/52.43044.
- [16] T. Hettel, M. Lawley, K. Raymond, Model synchronisation: Definitions for round-trip engineering, in: A. Vallecillo, J. Gray, A. Pierantonio (Eds.), *Theory and Practice of Model Transformations*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 31–45.